Service Availability™ Forum Application Interface Specification

Information Model Management Service SAI-AIS-IMM-A.01.01

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

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

SERVICE AVAILABILITY™ FORUM SPECIFICATION LICENSE AGREEMENT

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Table of Contents Information Model Management Service

1 Document Introduction 9

1.1 Document Purpose 9
1.2 AIS Documents Organization 9
1.3 History 9
1.4 References 9
1.5 How to Provide Feedback on the Specification 9
1.6 How to Join the Service Availability™ Forum 10
1.7 Additional Information 10
1.7.1 Member Companies 10
1.7.2 Press Materials 10

2 Overview 11

2.1 Information Model Management Service 11

3 Information Model Management Service API 13

4.0 IMM Service - Object Management API Specification 17

4.1 Include File and Library Name 17
4.2 Type Definitions 17
4.2.1 Handles Used by the IMM Service 17
4.2.2 Various IMM Service Names 17
4.2.3 SaImmValueTypeT 18
4.2.4 SaImmClassCategoryT 18
4.2.5 SaImmAttrFlagsT 19
4.2.6 SaImmAttrValueT 20
4.2.7 SaImmAttrDefinitionT 20
4.2.8 SaImmAttrValuesT 21
4.2.9 SaImmAttrModificationTypeT 21
4.2.10 SaImmAttrModificationT 22
4.2.11 SaImmScopeT 22
4.2.12 SaImmSearchOptionsT 23
4.2.13 SaImmSearchParametersT 23
4.2.14 SaImmCcbFlagsT 24
4.2.15 SaImmAdminOperationIdT 25
4.2.16 SaImmAdminOperationParamsT 25
4.2.17 SaImmCallbacksT 25
4.2.18 IMM Service Object Attributes 26
4.3 Library Life Cycle 27
4.3.1 saImmOmInitialize() 27
4.3.2 saImmOmSelectionObjectGet() 29
4.3.3 saImmOmDispatch() 31

4.3.4 saImmOmFinalize() 32
4.4 Object Class Management 33
4.4.1 saImmOmClassCreate() 33
4.4.2 saImmOmClassDescriptionGet() 35
4.4.3 saImmOmClassDescriptionMemoryFree() 36
4.4.4 saImmOmClassDelete() 37
4.5 Object Search 38
4.5.1 saImmOmSearchInitialize() 39
4.5.2 saImmOmSearchNext() 41
4.5.3 saImmOmSearchFinalize() 43
4.6 Object Access 43
4.6.1 saImmOmAccessorInitialize() 44
4.6.2 saImmOmAccessorGet() 45
4.6.3 saImmOmAccessorFinalize() 47
4.7 Object Administration Ownership 47
4.7.1 saImmOmAdminOwnerInitialize() 48
4.7.2 saImmOmAdminOwnerSet() 50
4.7.3 saImmOmAdminOwnerRelease() 52
4.7.4 saImmOmAdminOwnerFinalize() 54
4.7.5 saImmOmAdminOwnerClear() 55
4.8 Configuration Changes 56
4.8.1 saImmOmCcbInitialize() 57
4.8.2 saImmOmCcbObjectCreate() 58
4.8.3 saImmOmCcbObjectDelete() 61
4.8.4 saImmOmCcbObjectModify() 63
4.8.5 saImmOmCcbApply() 65
4.8.6 saImmOmCcbFinalize() 66
4.9 Administrative Operations Invocation 68
4.9.1 saImmOmAdminOperationInvoke(), saImmOmAdminOperationInvokeAsync() 68
4.9.2 SaImmOmAdminOperationInvokeCallbackT 71

5.0 IMM Service - Object Implementer API Specification 73

5.1 Include File and Library Name 73
5.2 Type Definitions 73
5.2.1 IMM Service Handle 73
5.2.2 SaImmOiImplementerNameT 73
5.2.3 SaImmOiCcbIdT 73
5.2.4 SaImmOiCallbacksT 73
5.3 Library Life Cycle 74
5.3.1 saImmOiInitialize() 74
5.3.2 saImmOiSelectionObjectGet() 77
5.3.3 saImmOiDispatch() 78
5.3.4 saImmOiFinalize() 79
5.4 Object Implementer 80

5.4.1 saImmOiImplementerSet() 81
5.4.2 saImmOiImplementerClear() 82
5.4.3 saImmOiClassImplementerSet() 83
5.4.4 saImmOiClassImplementerRelease() 85
5.4.5 saImmOiObjectImplementerSet() 86
5.4.6 saImmOiObjectImplementerRelease() 88
5.5 Runtime Objects Management 90
5.5.1 saImmOiRtObjectCreate() 91
5.5.2 saImmOiRtObjectDelete() 93
5.5.3 saImmOiRtObjectUpdate() 94
5.5.4 SaImmOiRtAttrUpdateCallbackT 96
5.6 Configuration Objects Implementer 97
5.6.1 SaImmOiCcbObjectCreateCallbackT 99
5.6.2 SaImmOiCcbObjectDeleteCallbackT 100
5.6.3 SaImmOiCcbObjectModifyCallbackT 101
5.6.4 SaImmOiCcbCompletedCallbackT 103
5.6.5 SaImmOiCcbApplyCallbackT 104
5.6.6 SaImmOiCcbAbortCallbackT 105
5.7 Administrative Operations 106
5.7.1 SaImmOiAdminOperationCallbackT 106
5.7.2 saImmOiAdminOperationResult() 107

1 Document Introduction

1.1 Document Purpose

This document defines the Information Model Management Service of the Application Interface Specification (AIS) of the Service Availability™ Forum (SA Forum). It is intended for use by implementers of the Application Interface Specification and by application developers who would use the Application Interface Specification to develop applications that must be highly available. The AIS is defined in the C programming language, and requires substantial knowledge of the C programming language.

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

1.2 AIS Documents Organization

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

1.3 History

SAI-AIS-IMM-A.01.01 is the first release of the Information Model Management Service specification.

1.4 References

The following document contains information that is relevant to the specification:

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

1.5 How to Provide Feedback on the Specification

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

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

1.6 How to Join the Service Availability™ Forum

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

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

1.7 Additional Information

1.7.1 Member Companies

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

1.7.2 Press Materials

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

2 Overview

This specification defines the Information Model Management Service within the Application Interface Specification (AIS).

2.1 Information Model Management Service

The different entities of an SA Forum cluster, such as components provided by the Availability Management Framework, checkpoints provided by the Checkpoint Service, or message queues provided by the Message Service are represented by various objects of the SA Forum information model.

The SA Forum information model (IM) is specified in UML and managed by the Information Model Management (IMM) Service.

The objects in the Information Model are provided with their attributes and administrative operations (i.e., operations that can be performed on the represented entities through system management interfaces). For management applications or Object Managers, the IMM provides the APIs to create, access and manage these objects.

Subsequently, it delivers the requested operations to the appropriate AIS services or applications (referred to as **Object Implementers**) that implement these objects for execution.

Information Model objects and attributes can be classified into two categories:

- Configuration objects and attributes
- Runtime objects and attributes

The IMM Service exposes two sets of APIs:

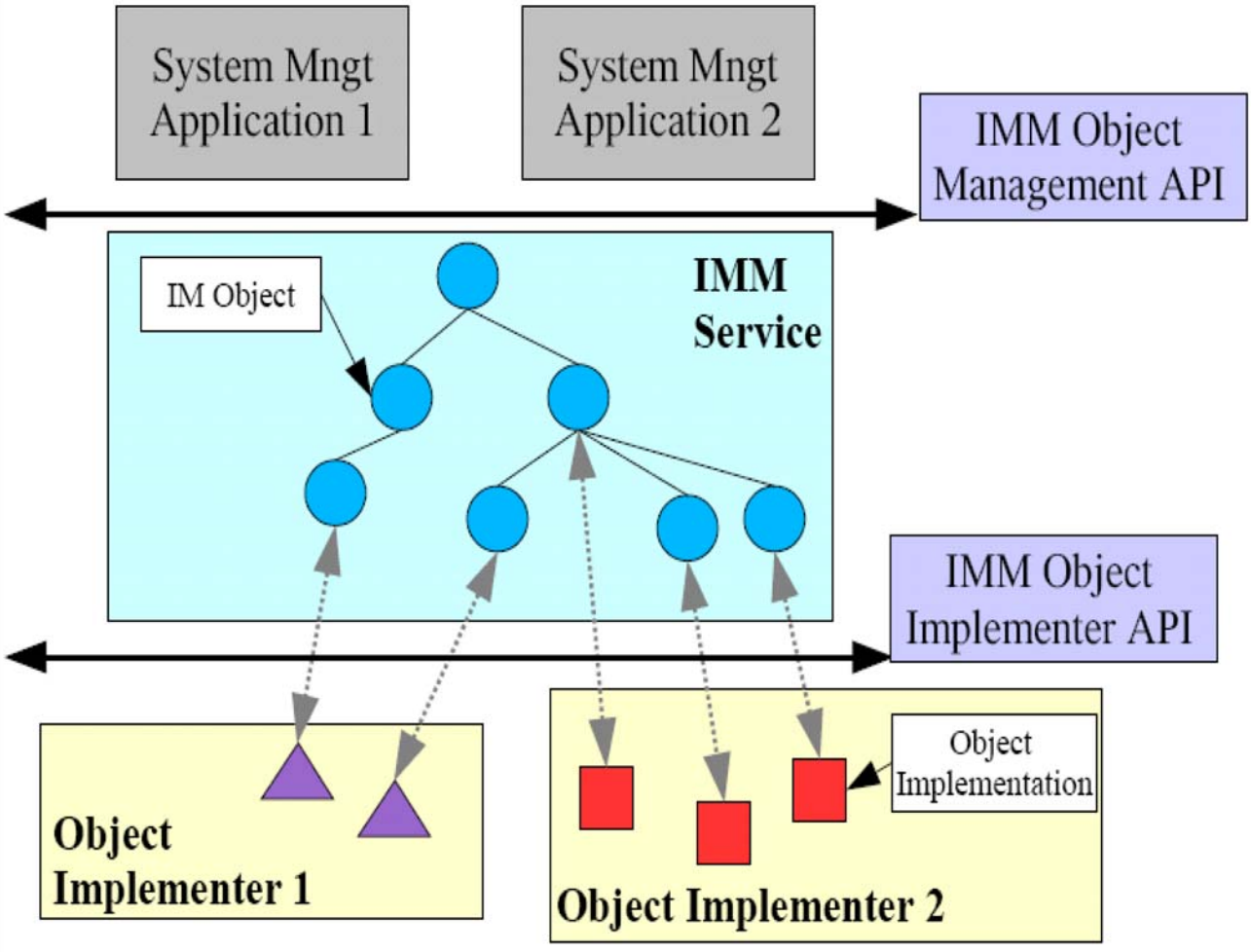
(1) An Object Management API (OM-API) exposed typically to System Management applications (for example, SNMP agents and CIM providers).

(2) An Object Implementer API (OI-API) restricted to Object Implementers.

3 Information Model Management Service API

The Service Availability™ Forum (SA Forum) information model (IM) is specified in UML and represents the various objects that constitute an SA Forum system. The SA Forum IM also specifies the attributes of these objects and administrative operations that can be performed on the entities through system management interfaces.

The Information Model Management (IMM) Service is the SA Forum service managing all objects of the SA Forum Information Model and provides the APIs to access and manage these objects.



The actual implementation of objects represented in the information model is not part of the IMM Service but is provided by user applications or other SA Forum services such as Availability Management Framework, Checkpoint Service, etc.

SA Forum services and applications that implement the IMM objects are called **Object Implementers** in the rest of this document.

IMM objects are organized in a tree hierarchy. The hierarchy follows the structure of the LDAP distinguished name of each object. Refer to the SA Forum Overview document ([1]) for more information about LDAP object names.

IMM objects and attributes can be classified into two categories:

- Configuration Objects and Attributes

  Configuration objects and attributes are the means by which system management applications provide input on the desired sets of objects and their handling that an Object Implementer should implement. The set of configuration objects and attributes constitute the prescriptive part of the information model.

  Configuration objects and attributes are typically under the control of system management applications. They are of a persistent nature and must survive a full cluster power-off.

  Configuration attributes are read-write attributes from an Object Management perspective but read-only from an Object Implementer perspective.

- Runtime Objects and Attributes

  Runtime objects and attributes are the means by which Object Implementers reflect in the information model the current state of the objects they implement. The set of runtime objects and attributes constitute the descriptive part of the information model. Runtime objects and attributes are typically under the control of Object Implementers.

  Runtime objects, which contain persistent runtime attributes are persistent and must survive a full cluster power-off. Non-persistent runtime attributes do not survive a full cluster power-off.

  Runtime attributes are read-only attributes from an Object Management perspective but read-write from an Object Implementer perspective.

As attributes cannot exist outside of an encapsulating object, configuration attributes can only belong to configuration objects as opposed to runtime attributes that may belong to objects of either category. Runtime objects can only have runtime attributes.

After a full cluster power-off, during its initialisation step, the IMM Service automatically re-creates all persistent objects with their persistent attributes from the copy it maintains on stable storage.

Object Implementers cannot on their own initiative create and delete configuration objects or modify configuration attributes through the Object Implementer interface. On the other hand, system management applications cannot directly create and delete runtime objects or modify runtime attributes. However, as a consequence of some administrative operations requested by these system management applications Object Implementers may create or delete runtime objects or modify runtime attributes to reflect the new system state after the completion of the administrative operation.

The IMM Service exposes two sets of APIs:

(1) An Object Management API (OM-API) exposed typically to system management applications (for example, SNMP agents and CIM providers).

(2) An Object Implementer API (OI-API) restricted to Object Implementers.

Chapter 4.0 (next chapter) describes the OM-API. The OI-API is found in Chapter 5.0.

4.0 IMM Service - Object Management API Specification

4.1 Include File and Library Name

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

#include <saImmOm.h>

To use the IMM Service Object Management API, an application must be bound with the following library:

libSaImmOm.so

4.2 Type Definitions

The Information Model Management Service uses the types described in the following sections.

