Service Availability^{TM} Forum Application Interface Specification

Information Model Management Service SAI-AIS-IMM-A.03.01

SERVICE AVAILABILITY™ FORUM

.

Table of Contents Information Model Management Service

List of Figures

List of Tables

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

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

1.2 AIS Documents Organization

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

1.3 History

The previous releases of the IMM Service specification are:

SAI-AIS-IMM-A.01.01

SAI-AIS-IMM-A.02.01

This section presents the changes of the current release, SAI-AIS-IMM-A.03.01, with respect to the SAI-AIS-IMM-A.02.01 release. Editorial changes that do not change semantics or syntax of the described interfaces are not mentioned.

1.3.1 New Topics

⇒ To deal with configuration changes that affect objects of different object implementers, the tasks performed by an object implementer have been split in three parts, and the following roles have been introduced for each of these tasks: CCB validator, CCB applier, and runtime owner.

The CCB validator verifies the changes proposed to an object in a CCB from a particular aspect, and different aspects can be verified by different CCB validators. Therefore, an IMM object may have multiple CCB validators. For a CCB to be applied, in a first step, all CCB validators of all affected objects need to accept the proposed changes. If so, the IMM Service applies the changes to the SA Forum Information Model.

CCB appliers deploy the changes contained in the SA Forum Information Model. As changes proposed to a configuration object may affect multiple entities in the system, an object may have multiple CCB appliers.

The runtime owner is responsible for updating the state information reflected in the SA Forum Information Model and for carrying out administrative operations. Each object in the SA Forum Information Model can have only one runtime owner.

The introduction of this functional split implied a fair amount of changes in the IMM Service specification. The changes are listed next:

- The `SA_IMM_CCB_REGISTERED_OI` flag was replaced with the `SA_IMM_CCB_ALLOW_ABSENT_VALIDATORS` and `SA_IMM_CCB_ALLOW_ABSENT_APPLIERS` flags and the `SaImmCcbFlagsT` type was superseded by `SaImmCcbFlagsT_3` (see Section 4.2.14 on page 41). Note that these flags indicate that the registration of the given category of object implementers is not required for the success of the CCB.
- The `SA_IMM_ATTR_IMPLEMENTER_NAME` attribute is no longer supported. Instead, three separate attributes of an object are defined (see Section 4.2.22 on page 43), one for each object implementer role: `SA_IMM_ATTR_VALIDATOR_NAME`, `SA_IMM_ATTR_APPLIER_NAME`, and `SA_IMM_ATTR_RUNTIME_OWNER_NAME`. These attributes of a configuration object are persistent runtime attributes, and as such, they can be configured.
- The `ccbId` used in callbacks of the OI interface was made global, and its life time is now tied to the life time of the CCB handle of a created CCB. The `ccbId` is returned in the `saImmOmCcbInitialize_3()` function (see Section 4.8.1 on page 86). A nonzero valid `ccbId` can be specified in the

saImmOmSearchInitialize_3() and saImmOmAccessorInitialize_3() functions (see Section 4.5.1 on page 62 and Section 4.6.1 on page 69, respectively), so that an object implementer can perform object search and use the accessor functions using this CCB identifier.
The SaImmOiCcbIdT was replaced with the SaImmCcbIdT type (see Section 4.2.15 on page 41).