4.2.1 Handles Used by the IMM Service

typedef SaUint64T SaImmHandleT;

typedef SaUint64T SaImmAdminOwnerHandleT;

typedef SaUint64T SaImmCcbHandleT;

typedef SaUint64T SaImmSearchHandleT;

typedef SaUint64T SaImmAccessorHandleT;

The acronym CCB stands for **Configuration Changes Bundle**.

4.2.2 Various IMM Service Names

The following types represent object class names, administrative owner names and object class attribute names. All these names are UTF-8 encoded character strings terminated by the NULL character.

```
typedef SaStringT SaImmClassNameT;
typedef SaStringT SaImmAttrNameT;
typedef SaStringT SaImmAdminOwnerNameT;
```

4.2.3 SaImmValueTypeT

SaImmValueTypeT contains various data types used by the IMM Service for class attributes and administrative operation parameters.

```
typedef enum {
    SA_IMM_ATTR_SAINT32T = 1,        /* SaInt32T */
    SA_IMM_ATTR_SAUINT32T = 2,       /* SaUint32T */
    SA_IMM_ATTR_SAINT64T = 3,        /* SaInt64T */
    SA_IMM_ATTR_SAUINT64T = 4,       /* SaUint64T */
    SA_IMM_ATTR_SATIMET = 5,         /* SaTimeT */
    SA_IMM_ATTR_SANAMET = 6,         /* SaNameT */
    SA_IMM_ATTR_SAFLOATT = 7,        /* SaFloatT */
    SA_IMM_ATTR_SADOUBLET = 8,       /* SaDoubleT */
    SA_IMM_ATTR_SASTRINGT = 9,       /* SaStringT */
    SA_IMM_ATTR_SAANYT = 10          /* SaAnyT */
} SaImmValueTypeT;
```

4.2.4 SaImmClassCategoryT

SaImmClassCategoryT is used to distinguish among different categories of object classes.

```
typedef enum {
    SA_IMM_CLASS_CONFIG = 1,
    SA_IMM_CLASS_RUNTIME = 2
} SaImmClassCategoryT;
```

The values of *SaImmClassCategoryT* indicate whether the object class is a configuration object class or a runtime object class.

4.2.5 SaImmAttrFlagsT

SaImmAttrFlagsT is used to specify the various characteristics of an attribute of an object class.

```
#define SA_IMM_ATTR_MULTI_VALUE0x00000001
#define SA_IMM_ATTR_RDN         0x00000002
#define SA_IMM_ATTR_CONFIG      0x00000100
#define SA_IMM_ATTR_WRITABLE    0x00000200
#define SA_IMM_ATTR_INITIALIZED 0x00000400
#define SA_IMM_ATTR_RUNTIME     0x00010000
#define SA_IMM_ATTR_PERSISTENT  0x00020000
#define SA_IMM_ATTR_CACHED      0x00040000

typedef SaUint64T SaImmAttrFlagsT;
```

The meaning of the flags listed above is:

- SA_IMM_ATTR_MULTI_VALUE: If this flag is specified, the attribute is a multi-value attribute; otherwise, the attribute is a single-value attribute.
- SA_IMM_ATTR_RDN: The attribute is used as the Relative Distinguished Name (RDN) for the containing object. Each object class must have one and only one RDN attribute. This attribute must be a single-value attribute and may not be modified after the object is created. The RDN attribute of a configuration object must be a configuration attribute.

The following two attributes are mutually exclusive as an attribute is either a configuration or a runtime attribute.

- SA_IMM_ATTR_CONFIG: The attribute is a configuration attribute. Configuration attributes are only allowed within object classes of the SA_IMM_CLASS_CONFIG category.
- SA_IMM_ATTR_RUNTIME: The attribute is a runtime attribute. Runtime attributes can belong to all object class categories.

The following two attributes are only meaningful for configuration attributes. Setting them for runtime attributes is not allowed and generates an error.

- SA_IMM_ATTR_WRITABLE: Setting this flag for a configuration attribute indicates that the attribute can be modified. If the flag is not present, the configura-

tion attribute can only be set when the object is created and cannot be modified or deleted later on.

- SA_IMM_ATTR_INITIALIZED: Setting this flag for a configuration attribute indicates that a value must be specified for this attribute when the object is created. This flag may not be set in the definition of a configuration attribute, which includes a default value for the attribute.

The following attributes are only meaningful for runtime attributes. Setting them for configuration attributes is not allowed and generates an error.

- SA_IMM_ATTR_PERSISTENT: Setting this flag for runtime attributes indicates that the attribute must be stored in a persistent manner by the IMM Service. If a runtime object has persistent attributes, or if one of its children has persistent attributes, its RDN attribute must be persistent.
- SA_IMM_ATTR_CACHED: Setting this flag for a runtime attribute indicates that the value of the attribute must be cached by the IMM Service.

4.2.6 SaImmAttrValueT

SaImmAttrValueT is used to represent the values of object attributes.

```
typedef void *SaImmAttrValueT;
```

4.2.7 SaImmAttrDefinitionT

SaImmAttrDefinitionT is used to specify the characteristics of an attribute belonging to a particular object class.

```
typedef struct {
    SaImmAttrNameT attrName;
    SaImmValueTypeT attrValueType;
    SaImmAttrFlagsT attrFlags;
    SaUint32T attrNtfId;
    SaImmAttrValueT attrDefaultValue;
} SaImmAttrDefinitionT;
```

The various fields of the structure above have the following usage:

- *attrName*: contains the attribute name.
- *attrValueType*: indicates what type of values can be assigned to this attribute.
- *attrFlags*: contains additional characteristics of this attribute.

- *attrNtfId*: identifier used to designate this attribute in Notification Service notifications. This field must be set to 0 for attributes that are not involved in Notification Service notifications.
- *attrDefaultValue*: contains a value that will automatically be assigned by the IMM Service to this attribute if no value is specified when an object containing this attribute is created. Must be set to NULL if there is no default value for this attribute.

4.2.8 SaImmAttrValuesT

SaImmAttrValuesT is used to specify the values of one attribute of an object.

```
typedef struct {
        SaImmAttrNameT  attrName;
        SaUint32T attrValuesNumber;
        SaImmAttrValueT *attrValues;
} SaImmAttrValuesT;
```

The *attrName* field indicates the attribute name and the *attrValuesNumber* field indicates the number of attribute values contained in the array of value descriptors pointed to by the *attrValues* field.

4.2.9 SaImmAttrModificationTypeT

SaImmAttrModificationTypeT specifies the type of modification to apply on the values of an attribute.

```
typedef enum {
        SA_IMM_ATTR_VALUES_ADD = 1,
        SA_IMM_ATTR_VALUES_DELETE = 2,
        SA_IMM_ATTR_VALUES_REPLACE = 3
} SaImmAttrModificationTypeT;
```

- SA_IMM_ATTR_VALUES_ADD is used to add one or several values to an attribute in an object. If the attribute did not already have a value, the attribute is added.
- SA_IMM_ATTR_DELETE is used to remove one or several specified values from an attribute of an object. If all values of the attribute are removed, the attribute is also removed from the object. If the intent is to remove an attribute without specifying all its values, the SA_IMM_ATTR_REPLACE enum can be used (see next enum value).

- SA_IMM_ATTR_REPLACE is used to replace all current values of an attribute with a new set of values. If the new set of values is empty, the attribute is removed. If one or several values are specified and the attribute does not exist in the object, the attribute is added to the object with the new set of values.

SaImmAttrModificationT is used to specify the modification to apply on an object attribute.

4.2.10 SaImmAttrModificationT

```
typedef struct {
    SaImmAttrModificationTypeTmodType;
    SaImmAttrValuesT modAttr;
} SaImmAttrModificationT;
```

The *modType* field indicates the type of modification to perform. The *modAttr* field specifies the attribute name and the values to be added to the attribute, removed from the attribute, or that will replace the existing values. An empty set of values can be specified by setting *attrValuesNumber* to 0 and *attrValues* to NULL in the *modAttr* field. It is an error to use such an empty set of values with the SA_IMM_ATTR_VALUES_ADD or SA_IMM_ATTR_VALUES_DELETE modification types.

4.2.11 SaImmScopeT

SaImmScopeT is used to specify the scope of some IMM Service operations.

```
typedef enum {
    SA_IMM_ONE = 1,
    SA_IMM_SUBLEVEL = 2,
    SA_IMM_SUBTREE = 3
} SaImmScopeT;
```

- SA_IMM_ONE indicates that the scope of the operation is targeted to a single object.
- SA_IMM_SUBLEVEL indicates that the scope of the operation is targeted to one object and its direct children.
- SA_IMM_SUBTREE indicates that the scope of the operation is targeted to one object and the entire subtree rooted at that object.

4.2.12 SaImmSearchOptionsT

SaImmSearchOptionsT is used to specify various options when performing searches among IMM Service objects.

typedef SaUint64T SaImmSearchOptionsT;

Two kinds of options can be specified by *SaImmSearchOptionsT:*

- Options related to the search criteria. Currently, only one such option is supported by the IMM Service. It must be specified for all search operations:

  #define SA_IMM_SEARCH_ONE_ATTR 0x0001

  SA_IMM_SEARCH_ONE_ATTR enables the retrieval of objects containing an attribute of a particular name and assigned to a particular value.

- Options used to specify which attributes of the objects matching the search criteria must be returned to the process performing the search. One and only one of these three options must be specified for each search operation:

  #define SA_IMM_SEARCH_GET_ALL_ATTR 0x0100

  #define SA_IMM_SEARCH_GET_NO_ATTR 0x0200

  #define SA_IMM_SEARCH_GET_SOME_ATTR 0x0400

  SA_IMM_SEARCH_GET_ALL_ATTR indicates that for each object matching the search criteria, all its attributes along with their values must be returned to the process performing the search.

  SA_IMM_SEARCH_GET_NO_ATTR indicates that no attributes of the objects matching the search criteria must be returned to the process performing the search. In this case, only the names of the objects matching the search criteria are returned.

  SA_IMM_SEARCH_GET_SOME_ATTR indicates that for each object matching the search criteria, only a subset of its attributes along with their values must be returned to the process performing the search. The list of attribute names to be returned is specified by another parameter of the search operation.

4.2.13 SaImmSearchParametersT

SaImmSearchParametersT is used to provide the criteria parameters used for search operations.

```
typedef struct {
    SaImmAttrNameT *attrName;
    SaImmValueTypeT attrValueType;
    SaImmAttrValueT attrValue;
} SaImmSearchOneAttrT;
```

The *SaImmSearchOneAttrT* type contains the attribute description for SA_IMM_SEARCH_ONE_ATTR search operations. The fields *attrName* and *attrValue* point to the attribute name and value being searched for. The *attrValueType* field indicates the type of value, which is assigned to the attribute.

If *attrValue* is not set to NULL, an object matches the search criteria if one of its attributes has a name identical to the name pointed to by *attrName*, the values for this attribute are of type *attrValueType,* and the value of the attribute (or one of its values for multi-valued attributes) is identical to the value pointed to by *attrValue*.

If *attrValue* is set to NULL, only the attribute name is used as a search criteria, and all objects having an attribute with such a name will be retrieved by the search operation, regardless of their values.

If *attrName* is set to NULL, *attrValue* must also be set to NULL. Such an empty criteria will match all IMM Service objects. This can be used to browse through all IMM Service objects.

```
typedef union {
    SaImmSearchOneAttrT searchOneAttr;
} SaImmSearchParametersT;
```

Note: Searching for a particular value of a non-cached runtime attribute should be used with care, as it forces the IMM Service to fetch all values from the Object Implementers, which creates extra load on the system.

4.2.14 SaImmCcbFlagsT

SaImmCcbFlagsT is used to specify the various characteristics of a CCB. Currently, only one value is provided.

```
#define SA_IMM_CCB_REGISTERED_OI        0x00000001

typedef SaUint64T SaImmCcbFlagsT;
```

SA_IMM_CCB_REGISTERED_OI: If this flag is specified, the CCB can only hold changes for objects that have a registered Object Implementer. This flag must be set by applications, which expect Object Implementers to validate the changes made through the CCB. If this flag is not set, the IMM Service accepts changes on objects with no registered implementer.

4.2.15 SaImmAdminOperationIdT

SaImmAdminOperationIdT is used to hold an identifier designating a particular administrative operation to perform on an object. The identifiers for all administrative operations of a given object class must have different integer values. However, the same values can be used for administrative operations of different object classes. In other words, the scope of an operation identifier is the object class.

typedef SaUint64T SaImmAdminOperationIdT;

4.2.16 SaImmAdminOperationParamsT

SaImmAdminOperationParamsT is used to specify the parameters of an administrative operation performed on an object.

typedef struct {

SaStringT paramName;

SaImmValueTypeT paramType;

*void *paramBuffer;*

SaUint32T paramSize;

} SaImmAdminOperationParamsT;

The *paramName* field indicates the name of the parameter. The *paramType* field indicates the type of the parameter. The *paramBuffer* field points to a buffer containing the parameter value. The *paramSize* field indicates the size in bytes of the parameter value.

4.2.17 SaImmCallbacksT

The *SaImmCallbacksT* structure defines the set of callbacks a process can provide to the IMM Service at initialization time.

```
typedef struct {
        SaImmOmAdminOperationInvokeCallbackT
                saImmOmAdminOperationInvokeCallback;
} SaImmCallbacksT;
```

4.2.18 IMM Service Object Attributes

#define SA_IMM_ATTR_CLASS_NAME "SaImmAttrClassName"

The IMM Service adds an attribute to each object holding the name of the class of the object. The name of this attribute is specified by the constant SA_IMM_ATTR_CLASS_NAME.