- The saImmOmCcbObjectCreate_2(), saImmOmCcbObjectDelete(), and saImmOmCcbObjectModify_2() functions (see Section 4.8.2 on page 88, Section 4.8.3 on page 91, and Section 4.8.4 on page 93, respectively) no longer trigger the invocation of the corresponding callback functions of object implementers SaImmOiCcbObjectCreateCallbackT_2, SaImmOiCcbObjectDeleteCallbackT, and SaImmOiCcbObjectModifyCallbackT_2. Instead, as explained in the next item, the validation of the CCB changes is now performed in the SaImmOiCcbValidateCallbackT function (which supersedes the SaImmOiCcbCompletedCallbackT function), and the deployment of the CCB changes is performed in the SaImmOiCcbApplyCallbackT_3 function.
- The saImmOmCcbApply_3() function (see Section 4.8.5 on page 95) triggers the invocation of the saImmOiCcbValidateCallback() function of validators (see Section 5.5.1 on page 146), which perform local and global validation (for the meaning of these terms, see Section 5.5 on page 144). If all CCB validators accept the CCB changes, the IMM Service applies these changes to the SA Forum Information Model and invokes the saImmOiCcbApplyCallback() (see Section 5.6.1 on page 151) of all CCB appliers to deploy these changes.
- The saImmOmCcbFinalize() (see Section 4.8.6 on page 98) function also invalidates the CCB identifier associated with the CCB handle to be finalized. Additionally, the description of this function was extended to state when it is allowed to invoke it.
- The object implementer roles CCB validator, CCB applier, and runtime owner are defined in the SaImmOiRoleFlagsT type in Section 5.2.3 on page 117. In several places in this document, these more specific terms are used instead of the generic term “object implementer”.
- A new role parameter has been added to the following functions to specify the role of an object implementer: saImmOiClassImplementerSet_3(), saImmOiClassImplementerRelease_3(), saImmOiObjectImplementerSet_3(), and saImmOiObjectImplementerRelease_3() (see Section 5.4.3 on page 133 up to Section 5.4.6 on page 141).

- The superseding functions `SaImmOiCcbValidateCallbackT` (see Section 5.5.1 on page 146), `SaImmOiCcbAbortCallbackT_3` (see Section 5.5.2 on page 147, and `SaImmOiCcbApplyCallbackT_3` (see Section 5.6.1 on page 151) use `SaImmCcbIdT` for the `ccbId` type and have extended semantics.
  To enable a CCB validator to dispose of the CCB identifier and of any associated state in case the CCB has been successfully validated by all CCB validators, the `SaImmOiCcbFinalizeCallbackT` function has been defined (see Section 5.5.3 on page 149).
  Due to the aforementioned changes in object implementer callback functions, the `SaImmOiCallbacksT_2` type was replaced with `SaImmOiCallbacksT_3` (see Section 5.2.7 on page 120). This modification, in turn, has led to the replacement of the `saImmOiInitialize_2()` function with the `saImmOiInitialize_3()` function (see Section 5.3.1 on page 121).
- The description of the `saImmOiAdminOperationResult()` (see Section 5.8.2.2 on page 171) has been extended to explain the IMM Service actions if the runtime owner unregisters during the execution of an administrative operation.
- A CCB iterator API has been introduced (see Section 5.7 on page 153) to allow CCB validators and appliers to iterate through configuration changes associated with a given CCB identifier. The following types have been introduced for this API:
  - `SaImmOiCcbIteratorHandleT` (see Section 5.2.1 on page 117);
  - `SaImmOiRoleFlagsT`, `SaImmOiCcbIteratorOptionT`, `SaImmOiObjectChangeT`, and `SaImmOiObjectTraverseT` (see Section 5.2.3 on page 117 up to Section 5.2.6 on page 119).
- The IMM Service UML model was extended by a new object class, `SaImmCcbApplier` (see Section 7.2 on page 177). This class contains the `saImmCcbApplierRank` configuration attribute, which represents the rank that the IMM Service uses to determine in which order it invokes the `saImmOiCcbApplyCallback()` function of CCB appliers.
- Appendix A on page 187 shows a use case for the usage of CCB validators and appliers.
- Appendix B on page 191 provides a series of sequence diagrams showing the API functions needed to update a configuration using CCBs.
- Appendix C on page 197 discusses compatibility issues that occur when object managers and object implementers use different versions of the IMM Service API.