#define SA_IMM_ATTR_ADMIN_OWNER_NAME "SaImmAttrAdminOwnerName"

When an object has been assigned an administrative owner, the IMM Service stores the name of the object administrative owner in one attribute of the object. The name of this attribute is specified by the constant SA_IMM_ATTR_ADMIN_OWNER_NAME. This attribute does not exist in objects having no administrative owners.

#define SA_IMM_ATTR_IMPLEMENTER_NAME "SaImmAttrImplementerName"

When an object has an implementer, the IMM Service stores the name of the Object Implementer in one attribute of the object. The name of this attribute is specified by the constant SA_IMM_ATTR_IMPLEMENTER_NAME. This attribute does not exist in objects having no implementers.

The above attributes are single-value attributes and their value is of type SA_IMM_ATTR_SASTRINGT. For configuration objects, these attributes are configuration attributes and for runtime objects, these attributes are runtime attributes. If the runtime object is persistent, these attributes are also persistent.

4.3 Library Life Cycle

4.3.1 saImmOmInitialize()

Prototype

SaAisErrorT saImmOmInitialize(

*SaImmHandleT *immHandle,*

*const SaImmCallbacksT *immCallbacks,*

*SaVersionT *version*

);

Parameters

immHandle - [out] A pointer to the handle designating this particular initialization of the IMM Service that is to be returned by the IMM Service. This handle provides access to the Object Management APIs of the IMM Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

immCallbacks - [in] If *immCallbacks* is set to NULL, no callback is registered; otherwise, it is a pointer to an *SaImmCallbacksT* structure, containing the callback functions of the process that the IMM Service may invoke. Only non-NULL callback functions in this structure will be registered. For the *SaImmCallbacksT* type definition, see Section 4.2.17 on page 25.

version - [in/out] As an input parameter, *version* is a pointer to the required IMM Service version (see the SA Forum Overview document). In this case, *minorVersion* is ignored and should be set to 0x00.
As an output parameter, the version actually supported by the IMM Service is delivered. For the *SaVersionT* type definition, see the SA Forum Overview document.

Description

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

If the invoking process exits after successfully returning from the *saImmOmInitialize()* function and before invoking *saImmOmFinalize()* to finalize the handle *immHandle* (see Section 4.3.4 on page 32), the IMM Service automatically finalizes this handle

and any other handles, which have been acquired via the handle *immHandle* when the death of the process is detected.

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

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

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

if (implementation supports the required *releaseCode*)

releaseCode = required *releaseCode*

else *{*

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

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

else

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

}

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

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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_VERSION - The *version* parameter is not compatible with the version of the Information Model Management Service implementation.

See Also

saImmOmSelectionObjectGet(), saImmOmDispatch(), saImmOmFinalize()

4.3.2 saImmOmSelectionObjectGet()

Prototype

```
SaAisErrorT saImmOmSelectionObjectGet(
        SaImmHandleT immHandle,
        SaSelectionObjectT *selectionObject
);
```

Parameters

immHandle - [in] The handle, obtained through the *saImmOmInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

selectionObject - [out] A pointer to the operating system handle that the invoking process can use to detect pending callbacks. For the *SaSelectionObjectT* type definition, see the SA Forum Overview document.

Description

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

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

The *selectionObject* returned by *saImmOmSelectionObjectGet()* is valid until *saImmOmFinalize()* is invoked on the same handle *immHandle*.

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 *immHandle* 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 Information Model Management 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).

See Also

saImmOmInitialize(), saImmOmDispatch(), saImmOmFinalize()

4.3.3 saImmOmDispatch()

Prototype

SaAisErrorT saImmOmDispatch(

SaImmHandleT immHandle,

SaDispatchFlagsT dispatchFlags

);

Parameters

immHandle - [in] The handle, obtained through the *saImmOmInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

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

Description

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

See Also

saImmOmInitialize(), saImmOmSelectionObjectGet(), saImmOmFinalize()

4.3.4 saImmOmFinalize()

Prototype

```
SaAisErrorT saImmOmFinalize(
        SaImmHandleT immHandle
);
```

Parameters

immHandle - [in] The handle, obtained through the *saImmOmInitialize()* function, designating this particular initialization of the IMM Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

Description

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

If the *saImmOmFinalize()* function returns successfully, the *saImmOmFinalize()* function releases all resources acquired when *saImmOmInitialize()* was called. Moreover, it implicitly invokes:

- *saImmOmSearchFinalize()* on all search handles initialized with *immHandle* and not yet finalized.
- *saImmOmAccessorFinalize()* on all accessor handles initialized with *immHandle* and not yet finalized.
- *saImmOmAdminOwnerFinalize()* on all administrative owner handles initialized with *immHandle* and not yet finalized.

Furthermore, *saImmOmFinalize()* cancels all pending callbacks related to asynchronous operations performed with *immHandle*. Note that because the callback invocation is asynchronous, it is still possible that some callback calls are processed after this call returns successfully.

After *saImmOmFinalize()* returns successfully, the selection object is no longer valid.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

See Also

saImmOmInitialize()

4.4 Object Class Management

The following APIs are used to create and delete object classes. A caller can also use them to query the definition of an existing object class.

4.4.1 saImmOmClassCreate()

Prototype

```
SaAisErrorT saImmOmClassCreate(
        SaImmHandleT immHandle,
        const SaImmClassNameT className,
        SaImmClassCategoryT classCategory,
        const SaImmAttrDefinitionT **attrDefinitions
);
```

Parameters

immHandle - [in] The handle, obtained through the *saImmOmInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

className - [in] The name of the object class to create. The *SaImmClassNameT* type is defined in Section 4.2.2 on page 17.

classCategory - [in] Category of the object class. The *SaImmClassCategoryT* type is defined in Section 4.2.4 on page 18.

attrDefinitions - [in] NULL terminated array of pointers to definitions of the class attributes. The *SaImmAttrDefinitionT* type is defined in Section 4.2.7 on page 20.

Description

This function creates a new object class of name *className*. The new object class can be a configuration or runtime object class, depending on the *classCategory* parameter setting.

Object class definitions are stored in a persistent manner by the IMM Service.

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, the *attrDefinitions* parameter contains a NULL or zero length attribute name, an invalid value type, an invalid default attribute value, or a set of attribute flags, which are inconsistent with the class category specified by the *classCategory* parameter.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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_EXIST - An object class with a name identical to *className* already exists.

See Also

saImmOmInitialize()

4.4.2 saImmOmClassDescriptionGet()

Prototype

SaAisErrorT saImmOmClassDescriptionGet(
SaImmHandleT immHandle,
const SaImmClassNameT className,
*SaImmClassCategoryT *classCategory,*
*SaImmAttrDefinitionT ***attrDefinitions*
);

Parameters

immHandle - [in] The handle, obtained through the *saImmOmInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

className - [in] The name of the object class for which a description is requested. The *SaImmClassNameT* type is defined in Section 4.2.2 on page 17.

classCategory - [out] Pointer to an *SaImmClassCategoryT* structure to contain the category of the object class. The *SaImmClassCategoryT* type is defined in Section 4.2.4 on page 18.

attrDefinitions - [out] Pointer to a NULL terminated array of pointers to definitions of the class attributes. The *SaImmAttrDefinitionT* type is defined in Section 4.2.7 on page 20.

Description

This function returns a description of the object class of name *className*.

The Information Model Management Service library allocates the memory to return the attribute definitions. When the calling process no longer needs to access the attribute definitions, the memory must be freed by calling the *saImmOmClassDescriptionMemoryFree()* function.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

SA_AIS_ERR_BAD_HANDLE - The handle *immHandle* 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 Information Model Management Service library or the provider of the service is out of memory and cannot provide the service.

SA_AIS_ERR_NOT_EXIST - There is no object class with a name identical to *className*.

See Also

saImmOmInitialize(), saImmOmClassCreate(), saImmOmClassDescriptionMemoryFree()

4.4.3 saImmOmClassDescriptionMemoryFree()

Prototype

```
SaAisErrorT saImmOmClassDescriptionMemoryFree(
        SaImmHandleT immHandle,
        SaImmAttrDefinitionT **attrDefinitions
);
```

Parameters

immHandle - [in] The handle, obtained through the *saImmOmInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

attrDefinitions - [in] NULL terminated array of pointers to attribute definitions to be freed. The *SaImmAttrDefinitionT* type is defined in Section 4.2.7 on page 20.

Description

This function deallocates the memory pointed to by *attrDefinitions*, and which was allocated by a previous call to the *saImmOmClassDescriptionGet()* function.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

See Also

saImmOmInitialize(), saImmOmClassCreate(), saImmOmClassDescriptionGet()

4.4.4 saImmOmClassDelete()

Prototype

SaAisErrorT saImmOmClassDelete(

SaImmHandleT immHandle,

const SaImmClassNameT className

);

Parameters

immHandle - [in] The handle, obtained through the *saImmOmInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

className - [in] Name of the object class to be deleted. The *SaImmClassNameT* type is defined in Section 4.2.2 on page 17.

Description

This function deletes the object class designated by *className* provided there are no existing objects of this class.

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 *immHandle* 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 Information Model Management 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 - There is no object class with a name identical to *className*.

SA_AIS_ERR_BUSY - The object class cannot be deleted as objects of this class still exist, or a request to create an object of this class has been added to a CCB.

See Also

saImmOmInitialize(), saImmOmClassCreate()

4.5 Object Search

This set of APIs is used to search for particular objects in the IMM Service object tree and also to obtain the values of some of their attributes.

In order to facilitate the management of the memory allocated by the IMM Service library to return the results of the search, the search is performed through a search iterator.

The search criteria is specified when the search iterator is initialized. At initialization time, the attributes to be retrieved are also specified for each object that matches the search criteria. Then, each invocation of the iterator returns the object name and the specified attributes of the next object satisfying the search criteria.

The iteration is terminated through the finalize API.

Every object created before the invocation of the *saImmOmSearchInitialize()* function, which matches the search criteria, and has not been modified or deleted before the invocation of *saImmOmSearchFinalize()*, will be returned exactly once by the *saImmOmSearchNext()* search iterator. No other guarantees are made: Objects that

are created after the iteration is initialized, modified, or deleted before the iteration is finalized, may or may not be returned by the search iterator.

4.5.1 saImmOmSearchInitialize()

Prototype

```
SaAisErrorT saImmOmSearchInitialize(
    SaImmHandleT immHandle,
    const SaNameT *rootName,
    SaImmScopeT scope,
    SaImmSearchOptionsT searchOptions,
    const SaImmSearchParametersT *searchParam,
    const SaImmAttrNameT *attributeNames,
    SaImmSearchHandleT *searchHandle
);
```

Parameters

immHandle - [in] The handle, obtained through the *saImmOmInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

rootName - [in] Pointer to the name of the root object for the search. If set to NULL, the search starts at the root of the IMM Service tree. For the *SaNameT* type definition, see the SA Forum Overview document.

scope - [in] Scope of the search. The *SaImmScopeT* type is defined in Section 4.2.11 on page 22.

searchOptions - [in] Specifies the type of criteria being used as well as which attribute values must be returned for each object matching the search criteria. The *SaImmSearchOptionsT* type is defined in Section 4.2.12 on page 23.

searchParam - [in] A pointer to the search parameters according to the search criteria specified in *searchOption*. The *SaImmSearchParametersT* type is defined in Section 4.2.13 on page 23.

attributeNames - [in] NULL terminated array of attribute names for which values must be returned while iterating through all objects matching the search criteria. Only used if the SA_IMM_SEARCH_GET_SOME_ATTR option has been set in the *searchOptions* parameter. It must be set to NULL otherwise. The *SaImmAttrNameT* type is defined in Section 4.2.2 on page 17.

searchHandle - [out] Search handle used later to iterate through all objects that match the search criteria. The *SaImmSearchHandleT* type is defined in Section 4.2.1 on page 17.

Description

This function initializes a search operation limited to a set of targeted objects designated by the *scope* and *rootName* parameters.

The targeted set of objects is determined as follows:

- If *scope* is SA_IMM_SUBLEVEL, the scope of the operation is the object designated by *rootName* and its direct children.
- If *scope* is SA_IMM_SUBTREE, the scope of the operation is the object designated by *rootName* and the entire subtree rooted at that object
- SA_IMM_ONE is not a valid value for the *scope* parameter.

The SA_IMM_SEARCH_ONE_ATTR option must be set in the *searchOptions* parameter.

One and only one of the following three options must be set in the *searchOptions* parameter:

- SA_IMM_SEARCH_GET_ALL_ATTR,
- SA_IMM_SEARCH_GET_NO_ATTR, or
- SA_IMM_SEARCH_GET_SOME_ATTR.

This parameter specifies which attributes must be returned for each object matching the search criteria. If SA_IMM_SEARCH_GET_SOME_ATTR is set, the *attributeNames* parameter specifies the names of the attributes to be returned. If SA_IMM_SEARCH_GET_SOME_ATTR is not set, the *attributeNames* parameter must be set to NULL.

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 *immHandle* 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 Information Model Management 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 - *rootName* is not the name of an existing object.

See Also

saImmOmInitialize()

4.5.2 saImmOmSearchNext()

Prototype

```
SaAisErrorT saImmOmSearchNext(
    SaImmSearchHandleT searchHandle,
    SaNameT *objectName,
    SaImmAttrValuesT ***attributes
);
```

Parameters

searchHandle - [in] Handle returned by *saImmOmSearchInitialize()*. For the *SaImmSearchHandleT* type definition, see Section 4.2.1 on page 17.

objectName - [out] Pointer to the name of the next object matching the search criteria. For the *SaNameT* type definition, see the SA Forum Overview document.

attributes - [out] Pointer to a NULL terminated array of pointers to data structures holding the names and values of the attributes of this object, which where selected when the search was initialized. The *SaImmAttrValuesT* type is defined in Section 4.2.8 on page 21.

Description

This function is used to obtain the next object matching the search criteria.

Attribute names and values will only be populated in the memory area pointed to by *attributes* if the handle *searchHandle* was obtained by specifying

SA_IMM_SEARCH_GET_ALL_ATTR or SA_IMM_SEARCH_GET_SOME_ATTR in the *searchOptions* parameter of the corresponding *saImmOmSearchInitialize()* call.

If one of the attributes requested by the search has no value or is a non-persistent runtime attribute, and there is no registered Object Implementer for the object, only the attribute name is returned (*attrValuesNumber* is set to 0 and *attrValues* is set to NULL in the *SaImmAttrValuesT* data structure specified by the *attributes* parameter).

The memory used to return the selected object attribute names and values is allocated by the library and will be deallocated at the next invocation of *saImmOmSearchNext()* or *saImmOmSearchFinalize()* for the same search handle.

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 *searchHandle* 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 Information Model Management 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 - All objects matching the search criteria have already been returned to the calling process. The caller can now invoke the *saImmOmSearchFinalize()* function. Note that if no object matches the search criteria, this value is returned at the first invocation of *saImmOmSearchNext()*.

See Also

saImmOmInitialize(), saImmOmSearchInitialize(), saImmOmSearchFinalize()

4.5.3 saImmOmSearchFinalize()

Prototype

SaAisErrorT saImmOmSearchFinalize(

SaImmSearchHandleT searchHandle

);

Parameters

searchHandle - [in] Handle returned by *saImmOmSearchInitialize()*. For the *SaImmSearchHandleT* type definition, see Section 4.2.1 on page 17.

Description

This function finalizes the search initialized by a previous call to *saImmOmSearchInitialize()*. It frees all memory previously allocated by that search, in particular, memory used to return attribute names and values in the previous *saImmOmSearchNext()* invocation.

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

See Also

saImmOmInitialize(), saImmOmSearchInitialize(), saImmOmSearchNext()

4.6 Object Access

This set of functions is used to access the values of some attributes of an object already known by its name. Once an application has discovered the object hierarchy, it can use this interface to fetch some particular attribute values.

The object accessor is a way to facilitate the management of the memory allocated by the IMM Service library to return attribute names and values.

4.6.1 saImmOmAccessorInitialize()

Prototype

SaAisErrorT saImmOmAccessorInitialize(

SaImmHandleT immHandle,

*SaImmAccessorHandleT *accessorHandle*

);

Parameters

immHandle - [in] The handle, obtained through the *saImmOmInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

accessorHandle - [out] Pointer to the object accessor handle. For the *SaImmAccessorHandleT* type definition, see Section 4.2.1 on page 17.

Description

This function initializes an object accessor.

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 *immHandle* 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 Information Model Management 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).

See Also

saImmOmInitialize()

4.6.2 saImmOmAccessorGet()

Prototype

```
SaAisErrorT saImmOmAccessorGet(
    SaImmAccessorHandleT accessorHandle,
    const SaNameT *objectName,
    const SaImmAttrNameT *attributeNames,
    SaImmAttrValuesT ***attributes
);
```

Parameters

accessorHandle - [in] Object accessor handle. For the *SaImmAccessorHandleT* type definition, see Section 4.2.1 on page 17.

objectName - [in] Pointer to the name of the object being accessed. For the *SaNameT* type definition, see the SA Forum Overview document.

attributeNames - [in] NULL terminated array of attribute names for which values must be returned. The *SaImmAttrNameT* type is defined in Section 4.2.2 on page 17.

attributes - [out] Pointer to a NULL terminated array of pointers to data structures containing the name and values of the attributes being accessed. The *SaImmAttrValuesT* type is defined in Section 4.2.8 on page 21.

Description

This function uses an object accessor to obtain the values assigned to some attributes of an object. If *attributeNames* is set to NULL, the values of all attributes of the object are returned.

If one of the requested attributes has no value or is a non-persistent runtime attribute, and there is no registered Object Implementer for the object, only the attribute name is returned (*attrValuesNumber* is set to 0 and *attrValues* is set to NULL in the *SaImmAttrValuesT* data structure specified by the *attributes* parameter).

The memory used to return the object attribute names and values is allocated by the library and will be deallocated at the next invocation of *saImmOmAccessorGet()* or *saImmOmAccessorFinalize()*.

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 *accessorHandle* 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 Information Model Management 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 - *objectName* is not the name of an existing object, or one of the names specified by *attributeNames* does not exist for the object designated by *objectName*.

See Also

saImmOmAccessorInitialize()

4.6.3 saImmOmAccessorFinalize()

Prototype

SaAisErrorT saImmOmAccessorFinalize(

SaImmAccessorHandleT accessorHandle

);

Parameters

accessorHandle - [in] Object accessor handle. For the *SaImmAccessorHandleT* type definition, see Section 4.2.1 on page 17.

Description

This function finalizes the object accessor and deallocates all memory previously allocated for this object accessor. In particular, the memory used to return the object attribute names and values during the previous invocation of *saImmOmAccessorGet()* is freed.

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

See Also

saImmOmAccessorInitialize()

4.7 Object Administration Ownership

Each object of the IMM Service may have at any time one and only one administrative owner, which has the ability to modify the object or invoke administrative operations on the object. The administrative owner is usually distinct from the Object

Implementer. Establishing the administrative ownership of an object, or a set of objects, guarantees that a process unrelated with this administrative owner will not modify the objects concurrently.

As management operations may be performed by a set of cooperating processes, an administrative owner is identified by its name and several processes may perform sequentially or concurrently administrative operations under the same administrative owner name (by initializing several administrative owner handles with the same name).

A process acting under that administrative owner name will typically release the administrative ownership on the objects. Note that this process need not necessarily be any of the one or more processes that set the administrative owner name of the objects. For recovery purposes, a process with appropriate privileges can also release the administrative ownership of a set of objects without acting under the name of their current administrative owner by invoking the *saImmOmAdminOwnerClear()* function.

Management applications are responsible for releasing the administrative ownership on objects when their management activities are completed.

4.7.1 saImmOmAdminOwnerInitialize()

Prototype

```
SaAisErrorT saImmOmAdminOwnerInitialize(
    SaImmHandleT immHandle,
    const SaImmAdminOwnerNameT adminOwnerName,
    SaBoolT releaseOwnershipOnFinalize,
    SaImmAdminOwnerHandleT *ownerHandle
);
```

Parameters

immHandle - [in] The handle, obtained through the *saImmOmInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

adminOwnerName - [in] Name of the administrative owner. The *SaImmAdminOwnerNameT* type is defined in Section 4.2.2 on page 17.

releaseOwnershipOnFinalize - [in] This parameter specifies how to release administrative ownerships that were acquired with the newly initialized handle *ownerHandle*

when this handle is finalized. For the *SaBoolT* type definition, see the SA Forum Overview document.

ownerHandle - [out] Pointer to the handle for the administrative owner. For the *SaImmAdminOwnerHandleT* type definition, see Section 4.2.1 on page 17.

Description

This function initializes a handle for an administrative owner whose name is pointed to by *adminOwnerName*. All objects owned by a particular administrative owner have the attribute whose name is defined by the constant SA_IMM_ATTR_ADMIN_OWNER_NAME set to the administrative owner name. For objects without an administrative owner, that attribute does not exist.

If *releaseOwnershipOnFinalize* is set to SA_TRUE, the IMM Service automatically releases all administrative ownerships that were acquired with the newly initialized handle *ownerHandle* when this handle is finalized.
If *releaseOwnershipOnFinalize* is set to SA_FALSE, the IMM Service does not automatically release the ownership when the handle is finalized. In this case, if a management application fails while holding the administrative ownership on some objects, it is the responsibility of the recovery procedure of the failed application to release it.

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 *immHandle* 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 Information Model Management 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).

See Also

saImmOmInitialize(), saImmOmAdminOwnerSet(), saImmOmAdminOwnerFinalize()

4.7.2 saImmOmAdminOwnerSet()

Prototype

```
SaAisErrorT saImmOmAdminOwnerSet(
    SaImmAdminOwnerHandleT ownerHandle,
    const SaNameT **objectNames,
    SaImmScopeT scope
);
```

Parameters

ownerHandle - [in] Administrative owner handle. For the *SaImmAdminOwnerHandleT* type definition, see Section 4.2.1 on page 17.

objectNames - [in] NULL terminated array of pointers to object names. For the *SaNameT* type definition, see the SA Forum Overview document.

scope - [in] Scope of the operation. The *SaImmScopeT* type is defined in Section 4.2.11 on page 22.

Description

This function sets the administrative owner, designated by *ownerHandle,* as the owner of the set of objects designated by the *scope* and *objectNames* parameters. This function can be used to acquire the administrative ownership of either configuration or runtime objects.

The targeted set of objects is determined as follows:

- If *scope* is SA_IMM_ONE, the scope of the operation are the objects designated by *objectNames*.
- If *scope* is SA_IMM_SUBLEVEL, the scope of the operation are the objects designated by *objectNames* and their direct children.
- If *scope* is SA_IMM_SUBTREE, the scope of the operation are the objects designated by *objectNames* and the entire subtrees rooted at these objects.

The operation fails if one of the targeted objects has already an administrative owner whose name is different from the name used to initialize *ownerHandle*. If the operation fails, the administrative owner of the targeted objects is not changed.

If the operation succeeds, the SA_IMM_ATTR_ADMIN_OWNER_NAME attribute of all targeted objects is set to the administrative owner name, specified when *ownerHandle* was initialized.

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 *ownerHandle* 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 Information Model Management 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 - At least one of the objects designated by *objectNames* is not the name of an existing object.

SA_AIS_ERR_EXIST - At least one of the objects targeted by this operation already has an administrative owner having a name different from the name used to initialize *ownerHandle*.

See Also

saImmOmAdminOwnerInitialize(), saImmOmAdminOwnerRelease(), saImmOmAdminOwnerClear()

4.7.3 saImmOmAdminOwnerRelease()

Prototype

```
SaAisErrorT saImmOmAdminOwnerRelease(
        SaImmAdminOwnerHandleT ownerHandle,
        const SaNameT **objectNames,
        SaImmScopeT scope
);
```

Parameters

ownerHandle - [in] Administrative owner handle. For the *SaImmAdminOwnerHandleT* type definition, see Section 4.2.1 on page 17.

objectNames - [in] NULL terminated array of pointers to object names. For the *SaNameT* type definition, see the SA Forum Overview document.

scope - [in] Scope of the operation. The *SaImmScopeT* type is defined in Section 4.2.11 on page 22.

Description

This function releases the administrative owner of the set of objects designated by the *scope* and *objectNames* parameters.

The targeted set of objects is determined as follows:

- If *scope* is SA_IMM_ONE, the scope of the operation are the objects designated by *objectNames*.
- If *scope* is SA_IMM_SUBLEVEL, the scope of the operation are the objects designated by *objectNames* and their direct children.
- If *scope* is SA_IMM_SUBTREE, the scope of the operation are the objects designated by *objectNames* and the entire subtrees rooted at these objects.

If the operation succeeds, the SA_IMM_ATTR_ADMIN_OWNER_NAME attribute of all targeted objects is removed from the objects.

The operation fails if one of the targeted objects is not already owned by the administrative owner with the name used to initialize *ownerHandle*. The operation also fails if an administrative operation is currently in progress on one of the targeted objects, or if a change request for one of the targeted objects is included in a CCB that has not been applied or finalized.
If the operation fails, the administrative owner of the targeted objects is not changed.

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 *ownerHandle* 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 Information Model Management 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 - At least one of the objects designated by *objectNames* is not the name of an existing object, or at least one of the objects targeted by this operation is not owned by the administrative owner whose name was used to initialize *ownerHandle*.

SA_AIS_ERR_BUSY - An administrative operation is currently in progress on one of the targeted objects, or a change request for one of the targeted objects is included in a CCB that has not been applied or finalized.

See Also

saImmOmAdminOwnerInitialize(), saImmOmAdminOwnerSet()

4.7.4 saImmOmAdminOwnerFinalize()

Prototype

SaAisErrorT saImmOmAdminOwnerFinalize(

SaImmAdminOwnerHandleT ownerHandle

);

Parameters

ownerHandle - [in] Administrative owner handle. For the *SaImmAdminOwnerHandleT* type definition, see Section 4.2.1 on page 17.

Description

This function releases *ownerHandle*. If *ownerHandle* has been initialized with the *releaseOwnershipOnFinalize* option set to SA_FALSE, this operation does not release the administrative ownership that has been set on objects by using this handle.

If *ownerHandle* has been initialized with the *releaseOwnershipOnFinalize* option set to SA_TRUE, this operation also releases the administrative ownership that has been set on objects by using this handle.

This function implicitly invokes *saImmOmCcbFinalize()* on all CCB handles initialized with *ownerHandle* and not yet finalized.

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

See Also

saImmOmAdminOwnerInitialize(), saImmOmCcbInitialize()

4.7.5 saImmOmAdminOwnerClear()

Prototype

```
SaAisErrorT saImmOmAdminOwnerClear(
        SaImmHandleT immHandle,
        const SaNameT **objectNames,
        SaImmScopeT scope
);
```

Parameters

immHandle - [in] The handle, obtained through the *saImmOmInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmHandleT* type definition, see Section 4.2.1 on page 17.

objectNames - [in] NULL terminated array of pointers to object names. For the *SaNameT* type definition, see the SA Forum Overview document.

scope - [in] Scope of the operation. The *SaImmScopeT* type is defined in Section 4.2.11 on page 22.

Description

This function clears the administrative owner of the set of objects designated by the *scope* and *objectNames* parameters.
The targeted set of objects is determined as follows:

- If *scope* is SA_IMM_ONE, the scope of the operation are the objects designated by *objectNames*.
- If *scope* is SA_IMM_SUBLEVEL, the scope of the operation are the objects designated by *objectNames* and their direct children.
- If *scope* is SA_IMM_SUBTREE, the scope of the operation are the objects designated by *objectNames* and the entire subtrees rooted at these objects.

The operation succeeds even if some targeted objects do not have an administrative owner, or if the set of targeted objects have different administrative owners.

If the operation succeeds, the SA_IMM_ATTR_ADMIN_OWNER_NAME attribute of all targeted objects is removed from the objects.