⇒ To support notifications and correlation Ids, the following changes have been made to the IMM document:

- Section 4.2.19 on page 42 and Section 4.2.20 on page 43 have been introduced.
- The `correlationIds` parameter has been added to the functions `saImmOmAdminOperationInvoke_2()`, `saImmOmAdminOperationInvokeAsync_2()`, and `saImmOmCcbApply()`. As a consequence, these function have been superseded, see Section 4.9.1 on page 102 and Section 4.8.5 on page 95.
- The `notificationId` parameter has been added to the `SaImmOmAdminOperationInvokeCallbackT` function, which was superseded, see Section 4.9.2 on page 107. Due to this replacement, the `SaImmCallbacksT` type was replaced with the `SaImmCallbacksT_3` type (see Section 4.2.21 on page 43). This latter modification has led, in turn, to the replacement of the `saImmOmInitialize()` function with the `saImmOmInitialize_3()` function (see Section 4.3.1 on page 46).
- The `correlationIds` parameter has been added to the `SaImmOiAdminOperationCallbackT_2` function (which was superseded, see Section 5.8.2.1 on page 169) and to the `SaImmOiCcbApplyCallbackT_3` function (see Section 5.6.1 on page 151).
- Chapter 8 now contains notifications produced by the IMM Service.

1.3.2 Clarifications

Chapter 3 clarifies in a paragraph when runtime objects are persistent.

1.3.3 Deleted Topics

Chapter 9 of the IMM Service specification A.02.01 on the IMM Service management interface, which was intended to indicate that the SNMP MIBs were not yet available, has been removed, as no MIBs are provided in the IMM Service specification A.03.01.

1.3.4 Other Changes

- The description of all attributes of Section 4.2.22 on page 43 has been corrected to state that, even for configuration objects, these attributes are runtime attributes and not configuration attributes. The names of the two attributes `saImmAttrClassName` and `saImmAttrAdminOwnerName` were corrected to start with lower case.

- The value for the `SA_IMM_ADMIN_EXPORT` `operationId` of the `SA_IMM_ADMIN_EXPORT` administrative operation (see Section 6.4.1 on page 174) was missing in the previous version of this specification; it is now defined in Section 6.3.1 on page 173.
- In Section 7.2 on page 177, a correction has been made to state that the `SaImmMngt` is a configuration object class and not a runtime object class.
- In function signatures having an `in` parameter that is a pointer to another pointer *y* (or to an array of pointers *z*), the `const` qualifier has been added to the pertinent parameter to indicate that *y* (or *z*) must not be modified in the called function. The following non-superseded functions have been affected: `saImmOmClassCreate_2()`, `saImmOmAdminOwnerSet()`, `saImmOmAdminOwnerRelease()`, `saImmOmAdminOwnerClear()`, `saImmOmCcbObjectCreate_2()`, `saImmOmCcbObjectModify_2()`, `saImmOiRtObjectCreate_2()`, `saImmOiRtObjectUpdate_2()`.

1.3.5 Superseded and Superseding Functions

The IMM Service defines for the version A.03.01 new functions and new type definitions to replace functions and type definitions of the version A.02.01. The superseded functions and type definitions are no longer supported in version A.03.01, and no description is provided for them in this document. Regarding the support of backward compatibility in SA Forum AIS, refer to [2].

The list of replaced functions and type definitions is presented in alphabetic order in Table 1.

The names of the superseding functions and type definitions are obtained by adding "_3" to the respective names of the previous version or by replacing "_2" by "_3" if the superseded functions or type definitions had already "_2" at the end of their names. Exceptions to these rules are indicated by footnotes in Table 1.

Table 1 Superseded Functions and Type Definitions in Version A.03.01

Functions and Type Definitions of Version A.02.01 no Longer Supported in A.03.01
`SaImmCallbacksT`
`SaImmOiAdminOperationCallbackT_2`
`SaImmCcbFlagsT`
`SaImmOiCallbacksT_2`
`SaImmOiCcbAbortCallbackT`

Table 1 Superseded Functions and Type Definitions in Version A.03.01 (Continued)