The operation fails if an administrative operation is currently in progress on one of the targeted objects, or if a change request for one of the targeted objects is included in a CCB that has not been applied or finalized.
If the operation fails, the administrative owner of the targeted objects is not changed.

This function is intended to be used only when recovering from situations where some management applications took ownership of some objects and did not release them.

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NOT_EXIST - At least one of the objects designated by *objectNames* is not the name of an existing object.

SA_AIS_ERR_BUSY - An administrative operation is currently in progress on one of the targeted objects, or a change request for one of the targeted objects is included in a CCB that has not been applied or finalized.

See Also

saImmOmAdminOwnerInitialize(), saImmOmAdminOwnerSet()

4.8 Configuration Changes

All changes of IMM Service configuration objects are performed in the context of configuration change bundles (CCB). Once a CCB has been initialized, change requests can be added to a CCB. A change request can be a creation, deletion, or modification. Later on, when the CCB is applied, all pending change requests included in the CCB are applied with all-or-nothing semantics (either all change requests are applied or none are applied). The change requests are applied in the order they have been added to the CCB.

A CCB is associated with a single administrative owner, and all objects modified through change requests included in one CCB must have the same administrative owner as the CCB.

The IMM Service does not prevent applications from reading (through *saImmOmSearchNext()* or *saImmOmAccessorGet()*) the attribute values of the objects modified by a CCB while a CCB is being applied. This means, for example, that a search operation may return for some matching objects the values that their attributes had before the CCB was applied and for other objects the values that their attributes had after the CCB was applied. However, the IMM Service must guarantee that all CCB changes are applied atomically for each particular object. The attribute values returned by *saImmOmSearchNext()* or *saImmOmAccessorGet()* for a particular object must all be the values before the CCB was applied or all be the values after the CCB was applied (in other words, mixing old and new values is not allowed).

The IMM Service enforces the following limitation regarding concurrent management tasks for a particular object: At a given time, an object can be the target of either a single CCB or a single administrative operation.

4.8.1 saImmOmCcbInitialize()

Prototype

```
SaAisErrorT saImmOmCcbInitialize(
        SaImmAdminOwnerHandleT ownerHandle,
        SaImmCcbFlagsT ccbFlags,
        SaImmCcbHandleT *ccbHandle
);
```

Parameters

ownerHandle - [in] Administrative owner handle. For the *SaImmAdminOwnerHandleT* type definition, see Section 4.2.1 on page 17.

ccbFlags - [in] CCB flags. For the *SaImmCcbFlagsT* type definition, see Section 4.2.14 on page 24.

ccbHandle - [out] Pointer to the CCB handle. For the *SaImmCcbHandleT* type definition, see Section 4.2.1 on page 17.

Description

This function initializes a new CCB and returns a handle for it. The CCB is initialized as empty (it contains no change requests).

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 *ownerHandle* 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 Information Model Management 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).

See Also

saImmOmAdminOwnerInitialize()

4.8.2 saImmOmCcbObjectCreate()

Prototype

```
SaAisErrorT saImmOmCcbObjectCreate(
    SaImmCcbHandleT ccbHandle,
    const SaImmClassNameT className,
    const SaNameT *parentName,
    const SaImmAttrValuesT **attrValues
);
```

Parameters

ccbHandle - [in] CCB handle. For the *SaImmCcbHandleT* type definition, see Section 4.2.1 on page 17.

className - [in] Object name class. The *SaImmClassNameT* type is defined in Section 4.2.2 on page 17.

parentName - [in] Pointer to the name of the new object's parent. For the *SaNameT* type definition, see the SA Forum Overview document.

attrValues - [in] NULL terminated array of pointers to attribute descriptors. The *SaImmAttrValuesT* type is defined in Section 4.2.8 on page 21.

Description

This function adds to the CCB, identified by its handle *ccbHandle,* a request to create a new IMM Service object. Once this new object is successfully created, it will be automatically owned by the administrative owner of the CCB. The new object is created as a child of the object designated by the *parentName* DN. If *parentName* is set to NULL, the new object is created as a top level object.

This function can only be used to create configuration objects. The attributes specified by the *attrValues* array must match the object class definition. Only configuration attributes can be specified by the *attrValues* array.

Attributes named SA_IMM_ATTR_CLASS_NAME, SA_IMM_ATTR_ADMIN_OWNER_NAME, and SA_IMM_ATTR_IMPLEMENTER_NAME cannot be specified by the *attrValues* descriptors as these attributes are set automatically by the IMM Service.

The creation will only be performed when the CCB is applied. However, the IMM Service invokes any existing Object Implementer synchronously to validate the creation request and may return an error if this creation is not a valid operation.

The IMM Service adds an attribute of name SA_IMM_ATTR_CLASS_NAME to the new object; the value of this attribute contains the name of the object class as specified by the *objectClass* parameter.

If the parent object is not administratively owned by the administrative owner of the CCB, this function fails and returns SA_AIS_ERR_BAD_OPERATION.

If this function returns an error, the creation request has not been added to the CCB.

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular:

- the *className* parameter specifies a runtime object class,
- there is no valid RDN attribute specified for the new object,
- all of the configuration attributes required at object creation are not provided by the caller.

- the *attrValues* parameter includes:

  - runtime attributes,
  - attributes not defined for the specified class,
  - attributes with values that do not match the defined value type for the attribute,
  - multiple values for a single-valued attribute.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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_BAD_OPERATION - The parent object is not administratively owned by the administrative owner of the CCB or the creation of the object has been rejected by its Object Implementer.

SA_AIS_ERR_NOT_EXIST - This value is returned due to one or more of the following reasons:

- The *parentName* parameter is not the name of an existing object.
- The *className* parameter is not the name of an existing object class.
- One or more of the attributes specified by *attrValues* are not valid attribute names for *className*.
- There is no registered Object Implementer for the object to be created, and the CCB has been initialized with the SA_IMM_CCB_REGISTERED_OI flag set.

SA_AIS_ERR_EXIST - An object with the same name already exists.

SA_AIS_FAILED_OPERATION - The operation failed because the CCB has been aborted due to the registration of an Object Implementer or a problem with one of the registered Object Implementers. The CCB is now empty.

See Also

saImmOmCcbInitialize(), saImmOmCcbApply()

4.8.3 saImmOmCcbObjectDelete()

Prototype

SaAisErrorT saImmOmCcbObjectDelete(

SaImmCcbHandleT ccbHandle,

*const SaNameT *objectName*

);

Parameters

ccbHandle - [in] CCB handle. For the *SaImmCcbHandleT* type definition, see Section 4.2.1 on page 17.

objectName - [in] Pointer to the object name. For the *SaNameT* type definition, see the SA Forum Overview document.

Description

This function adds to the CCB, identified by its handle *ccbHandle,* a request to delete the configuration object identified by the *objectName* parameter and the entire sub-tree of configuration objects rooted at that object.

This operation fails if one of the targeted objects is not a configuration object administratively owned by the administrative owner of the CCB.

The deletion will only be performed when the CCB is applied. However, the IMM Service invokes any existing Object Implementer synchronously to validate the deletion request and may return an error if the deletion is not a valid operation.

If this function returns an error, the deletion request has not been added to the CCB.

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 *ccbHandle* 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 Information Model Management 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_BAD_OPERATION - At least one of the targeted objects is not a configuration object owned by the administrative owner of the CCB, or its Object Implementer has rejected its deletion.

SA_AIS_ERR_NOT_EXIST - This value is returned due to one or both of the following reasons:

- The *objectName* parameter is not the name of an existing object.
- There is no registered Object Implementer for at least one of the objects targeted by this operation, and the CCB has been initialized with the SA_IMM_CCB_REGISTERED_OI flag set.

SA_AIS_ERR_EXIST - At least one of the targeted objects has one child in the IMM Service hierarchy that is not targeted by this operation.

SA_AIS_ERR_BUSY - At least one of the targeted objects is already the target of an administrative operation or of a change request in another CCB.

SA_AIS_FAILED_OPERATION - The operation failed because the CCB has been aborted due to the registration of an Object Implementer or a problem with one of the registered Object Implementers. The CCB is now empty.

See Also

saImmOmCcbInitialize(), saImmOmCcbApply()

4.8.4 saImmOmCcbObjectModify()

Prototype

SaAisErrorT saImmOmCcbObjectModify(

SaImmCcbHandleT ccbHandle,

*const SaNameT *objectName,*

*const SaImmAttrModificationT **attrMods*

);

Parameters

ccbHandle - [in] CCB handle. For the *SaImmCcbHandleT* type definition, see Section 4.2.1 on page 17.

objectName - [in] Pointer to the name of the object to be modified. For the *SaNameT* type definition, see the SA Forum Overview document.

attrMods - [in] NULL terminated array of pointers to descriptors of the modifications to perform. The *SaImmAttrModificationT* type is defined in Section 4.2.10 on page 22.

Description

This function adds to the CCB, identified by its handle *ccbHandle,* a request to modify configuration attributes of an IMM Service object. Only writable configuration attributes can be modified (SA_IMM_ATTR_WRITABLE).

This operation fails if the targeted object is not administratively owned by the administrative owner of the CCB.

The modify request will only be performed when the CCB is applied. However, the IMM Service invokes any existing Object Implementer synchronously to validate the modify request and may return an error if the requested modifications are not allowed.

Attributes named SA_IMM_ATTR_CLASS_NAME, SA_IMM_ATTR_ADMIN_OWNER_NAME, and SA_IMM_ATTR_IMPLEMENTER_NAME cannot be modified.

If this function returns an error, the modify request has not been added to the CCB.

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, the *attrMods* parameter includes:

- runtime attributes,
- attributes not defined for the specified class,
- attributes with values that do not match the defined value type for the attribute,
- a new value for the RDN attribute,
- attributes that cannot be modified,
- multiple values or additional values for a single-valued attribute.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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_BAD_OPERATION - The modified object is not a configuration object owned by the administrative owner of the CCB, or its Object Implementer has rejected the modification.

SA_AIS_ERR_NOT_EXIST - This value is returned due to one or more of the following reasons:

- The *objectName* parameter is not the name of an existing object.
- One or more attribute names specified in the *attrMods* parameter are not valid for the object class.
- There is no registered Object Implementer for the object specified by the *objectName* parameter, and the CCB has been initialized with the SA_IMM_CCB_REGISTERED_OI flag set.

SA_AIS_ERR_NOT_EXIST - The *objectName* parameter is not the name of an existing object, or one or more attribute names specified in the *attrMods* parameter are not valid for the object class.

SA_AIS_ERR_BUSY - The object designated by *objectName* is already the target of an administrative operation or of a change request in another CCB.

SA_AIS_FAILED_OPERATION - The operation failed because the CCB has been aborted due to the registration of an Object Implementer or a problem with one of the registered Object Implementers. The CCB is now empty.

See Also

saImmOmCcbInitialize(), saImmOmCcbApply()

4.8.5 saImmOmCcbApply()

Prototype

```
SaAisErrorT saImmOmCcbApply(
        SaImmCcbHandleT ccbHandle
);
```

Parameters

ccbHandle - [in] CCB handle. For the *SaImmCcbHandleT* type definition, see Section 4.2.1 on page 17.

Description

This function applies all requests included in the configuration change bundle identified by its handle *ccbHandle*. The requests are applied with all-or-nothing semantics, that is, either all requests are applied or none are applied. All requests are applied in the order in which they have been added to the CCB.

Any existing Object Implementer involved by the change requests contained in the CCB is invoked to apply the changes. The Object Implementers are responsible for checking that the set of requested changes is valid.

This operation fails if the administrative ownership of one of the objects targeted by this CCB has changed since the change was added to the CCB, and the new administrative owner of the object is not anymore the administrative owner of the CCB.

When this call returns with success or failure, all requests included in the CCB when the call was issued have been removed. The CCB is empty and can be populated again with change requests belonging to the same administrative owner.

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 *ccbHandle* 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 Information Model Management 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_BAD_OPERATION - The changes requested do not constitute a valid set of changes.

SA_AIS_FAILED_OPERATION - The operation failed because the CCB has been aborted due to the registration of an Object Implementer or a problem with one of the registered Object Implementers. The CCB is now empty.

See Also

saImmOmCcbInitialize(), saImmOmCcbObjectCreate(), saImmOmCcbObjectDelete(), saImmOmCcbObjectModify()

4.8.6 saImmOmCcbFinalize()

Prototype

```
SaAisErrorT saImmOmCcbFinalize(
        SaImmCcbHandleT ccbHandle
);
```

Parameters

ccbHandle - [in] CCB handle. For the *SaImmCcbHandleT* type definition, see Section 4.2.1 on page 17.

Description

This function finalizes the CCB identified by *ccbHandle*.

All change requests contained in the CCB are removed without being applied.