Functions and Type Definitions of Version A.02.01 no Longer Supported in A.03.01
`SaImmOiCcbApplyCallbackT`
`SaImmOiCcbCompletedCallbackT`[1]
`SaImmOiCcbIdT`[2]
`SaImmOiCcbObjectCreateCallbackT_2`[3]
`SaImmOiCcbObjectDeleteCallbackT`[3]
`SaImmOiCcbObjectModifyCallbackT_2`[3]
`saImmOiClassImplementerRelease()`
`saImmOiClassImplementerSet()`
`saImmOiInitialize_2()`
`saImmOiObjectImplementerRelease()`
`saImmOiObjectImplementerSet()`
`saImmOmAccessorInitialize()`
`saImmOmAdminOperationInvoke_2()`
`saImmOmAdminOperationInvokeAsync_2()`
`saImmOmCcbApply()`
`saImmOmCcbInitialize()`
`saImmOmInitialize()`
`saImmOmSearchInitialize_2()`

1. The name of the superseding function is `SaImmOiCcbValidateCallbackT`
2. This type definition was replaced with the `SaImmCcbIdT` type.
3. This function has been removed. To a great extent, its functionality is now provided by the `SaImmOiCcbValidateCallbackT` function together with the object iterator API functions.

1.3.6 Changes in Return Values of API and Administrative Functions

The following table applies only to functions that have not been superseded.

Table 2 Changes in Return Values of API and Administrative Functions

Function	Return Value	Change Type
SA_IMM_ADMIN_EXPORT administrative operation	SA_AIS_ERR_INVALID_PARAM SA_AIS_ERR_TIMEOUT SA_AIS_ERR_NO_MEMORY	new
saImmOiImplementerSet()	SA_AIS_ERR_INIT	new
saImmOiRtObjectCreate_2(), saImmOiRtObjectUpdate_2()	SA_AIS_ERR_INVALID_PARAM	changed
saImmOiRtObjectCreate_2(), saImmOiRtObjectUpdate_2(), saImmOmAccessorGet_2(), saImmOmAdminOperationContinuationClear(), saImmOmAdminOperationContinue(), saImmOmAdminOperationContinueAsync(), saImmOmCcbObjectCreate_2(), saImmOmCcbObjectModify_2(), saImmOmClassCreate_2(), saImmOmClassDescriptionGet_2(), saImmOmClassDescriptionMemoryFree_2(), saImmOmSearchNext_2()	SA_AIS_ERR_VERSION[1]	new
saImmOmAdminOwnerFinalize()	SA_AIS_ERR_BUSY	new
saImmOmCcbObjectCreate_2(), saImmOmCcbObjectDelete(), saImmOmCcbObjectModify_2()	SA_AIS_ERR_FAILED_OPERATION	extended
saImmOmCcbObjectCreate_2(), saImmOmCcbObjectDelete(), saImmOmCcbObjectModify_2()	SA_AIS_ERR_TIMEOUT SA_AIS_ERR_BAD_OPERATION SA_AIS_ERR_NOT_EXIST	changed
saImmOmCcbObjectCreate_2(), saImmOmCcbObjectModify_2()	SA_AIS_ERR_INVALID_PARAM	changed

1. This return value should have been added in the IMM Service B.02.01 specification to all functions in this row.

1.4 References

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

[1] Service Availability™ Forum, Service Availability Interface, Overview, SAI-Overview-B.05.01

[2] Service Availability™ Forum, Service Availability Interface, C Programming Model, SAI-AIS-CPROG-B.05.01

[3] Service Availability™ Forum, Information Model in XML Metadata Interchange (XMI) v2.1 format, SAI-IM-XMI-A.04.01.xml.zip

[4] Service Availability™ Forum, IMM XML Schema Definition, SAI-AIS-IMM-XSD.A.01.01.xsd

[5] Service Availability™ Forum, Application Interface Specification, Cluster Membership Service, SAI-AIS-CLM-B.04.01

[6] Service Availability™ Forum, Application Interface Specification, Notification Service, SAI-AIS-NTF-A.03.01

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

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

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 Web site (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 SA Forum Web site (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 SA Forum Web site (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 SA Forum Web site (http://www.saforum.org).