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

See Also

saImmOmCcbInitialize()

4.9 Administrative Operations Invocation

4.9.1 saImmOmAdminOperationInvoke(), saImmOmAdminOperationInvokeAsync()

Prototype

```
SaAisErrorT saImmOmAdminOperationInvoke(
        SaImmAdminOwnerHandleT ownerHandle,
        const SaNameT *objectName,
        SaImmAdminOperationIdT operationId,
        const SaImmAdminOperationParamsT **params,
        SaAisErrorT *operationReturnValue,
        SaTimeT timeout
);
```

Parameters

ownerHandle - [in] Administrative owner handle. For the *SaImmAdminOwnerHandleT* type definition, see Section 4.2.1 on page 17.

objectName - [in] Pointer to the object name. For the *SaNameT* type definition, see the SA Forum Overview document.

operationId - [in] Identifier of the administrative operation. The *SaImmAdminOperationIdT* type is defined in Section 4.2.15 on page 25.

params - [in] NULL terminated array of pointers to parameter descriptors. The *SaImmAdminOperationParamsT* type is defined in Section 4.2.16 on page 25.

operationReturnValue - [out] Value returned by the Object Implementer for the invoked operation. This value is specific to the administrative operation being performed. For more details about this value, refer to the Object Implementer administrative operation description.

timeout - [in] The *saImmOmAdminOperationInvoke()* invocation is considered to have failed if it does not complete by the time specified. For the *SaTimeT* type definition, see the SA Forum Overview document.

```
SaAisErrorT saImmOmAdminOperationInvokeAsync(
    SaImmAdminOwnerHandleT ownerHandle,
    SaInvocationT invocation,
    const SaNameT *objectName,
    SaImmAdminOperationIdT operationId,
    const SaImmAdminOperationParamsT **params
);
```

Parameters

ownerHandle - [in] Administrative owner handle. For the *SaImmAdminOwnerHandleT* type definition, see Section 4.2.1 on page 17.

invocation - [in] Used to match this invocation of *saImmOmAdminOperationInvokeAsync()* with the corresponding invocation of *SaImmOmAdminOperationInvokeCallbackT* callback. For the *SaInvocationT* type definition, see the SA Forum Overview document.

objectName - [in] Pointer to the object name. For the *SaNameT* type definition, see the SA Forum Overview document.

operationId - [in] Identifier of the administrative operation. The *SaImmAdminOperationIdT* type is defined in Section 4.2.15 on page 25.

params - [in] NULL terminated array of pointers to parameter descriptors. The *SaImmAdminOperationParamsT* type is defined in Section 4.2.16 on page 25.

Description

Using the IMM Service as an intermediary, these two functions request the implementer of the object, designated by its name *objectName,* to perform an administrative operation characterized by *operationId* on that object. Administrative operations can be performed on configuration or runtime objects.

Each descriptor of the params array represents an input parameter of the administrative operation to execute.

The function *saImmOmAdminOperationInvoke()* is the synchronous variant and returns only when the Object Implementer has successfully completed the execution of the administrative operation, or when an error has been detected by the IMM Service or the Object Implementer.

The function *saImmOmAdminOperationInvokeAsync()* is the asynchronous variant and returns as soon as the IMM Service has registered the request to be transmitted to the Object Implementer. If the IMM Service detects an error while registering the

request, an error is returned immediately, and no further invocation of the *SaImmOmAdminOperationInvokeCallbackT* callback must be expected for this invocation of *saImmOmAdminOperationInvokeAsync()*. If no error is detected by the IMM Service while registering the request, the invocation of *saImmOmAdminOperationInvokeAsync()* completes successfully, and a later invocation of the *SaImmOmAdminOperationInvokeCallbackT* callback will occur to indicate the success or failure of the administrative operation on the target object.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

SA_AIS_ERR_INIT - The initialization with *saImmOmInitialize()* of the IMM Service handle used itself to initialize *ownerHandle* was incomplete, since the *SaImmOmAdminOperationInvokeCallbackT* callback function was missing. This return value only applies to the *saImmOmAdminOperationInvokeAsync()* function.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management Service library or its library 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_BAD_OPERATION - The object designated by *objectName* is not owned by the administrative owner associated with *ownerHandle*.

SA_AIS_ERR_NOT_EXIST - The *objectName* parameter is not the name of an existing object, or there is no registered Object Implementer for this object.

SA_AIS_ERR_BUSY - The object designated by *objectName* is already the target of an administrative operation or of a change request in a CCB.

SA_AIS_FAILED_OPERATION - The operation failed due to a problem with the Object Implementer.

See Also

saImmOmAdminOwnerInitialize()

4.9.2 SaImmOmAdminOperationInvokeCallbackT

Prototype

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

Parameters

invocation - [in] Used to match this callback invocation to the corresponding invocation of *saImmOmAdminOperationInvokeAsync()*. For the *SaInvocationT* type definition, see the SA Forum Overview document.

operationReturnValue - [in] Value returned by the Object Implementer for the operation invoked through the corresponding *saImmOmAdminOperationInvokeAsync()* function. This value is specific to the administrative operation being performed, and it is valid only if the *error* parameter is set to SA_AIS_OK. For more details about this value, refer to the Object Implementer administrative operation description. For the *SaTimeT* type definition, see the SA Forum Overview document.

error - [in] Indicates whether the IMM Service succeeded or not to invoke the Object Implementer. For the *SaAisErrorT* type definition, see the SA Forum Overview document.
The returned values are:

- SA_AIS_OK - The function completed successfully.
- SA_AIS_ERR_LIBRARY - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.
- SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout occurred before the call could complete. It is unspecified whether the call succeeded or whether it did not.
- SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The process may retry later.
- SA_AIS_ERR_BAD_HANDLE - The handle *ownerHandle* in the corresponding invocation of the *saImmOmAdminOperationInvokeAsync()* function 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 IMM 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 - The *objectName* parameter is not the name of an existing object, or there is no registered Object Implementer for this object.
- SA_AIS_ERR_BUSY - The object designated by *objectName* in the corresponding invocation of the *saImmOmAdminOperationInvokeAsync()* function was already the target of an administrative operation or of a change request in a CCB.
- SA_AIS_FAILED_OPERATION - The operation failed due to a problem with the Object Implementer.

Object Implementers may extend the preceding list of return values with return values specific to the administrative operation being performed. Refer to the Object Implementer administrative operation description for more details.

Description

The IMM Service invokes this callback function when the operation requested by the invocation of *saImmOmAdminOperationInvokeAsync()* completes successfully, or an error is detected. This callback is invoked in the context of a thread issuing an *saImmOmDispatch()* call on the handle *immHandle,* which was used to initialize the *ownerHandle* specified in the *saImmOmAdminOperationInvokeAsync()* call.

Return Values

None.

See Also

saImmOmAdminOperationInvokeAsync(), saImmOmDispatch()

5.0 IMM Service - Object Implementer API Specification

5.1 Include File and Library Name

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

#include <saImmOi.h>

To use the IMM Service Object Implementer API, an application must be bound with the following library:

libSaImmOi.so

5.2 Type Definitions

5.2.1 IMM Service Handle

The following handle is used by IMM Service Object Implementer API functions:

typedef SaUint64T SaImmOiHandleT;

5.2.2 SaImmOiImplementerNameT

SaImmOiImplementerNameT represents an Object Implementer name; it points to an UTF-8 encoded character string, terminated by the NULL character.

typedef SaStringT SaImmOiImplementerNameT;

5.2.3 SaImmOiCcbIdT

typedef SaUint64T SaImmOiCcbIdT;

This type is used through in the IMM Service Object Implementer APIs to identify a particular configuration change bundle (CCB).

5.2.4 SaImmOiCallbacksT

The *SaImmOiCallbacksT* structure defines the set of callbacks a process implementing IMM Service objects can provide to the IMM Service at initialization time.

```
typedef struct {
    SaImmOiRtAttrUpdateCallbackT saImmOiRtAttrUpdateCallback;
    SaImmOiCcbObjectCreateCallbackT saImmOiCcbObjectCreateCallback;
    SaImmOiCcbObjectDeleteCallbackT saImmOiCcbObjectDeleteCallback;
    SaImmOiCcbObjectModifyCallbackT saImmOiCcbObjectModifyCallback;
    SaImmOiCcbCompletedCallbackT saImmOiCcbCompletedCallback;
    SaImmOiCcbApplyCallbackT saImmOiCcbApplyCallback;
    SaImmOiCcbAbortCallbackT saImmOiCcbAbortCallback;
    SaImmOiAdminOperationCallbackT saImmOiAdminOperationCallback;
} SaImmOiCallbacksT;
```

5.3 Library Life Cycle

5.3.1 saImmOiInitialize()

Prototype

```
SaAisErrorT saImmOiInitialize(
    SaImmOiHandleT *immOiHandle,
    const SaImmOiCallbacksT *immOiCallbacks,
    SaVersionT *version
);
```

Parameters

immOiHandle - [out] A pointer to the handle designating this particular initialization of the Information Model Management Service that is to be returned by the Information Model Management Service. This handle provides access to the Object Implementer APIs of the IMM Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

immOiCallbacks - [in] A pointer to an *SaImmOiCallbacksT* structure, containing the callback functions of the process that the IMM Service may invoke. Only non-NULL callback functions in this structure will be registered. The *SaImmOiCallbacksT* type is defined in Section 5.2.4 on page 73.

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

As an output parameter, the version actually supported by the Information Model Management Service is delivered. For the *SaVersionT* type definition, see the SA Forum Overview document.

Description

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

The returned handle *immOiHandle* is not associated with any implementer name. The association of the handle with an implementer name is performed by the invocation of the *saImmOiImplementerSet()* function.

If the invoking process exits after successfully returning from the *saImmOiInitialize()* function and before invoking *saImmOiFinalize()* to finalize the handle *immOiHandle* (see Section 5.3.4 on page 79), the IMM Service automatically finalizes this handle when the death of the process is detected.

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

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

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

if (implementation supports the required *releaseCode*)

releaseCode = required *releaseCode*

else *{*

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

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

else

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

}

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

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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_VERSION - The *version* parameter is not compatible with the version of the Information Model Management Service implementation.

See Also

saImmOiSelectionObjectGet(), saImmOiDispatch(), saImmOiFinalize(), saImmOiImplementerSet()

5.3.2 saImmOiSelectionObjectGet()

Prototype

```
SaAisErrorT saImmOiSelectionObjectGet(
        SaImmOiHandleT immOiHandle,
        SaSelectionObjectT *selectionObject
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

selectionObject - [out] A pointer to the operating system handle that the invoking process can use to detect pending callbacks. For the *SaSelectionObjectT* type definition, see the SA Forum Overview document.

Description

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

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

The *selectionObject* returned by *saImmOiSelectionObjectGet()* is valid until *saImmOiFinalize()* is invoked on the same handle *immOiHandle*.

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 *immOiHandle* 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 Information Model Management 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).

See Also

saImmOiInitialize(), saImmOiDispatch(), saImmOiFinalize()

5.3.3 saImmOiDispatch()

Prototype

```
SaAisErrorT saImmOiDispatch(
        SaImmOiHandleT immOiHandle,
        SaDispatchFlagsT dispatchFlags
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

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

Description

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

See Also

saImmOiInitialize(), saImmOiSelectionObjectGet(), saImmOiFinalize()

5.3.4 saImmOiFinalize()

Prototype

SaAisErrorT saImmOiFinalize(

SaImmOiHandleT immOiHandle

);

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

Description

The *saImmOiFinalize()* function closes the association, represented by the *immOiHandle* parameter, between the invoking process and the Information Model Management Service. The process must have invoked *saImmOiInitialize()* before it invokes this function. A process must invoke this function once for each handle it acquired by invoking *saImmOiInitialize()*.

If the *saImmOiFinalize()* function returns successfully, the *saImmOiFinalize()* function releases all resources acquired when *saImmOiInitialize()* was called. Furthermore, *saImmOiFinalize()* cancels all pending callbacks related to asynchronous operations performed with the handle *immOiHandle*. Note that because the callback invocation is asynchronous, it is still possible that some callback calls are

processed after this call returns successfully. After *saImmOiFinalize()* is invoked, the selection object is no longer valid.

This function does not release the associations established between object classes or objects and the implementer name, which may still be associated with the handle *immOiHandle*.
The next time a process associates the same implementer name with an Object Implementer handle, that process automatically becomes the implementer of all objects having the same implementer name.

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

See Also

saImmOiInitialize()

5.4 Object Implementer

As a runtime object is created by its Object Implementer, the IMM Service can automatically set the name of the implementer of a runtime object when the object is created.

On the other hand, configuration objects are typically created by management applications, which are not the Object Implementers. Configuration Object Implementers must explicitly indicate to the IMM Service which configuration objects they implement. This can be done for all objects of a given class or by targeting a particular set of objects.

The implementer of an object is identified by an implementer name. Once set, the implementer name remains associated with the object until explicitly released. This applies even if the process, which was registered as the Object Implementer, clears the implementer name associated with its Object Implementer handle. This enables

faster recovery of Object Implementers failures as the new Object Implementer does not have to explicitly re-register all objects it implements. Simply registering itself with the same implementer name allows the IMM Service to associate all objects with the same implementer name with that process.

5.4.1 saImmOiImplementerSet()

Prototype

SaAisErrorT saImmOiImplementerSet(

SaImmOiHandleT immOiHandle,

const SaImmOiImplementerNameT implementerName

);

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

ImplementerName - [in] Name of the Object Implementer. The *SaImmOiImplementerNameT* type is defined in Section 5.2.2 on page 73.

Description

This function sets the implementer name specified in the *implementerName* parameter for the handle *immOiHandle*. In order to be a valid parameter to all Object Implementer APIs except for *saImmOiSelectionObjectGet(), saImmOiDispatch(), saImmOiImplementerSet()* and *saImmOiFinalize(),* an Object Implementer handle must be successfully associated with an implementer name.

This function also registers the invoking process as an Object Implementer whose name is specified in the *ImplementerName* parameter. At any given time, only a single process in the entire cluster can be registered under a particular Object Implementer name.

The invoking process becomes the implementer of all existing IMM Service objects having an implementer name identical to *ImplementerName*.

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

SA_AIS_ERR_EXIST - An Object Implementer with the same name is already registered with the IMM Service.

See Also

saImmOiInitialize(), saImmOiImplementerClear()

5.4.2 saImmOiImplementerClear()

Prototype

```
SaAisErrorT saImmOiImplementerClear(
        SaImmOiHandleT immOiHandle
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

Description

This function clears the implementer name associated with the *immOiHandle* handle and unregisters the invoking process as an Object Implementer for the name previously associated with *immOiHandle*.

With no associated implementer name, *immOiHandle* is only a valid parameter for the following APIs: *saImmOiSelectionObjectGet(), saImmOiDispatch(), saImmOiImplementerSet()* and *saImmOiFinalize()*.

IMM object classes and objects, which have an implementer name equal to the name previously associated with *immOiHandle* keep the same implementer name but stay without any registered Object Implementer until a process invokes *saImmOiImplementerSet()* again with the same implementer name.

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

See Also

saImmOiInitialize(), saImmOiImplementerSet()

5.4.3 saImmOiClassImplementerSet()

Prototype

```
SaAisErrorT saImmOiClassImplementerSet(
        SaImmOiHandleT immOiHandle,
        const SaImmClassNameT className
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

className - [in] Object class name. The *SaImmClassNameT* type is defined in Section 4.2.2 on page 17.

Description

This function informs the IMM Service that all objects, which are instances of the class designated by its name, *className,* are implemented by the Object Implementer whose name has been associated with the handle *immOiHandle*.

The current process becomes the current implementer of all objects of the class designated by its name, *className,* unless the object class has already an Object Imple-

menter whose name is different from the implementer name provided when *immOiHandle* was initialized.

The IMM Service adds an attribute of name SA_IMM_ATTR_IMPLEMENTER_NAME to all objects of that class (existing objects as well as objects created in the future) with a value equal to the implementer name associated with the handle *immOiHandle*. This is performed for each existing instance of *className* and also for new instances of *className* when created in the future.

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 *immOiHandle* is invalid, since it is corrupted, uninitialized, has already been finalized, or it is not associated with an implementer name.

SA_AIS_ERR_BAD_OPERATION -The *className* parameter specifies a runtime object class.

SA_AIS_ERR_NOT_EXIST - The *className* parameter is not the name of an existing class.

SA_AIS_ERR_EXIST - The *className* parameter has already an Object Implementer whose name is different from the implementer name associated with the handle *immOiHandle*.

See Also

saImmOiInitialize()

5.4.4 saImmOiClassImplementerRelease()

Prototype

SaAisErrorT saImmOiClassImplementerRelease(

SaImmOiHandleT immOiHandle,

const SaImmClassNameT className

);

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

className - [in] Object class name. The *SaImmClassNameT* type is defined in Section 4.2.2 on page 17.

Description

This function informs the IMM Service that the implementer, whose name is associated with the handle *immOiHandle,* must not be considered anymore as the implementer of the objects, which are instances of the class designated by its name, *className*.

If the operation succeeds, the IMM Service removes the attribute of name SA_IMM_ATTR_IMPLEMENTER_NAME as well as all runtime cached attributes from all objects of that class.

This operation fails if the invoking process is not the current implementer of the class designated by its name, *className*, or if one or more objects affected by the operation are currently taking part in an in-progress CCB and/or administrative operations.

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 *immOiHandle* is invalid, since it is corrupted, uninitialized, has already been finalized, or it is not associated with an implementer name.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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_BAD_OPERATION -The *className* parameter specifies a runtime object class.

SA_AIS_ERR_NOT_EXIST - The *className* parameter is not the name of an existing class, or the implementer of object instances from *className* is different from the implementer name associated with the handle *immOiHandle*.

SA_AIS_ERR_BUSY - One or more objects affected by this operation are taking part in an in-progress CCB and/or an administrative operation.

See Also

saImmOiInitialize(), saImmOiClassImplementerSet()

5.4.5 saImmOiObjectImplementerSet()

Prototype

```
SaAisErrorT saImmOiObjectImplementerSet(
        SaImmOiHandleT immOiHandle,
        const SaNameT *objectName,
        SaImmScopeT scope
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

objectName - [in] Pointer to the object name. For the *SaNameT* type definition, see the SA Forum Overview document.

scope - [in] Scope of the operation. The *SaImmScopeT* type is defined in Section 4.2.11 on page 22.

Description

This function informs the IMM Service that all objects, which are designated by the *scope* and *objectName* parameters, are implemented by the Object Implementer whose name has been associated with the handle *immOiHandle*.

The current process becomes the current implementer of all targeted objects.

The targeted set of objects is determined as follows:

- If *scope* is SA_IMM_ONE, the scope of the operation is the object designated by *objectName*.
- If *scope* is SA_IMM_SUBLEVEL, the scope of the operation is the object designated by *objectName* and its direct children.
- If *scope* is SA_IMM_SUBTREE, the scope of the operation is the object designated by *objectName* and the entire subtree rooted at that object.

The operation fails if one of the targeted objects has already an implementer whose name is different from the name associated with the handle *immOiHandle*. If the operation fails, the implementer of the targeted objects is not changed.

If the operation succeeds, the SA_IMM_ATTR_IMPLEMENTER_NAME attribute of all targeted objects is set to the implementer name associated with the handle *immOiHandle*.

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 *immOiHandle* is invalid, since it is corrupted, uninitialized, has already been finalized, or it is not associated with an implementer name.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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_BAD_OPERATION - One or more targeted objects are runtime objects.

SA_AIS_ERR_NOT_EXIST - The *objectName* parameter is not the name of an existing object.

SA_AIS_ERR_EXIST - At least one of the objects targeted by this operation already has an implementer having a name different from the name associated with the handle *immOiHandle*.

See Also

saImmOiInitialize(), saImmOiObjectImplementerRelease()

5.4.6 saImmOiObjectImplementerRelease()

Prototype

```
SaAisErrorT saImmOiObjectImplementerRelease(
        SaImmOiHandleT immOiHandle,
        const SaNameT *objectName,
        SaImmScopeT scope
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

objectName - [in] Pointer to the object name. For the *SaNameT* type definition, see the SA Forum Overview document.

scope - [in] Scope of the operation. The *SaImmScopeT* type is defined in Section 4.2.11 on page 22.

Description

This function informs the IMM Service that the implementer whose name is associated with the handle *immOiHandle* must not be considered anymore as the implementer of the set of objects designated by the *scope* and *objectName* parameters.

The targeted set of objects is determined as follows:

- If *scope* is SA_IMM_ONE, the scope of the operation is the object designated by *objectName*.
- If *scope* is SA_IMM_SUBLEVEL, the scope of the operation is the object designated by *objectName* and its direct children.
- If *scope* is SA_IMM_SUBTREE, the scope of the operation is the object designated by *objectName* and the entire subtree rooted at that object.

The operation fails if one of the targeted objects is not implemented by the current process, or if one or more objects affected by the operation are taking part in an in-progress CCB and/or an administrative operation. If the operation fails, the implementer of the targeted objects is not changed.

If the operation succeeds, the SA_IMM_ATTR_IMPLEMENTER_NAME attribute and all cached runtime attributes of all targeted objects is removed from the objects.

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 *immOiHandle* is invalid, since it is corrupted, uninitialized, has already been finalized, or it is not associated with an implementer name.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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_BAD_OPERATION - One or more targeted objects are runtime objects.

SA_AIS_ERR_NOT_EXIST - The *objectName* parameter is not the name of an existing object or at least one of the objects targeted by this operation is not owned by the administrative owner whose name was used to initialize *ownerHandle*.

SA_AIS_ERR_BUSY - One or more objects affected by this operation are taking part in an in-progress CCB and/or an administrative operation.

See Also

saImmOiInitialize(), saImmOiClassImplementerSet()

5.5 Runtime Objects Management

The set of functions contained in this section are used by an Object Implementer to create or delete runtime objects and update the runtime attributes of either configuration or runtime objects. They are similar to the functions provided in the IMM Service Object Management interface, the difference being that they are not part of a configuration change bundle (CCB).

The values of non-persistent attributes are not accessible when there is no registered implementer for the objects they belong to.

Runtime attributes whose values are cached by the IMM Service must be updated by its Object Implementer whenever their value changes. The value of non-cached attributes must only be updated by the Object Implementer on request by the IMM Service through the invocation of the *SaImmOiRtAttrUpdateCallbackT* callback function.

Updating cached runtime attribute values in the IMM Service generates some load on the system each time the values change. Attributes whose values change frequently but are rarely read through the Object Management API should typically not be cached.

5.5.1 saImmOiRtObjectCreate()

Prototype

```
SaAisErrorT saImmOiRtObjectCreate(
        SaImmOiHandleT immOiHandle,
        const SaImmClassNameT className,
        const SaNameT *parentName,
        const SaImmAttrValuesT **attrValues
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

className - [in] Object class name. The *SaImmClassNameT* type is defined in Section 4.2.2 on page 17.

parentName - [in] Pointer to the name of the new object's parent. For the *SaNameT* type definition, see the SA Forum Overview document.

attrValues- [in] NULL terminated array of pointers to attribute descriptors. The *SaImmAttrValuesT* type is defined in Section 4.2.8 on page 21.

Description

This function creates a new IMM Service runtime object.

The new object is created as a child of the object designated by the *parentName* DN. If *parentName* is set to NULL, the new object is created as a top level object.

The attributes specified by the *attrValues* array must match the object class definition. Only runtime attributes can be specified by the *attrValues* array.

The *attrValues* array must contain one and only one attribute with the SA_IMM_ATTR_RDN flag set; this attribute is used as the Relative Distinguished Name of the new object.

Attributes named SA_IMM_ATTR_CLASS_NAME, SA_IMM_ATTR_ADMIN_OWNER_NAME and SA_IMM_ATTR_IMPLEMENTER_NAME cannot be specified by the *attrValues* descriptors as these attributes are set automatically by the IMM Service.

The IMM Service adds an attribute of name SA_IMM_ATTR_CLASS_NAME to the new object, and the value of this attribute contains the name of the object class as specified by the *className* parameter.

The invoking process becomes the implementer of the new object, and the IMM Service adds an attribute of name SA_IMM_ATTR_IMPLEMENTER_NAME to the new object with a value equal to the implementer name associated with the handle *immOiHandle*.

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 *immOiHandle* is invalid, since it is corrupted, uninitialized, has already been finalized, or it is not associated with an implementer name.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular:

- the *className* parameter specifies a configuration object class,
- there is no valid RDN attribute specified for the new object,
- some cached attributes do not have values,
- the *attrValues* parameter includes:
  - attributes not defined for the specified class,
  - attributes with values that do not match the defined value type for the attribute,
  - multiple values for a single-valued attribute.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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 due to one or more of the following reasons:

- The *parentName* parameter is not the name of an existing object.

- The *className* parameter is not the name of an existing object class.
- One or more of the attributes specified by *attrValues* are not valid attribute names for *className*.

SA_AIS_ERR_EXIST - An object with the same name already exists.

See Also

saImmOiInitialize()

5.5.2 saImmOiRtObjectDelete()

Prototype

```
SaAisErrorT saImmOiRtObjectDelete(
        SaImmOiHandleT immOiHandle,
        const SaNameT *objectName
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

objectName - [in] Pointer to the object name. For the *SaNameT* type definition, see the SA Forum Overview document.

Description

This function deletes the object identified by the *objectName* parameter and the entire subtree of objects rooted at that object.

This operation fails if one of the targeted objects is not a runtime object implemented by the invoking process.

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 *immOiHandle* is invalid, since it is corrupted, uninitialized, has already been finalized, or it is not associated with an implementer name.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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_BAD_OPERATION - One or more of the targeted objects are configuration objects.

SA_AIS_ERR_NOT_EXIST - The *objectName* parameter is not the name of an existing object.

SA_AIS_ERR_EXIST - At least one of the targeted objects has one child in the IMM Service hierarchy that is not targeted by this operation.

See Also

saImmOiInitialize()

5.5.3 saImmOiRtObjectUpdate()

Prototype

```
SaAisErrorT saImmOiRtObjectUpdate(
        SaImmOiHandleT immOiHandle,
        const SaNameT *objectName,
        const SaImmAttrModificationT **attrMods
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

objectName - [in] Pointer to the name of the updated object. For the *SaNameT* type definition, see the SA Forum Overview document.

attrMods - [in] NULL terminated array of pointers to descriptors of the modifications to perform. The *SaImmAttrModificationT* type is defined in Section 4.2.10 on page 22.

Description

This function updates runtime attributes of a configuration or runtime object.

Attributes named SA_IMM_ATTR_CLASS_NAME, SA_IMM_ATTR_ADMIN_OWNER_NAME and SA_IMM_ATTR_IMPLEMENTER_NAME cannot be modified.

This operation fails if the targeted object is not implemented by the invoking process.

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 *immOiHandle* is invalid, since it is corrupted, uninitialized, has already been finalized, or it is not associated with an implementer name.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, the *attrMods* parameter includes:

- configuration attributes,
- a new value for the RDN attribute,
- attributes not defined for the specified class,
- attributes with values that do not match the defined value type for the attribute,
- multiple values or additional values for a single-valued attribute.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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_BAD_OPERATION - The modifications requested on the object are not valid.

SA_AIS_ERR_NOT_EXIST - The *objectName* parameter is not the name of an existing object, or one or more attribute names specified in the *attrMods* parameter are not valid for the object class.

SA_AIS_FAILED_OPERATION - The targeted object is not implemented by the invoking process.

See Also

saImmOiInitialize()

5.5.4 SaImmOiRtAttrUpdateCallbackT

Prototype

```
typedef SaAisErrorT (*SaImmOiRtAttrUpdateCallbackT)(
    SaImmOiHandleT immOiHandle,
    const SaNameT *objectName,
    const SaImmAttrNameT *attributeNames
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

objectName - [in] Pointer to the name of the object for which the update is requested. For the *SaNameT* type definition, see the SA Forum Overview document.

attributeNames - [in] NULL terminated array of attribute names for which values must be updated. The *SaImmAttrNameT* type is defined in Section 4.2.2 on page 17.

Description

The IMM Service invokes this callback function to request an Object Implementer to update the values of some attributes of a runtime object. These attributes are attributes whose values are not cached by the IMM Service. The target object is identified by its name, *objectName*. The process must use the *saImmOiRtObjectUpdate()* function to update the values of the attributes whose names are specified by the *attributeNames* parameter.

If a requested attribute has no value, the SA_IMM_ATTR_VALUES_REPLACE flag of the *SaImmAttrModificationTypeT* structure can be used in the *saImmOiRtObjectUpdate()* call to set the attribute value to the empty set.

On successful return of this callback, all requested attributes have been updated.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_NO_MEMORY - The implementer process is out of memory and cannot provide the service.