2 Overview

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

The IMM Service is a cluster-wide service that must be highly-available in the sense that no single failure should take the entire service down.

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 SA Forum Information Model are provided with their attributes and administrative operations (that is, operations that can be performed on the represented entities through system management interfaces). For management applications or object managers, the IMM Service provides the APIs to create, access, and manage these objects.

The IMM Service 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 defines two sets of APIs:

(1) An object management API (OM-API), which is typically exposed to system management applications (for example, SNMP agents).

(2) An object implementer API (OI-API) restricted to object implementers.

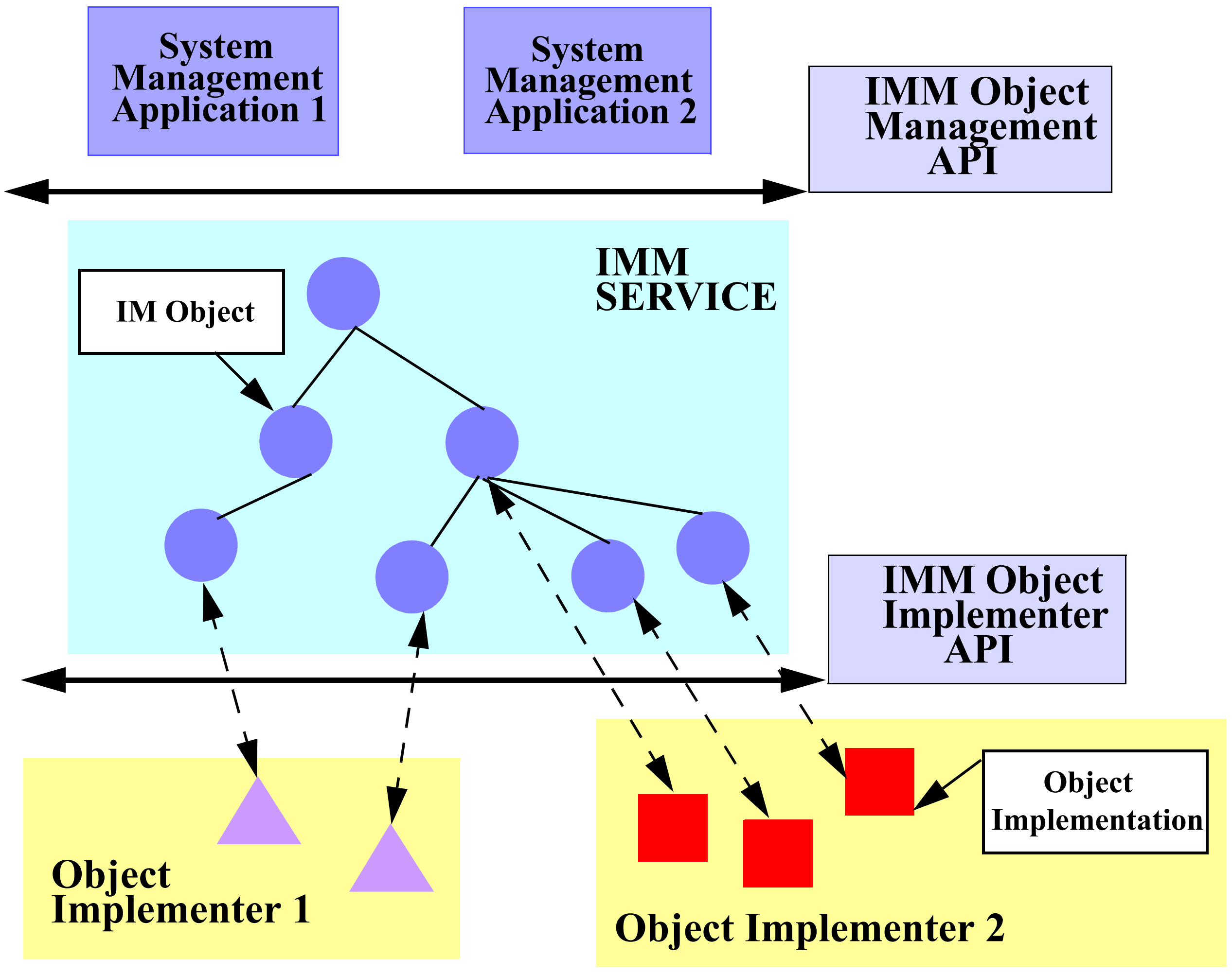
3 Information Model Management Service API