SA_AIS_ERR_NO_RESOURCES -The implementer process is out of required resources (other than memory) to provide the service.

SA_AIS_FAILED_OPERATION - The implementer process failed to update the requested attributes due to an error occurring in the *saImmOiRtObjectUpdate()* invocation.

See Also

saImmOiInitialize()

5.6 Configuration Objects Implementer

Implementers of configuration objects are invoked via callbacks by the IMM Service when requests to change the objects they implement are added to a configuration change bundle (CCB) and also when the CCB is being applied. On each callback invocation indicating the addition of a change request to a CCB, the Object Implementer is responsible for validating the change and memorizing it, so it can react appropriately when all change requests contained in the CCB are applied by invoking the *saImmOmCcbApply()* function.

If a change is added to a CCB for a particular object but its Object Implementer did not provide the appropriate callback for the change or the callbacks used by the IMM Service to eventually apply or abort the CCB, the change is rejected with an SA_AIS_ERR_FAILED_OPERATION error.

Each change request added to a CCB must be validated by the Object Implementer with the understanding that the new request will be applied after all requests already present in the CCB are applied. So the validation should not consider the current state of the IMM Information Model but the state it would have with all prior requests being applied. Before invoking the Object Implementer callbacks, the IMM Service validates that the Information Model tree hierarchy is consistent:

- It checks that a newly created object has a parent in the hierarchy,
- and it checks that an object being deleted has no child.

If changes are made on configuration objects for which there is no registered Object Implementer, the IMM Service still applies the changes when the CCB is applied without invoking any Object Implementer callbacks for these changes.

If an Object Implementer either registers or unregisters itself while some registered CCB changes are still pending for objects it implements (i.e., the IMM Service has not yet passed the step of successfully invoking all *SaImmOiCcbCompletedCallbackT* functions of registered Object Implementers for the CCB), the IMM Service aborts the CCBs which hold these changes.

When the user of the Object Management API requests the IMM Service to apply all change requests contained in a CCB, the IMM Service gives a last chance to the Object Implementers to validate that all changes will bring the set of configuration objects they implement in a consistent state. As a CCB may contain change requests for objects having different implementers, the IMM Service applies a CCB in two steps:

- In the first step, the IMM Service indicates to each Object Implementer, which has at least one object changed by the CCB requests, that the CCB is now complete and that it must validate the entire set of CCB changes. This indication is done by invoking the *SaImmOiCcbCompletedCallbackT* callback function. If one of the Object Implementers returns an error, the attempt to apply the CCB fails, and the *saImmOmCcbApply()* function returns an error.
- If all implementers agreed with the proposed changes, the IMM Service applies the changes. In a second step, the IMM Service informs the implementers that the changes have been applied by invoking the *SaImmOiCcbApplyCallbackT* callback function. If one implementer rejected the proposed changes, the IMM Service informs implementers affected by the CCB that the CCB is aborted by invoking the *SaImmOiCcbAbortCallbackT* callback function.

5.6.1 SaImmOiCcbObjectCreateCallbackT

Prototype

```
typedef SaAisErrorT (*SaImmOiCcbObjectCreateCallbackT)(
    SaImmOiHandleT immOiHandle,
    SaImmOiCcbIdT ccbId,
    const SaImmClassNameT className,
    const SaNameT *parentName,
    const SaImmAttrValuesT **attr
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

ccbId - [in] CCB identifier. The *SaImmOiCcbIdT* type is defined in Section 5.2.3 on page 73.

className - [in] Object class name. The *SaImmClassNameT* type is defined in Section 4.2.2 on page 17.

parentName - [in] Pointer to the name of the new object's parent. For the *SaNameT* type definition, see the SA Forum Overview document.

attr - [in] NULL terminated array of pointers to attribute descriptors.
The *SaImmAttrValuesT* type is defined in Section 4.2.8 on page 21.

Description

The IMM Service invokes this callback function to enable an Object Implementer to validate and register a change request being added to a CCB identified by *ccbId*. The change request is a creation request for a configuration object of a class, which is implemented by the process implementing the callback.

All parameters of the creation request are provided as parameters of the callback function to enable the implementer process to validate and memorize the creation request. Refer to the description of the *saImmOmCcbObjectCreate()* function for details on these parameters. All the parameters of the creation request may be memorized by the implementer process and associated with the *ccbId* identifier, because these parameters will not be provided later on when the CCB is finally applied.

The changes will only be applied by the IMM Service after a successful invocation of the *SaImmOiCcbCompletedCallbackT* callback.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_NO_MEMORY - The implementer process is out of memory and cannot allocate the memory required to register the request.

SA_AIS_ERR_NO_RESOURCES -The implementer process is out of required resources (other than memory) to register the request.

SA_AIS_BAD_OPERATION - The implementer process rejects the creation request.

See Also

saImmOmCcbObjectCreate(), SaImmOiCcbCompletedCallbackT

5.6.2 SaImmOiCcbObjectDeleteCallbackT

Prototype

```
typedef SaAisErrorT (*SaImmOiCcbObjectDeleteCallbackT)(
        SaImmOiHandleT immOiHandle,
        SaImmOiCcbIdT ccbId,
        const SaNameT *objectName
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

ccbId - [in] CCB identifier. The *SaImmOiCcbIdT* type is defined in Section 5.2.3 on page 73.

objectName - [in] Pointer to the object name. For the *SaNameT* type definition, see the SA Forum Overview document.

Description

The IMM Service invokes this callback function to enable an Object Implementer to validate and memorize a deletion request being added to a CCB identified by *ccbId*. The deletion request is a request to delete the object identified by the *objectName*

parameter and the entire subtree of objects rooted at that object and that are implemented by the process implementing the callback.

The *objectName* parameter may be memorized by the implementer process and associated with the *ccbId* identifier, because these parameters will not be provided later on when the CCB is finally applied.

The changes will only be applied by the IMM Service after a successful invocation of the *SaImmOiCcbCompletedCallbackT* callback.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_NO_MEMORY - The implementer process is out of memory and cannot allocate the memory required to validate and memorize the request.

SA_AIS_ERR_NO_RESOURCES -The implementer process is out of required resources (other than memory) to validate and memorize the request.

SA_AIS_BAD_OPERATION - The implementer process rejects the deletion request.

See Also

saImmOmCcbObjectDelete(), SaImmOiCcbCompletedCallbackT

5.6.3 SaImmOiCcbObjectModifyCallbackT

Prototype

```
typedef SaAisErrorT (*SaImmOiCcbObjectModifyCallbackT)(
        SaImmOiHandleT immOiHandle,
        SaImmOiCcbIdT ccbId,
        const SaNameT *objectName,
        const SaImmAttrModificationT **attrMods
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

ccbId - [in] CCB identifier. The *SaImmOiCcbIdT* type is defined in Section 5.2.3 on page 73.

objectName - [in] Pointer to the object name. For the *SaNameT* type definition, see the SA Forum Overview document.

attrMods - [in] NULL terminated array of pointers to descriptors of the modifications to perform. The *SaImmAttrModificationT* type is defined in Section 4.2.10 on page 22.

Description

The IMM Service invokes this callback function to enable an Object Implementer to validate and memorize a change request being added to a CCB identified by *ccbId*. The change request is a request to modify configuration attributes of a configuration object implemented by the process implementing the callback.

All parameters of the modification request are provided as parameters of the callback function to enable the implementer process to validate and memorize the modification request. Refer to the description of the *saImmOmCcbObjectModify()* function for details on these parameters. All the parameters of the modification request may be memorized by the implementer process and associated with the *ccbId* identifier, because these parameters will not be provided later on when the CCB is finally applied.

The changes will only be applied by the IMM Service after a successful invocation of the *SaImmOiCcbCompletedCallbackT* callback.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_NO_MEMORY - The implementer process is out of memory and cannot allocate the memory required to validate and memorize the request.

SA_AIS_ERR_NO_RESOURCES -The implementer process is out of required resources (other than memory) to validate and memorize the request.

SA_AIS_BAD_OPERATION - The implementer process rejects the modification request.

See Also

saImmOmCcbObjectModify(), SaImmOiCcbCompletedCallbackT

5.6.4 SaImmOiCcbCompletedCallbackT

Prototype

```
typedef SaAisErrorT (*SaImmOiCcbCompletedCallbackT)(
    SaImmOiHandleT immOiHandle,
    SaImmOiCcbIdT ccbId
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

ccbId - [in] CCB identifier. The *SaImmOiCcbIdT* type is defined in Section 5.2.3 on page 73.

Description

The IMM Service invokes this callback function to inform an Object Implementer that the CCB identified by *ccbId* is now complete (no additional requests will be added). The implementer process must check that the sequence of change requests contained in the CCB is valid, and that no errors will be generated when these changes are applied.

If all Object Implementers, which implement objects changed by the CCB, agree with the changes, the IMM Service will apply the changes and then invoke the *SaImmOiCcbApplyCallbackT* callback to notify all Object Implementers that the CCB has been applied.

Return Values

SA_AIS_OK - The function completed successfully.

SA_AIS_ERR_NO_MEMORY - The implementer process is out of memory and cannot allocate the memory required to later apply all requested changes.

SA_AIS_ERR_NO_RESOURCES -The implementer process is out of required resources (other than memory) to later apply all requested changes.

SA_AIS_BAD_OPERATION - The validation by the implementer process of all change requests contained in the CCB failed.

See Also

saImmOmCcbApply(), SaImmOiCcbObjectCreateCallbackT, SaImmOiCcbObjectDeleteCallbackT, SaImmOiCcbObjectModifyCallbackT

5.6.5 SaImmOiCcbApplyCallbackT

Prototype

```
typedef void (*SaImmOiCcbApplyCallbackT)(
        SaImmOiHandleT immOiHandle,
        SaImmOiCcbIdT ccbId
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

ccbId - [in] CCB identifier. The *SaImmOiCcbIdT* type is defined in Section 5.2.3 on page 73.

Description

The IMM Service invokes this callback function to inform an Object Implementer that the CCB identified by *ccbId* has been applied by the IMM Service.

All configuration changes have already been validated by the Object Implementer in a previous call to *SaImmOiCcbCompletedCallbackT*.

Each Object Implementer is responsible for determining the effect of the configuration changes.

Return Values

None

See Also

saImmOmCcbApply(), SaImmOiCcbCompletedCallbackT

5.6.6 SaImmOiCcbAbortCallbackT

Prototype

*typedef void (*SaImmOiCcbAbortCallbackT)(*
SaImmOiHandleT immOiHandle,
SaImmOiCcbIdT ccbId
);

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

ccbId - [in] CCB identifier. The *SaImmOiCcbIdT* type is defined in Section 5.2.3 on page 73.

Description

The IMM Service invokes this callback function to inform an Object Implementer that the CCB identified by *ccbId* is aborted, and the Object Implementer can remove all change requests memorized for this CCB.

Return Values

None

See Also

saImmOmCcbApply()

5.7 Administrative Operations

5.7.1 SaImmOiAdminOperationCallbackT

Prototype

*typedef void (*SaImmOiAdminOperationCallbackT) (*
SaImmOiHandleT immOiHandle,
SaInvocationT invocation,
*const SaNameT *objectName,*
SaImmAdminOperationIdT operationId,
*const SaImmAdminOperationParamsT **params*
);

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

invocation - [in] Used to match this invocation of *SaImmOiAdminOperationCallbackT* with the corresponding invocation of *saImmOiAdminOperationResult()*. For the *SaInvocationT* type definition, see the SA Forum Overview document.

objectName - [in] Pointer to the object name. For the *SaNameT* type definition, see the SA Forum Overview document.

operationId - [in] Identifier of the administrative operation. The *SaImmAdminOperationIdT* type is defined in Section 4.2.15 on page 25.

params - [in] NULL terminated array of pointers to parameter descriptors. The *SaImmAdminOperationParamsT* type is defined in Section 4.2.16 on page 25.

Description

The IMM Service invokes this callback function to request an Object Implementer to execute an administrative operation on the object designated by the its name, *objectName*. The administrative operation identified by the *operationId* parameter has been initiated by an invocation of the *saImmOmAdminOperationInvoke*() or *saImmOmAdminOperationInvokeAsync()* functions.

Each descriptor of the *params* array represents an input parameter of the administrative operation to execute.

The Object Implementer indicates the success or failure of the administrative operation by invoking the *saImmOiAdminOperationResult()* function. The invocation of *saImmOiAdminOperationResult()* can be done from the callback itself or outside of the callback by any thread of the process, which initialized the *immOiHandle*.

Return Values

None.

See Also

saImmOiInitialize(), saImmOmAdminOperationInvoke(), saImmOmAdminOperationInvokeAsync(), saImmOiAdminOperationResult()

5.7.2 saImmOiAdminOperationResult()

Prototype

```
SaAisErrorT saImmOiAdminOperationResult(
        SaImmOiHandleT immOiHandle,
        SaInvocationT invocation,
        SaAisErrorT result
);
```

Parameters

immOiHandle - [in] The handle, obtained through the *saImmOiInitialize()* function, designating this particular initialization of the Information Model Management Service. For the *SaImmOiHandleT* type definition, see Section 5.2.1 on page 73.

invocation - [in] Used to match this invocation of *saImmOiAdminOperationResult()* with the previous corresponding invocation of the *SaImmOiAdminOperationCallbackT* callback. For the *SaInvocationT* type definition, see the SA Forum Overview document.

result - [in] Result of the execution of the administrative operation. For the *SaAisErrorT* type definition, see the SA Forum Overview document.

Description

This function is used by an Object Implementer to provide to the IMM Service the result of the execution of an administrative operation requested through an invocation of the *SaImmOiAdminOperationCallbackT* callback.

This function can only be called by the process for which the *SaImmOiAdminOperationCallbackT* callback has been invoked.

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 *immOiHandle* is invalid, since it is corrupted, uninitialized, has already been finalized, or it is not associated with an implementer name.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the Information Model Management 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).

See Also

saImmOiInitialize(), SaImmOiAdminOperationCallbackT