The Service Availability™ Forum (SA Forum) Information Model (IM) is specified in UML, and it is the collection of various managed objects that represent the logical entities of an SA Forum system. The SA Forum IM also specifies the attributes of these managed objects and administrative operations that can be performed on the entities they represent by using system management interfaces.

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

FIGURE 1 presents an overview of the interfaces provided by the IMM Service.

FIGURE 1 IMM Service Interfaces


System Management Application 1
System Management Application 2
IMM Object Management API
IMM SERVICE
IM Object
IMM Object Implementer API
Object Implementation
Object Implementer 1
Object Implementer 2

The implementation of the logical entities represented by the different managed objects in the SA Forum Information Model is not part of the IMM Service; instead, it is provided by user applications or other AIS Services such as the Checkpoint Service or the Availability Management Framework.

AIS Services and applications that implement the logical entities represented by IMM objects are called **object implementers** in the remainder of this document.

IMM objects are organized in a tree hierarchy. The hierarchy follows the structure of the LDAP distinguished name of each object. For more information about LDAP object names, refer to [2].

IMM objects and attributes can be classified into two categories:

- Configuration Objects and Attributes
    - **Configuration objects** and **configuration attributes** are the means by which system management applications provide input to an object implementer on the desired sets of objects and on their handling. The set of configuration objects and attributes constitute the prescriptive part of the SA Forum 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 **runtime attributes** are the means by which object implementers reflect in the SA Forum Information Model the current state of the entities they implement. The set of runtime objects and attributes constitute the descriptive part of the SA Forum Information Model. Runtime objects and attributes are typically under the control of object implementers.
    - Runtime objects that contain persistent runtime attributes or have persistent children (configuration or runtime objects) 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.

Object implementers cannot on their own initiative create and delete configuration objects or modify configuration attributes by using 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), which is typically exposed to system management applications (for example, SNMP agents).

(2) An **object implementer API** (OI-API), which is intended to be used by object implementers.

The OM-API is described in Chapter 4. The OI-API is found in Chapter 5.

The IMM Service acts as a mediator between object managers and object implementers. In particular, an object manager uses the object management API to carry out configuration changes and administrative operations on entities of the system implemented by different object implementers. In order to do this, an object manager manipulates the managed object of the SA Forum Information Model maintained by the IMM Service. In turn, the IMM Service propagates these manipulations to the appropriate object implementers for deployment using the object implementer API.
All configuration changes of IMM Service configuration objects are performed in the context of configuration change bundles (CCB). To carry out these configuration changes, object implementers are invoked by the IMM Service in different roles.
Object implementers that verify the correctness and consistency of CCBs act in the role of CCB validators.
Object implementers that deploy the validated CCB in the system act in the role of CCB appliers. An object implementer registers with the IMM Service in which role or roles it is going to act for the different configuration objects.

3.1 IMM Service State Transitions During CCB Processing

In this section, the term “manager” is used to represent an object management application.

When the IMM Service processes a CCB, the IMM Service associates a state machine with that CCB. This state machine goes through a number of state transitions (see FIGURE 2 on page 28).

Some of the state transitions are triggered by function calls, whereas other state transitions are triggered by the return values of the object implementer callback functions.

A manager initiates a set of configuration changes by calling the `saImmOmCcbInitialize_3()` function, which initializes the state machine of the IMM Service for the corresponding CCB and sets it to the **populate state**. The manager then calls the `saImmOmCcbObjectCreate_2()`, `saImmOmCcbObjectDelete()`, and `saImmOmCcbObjectModify()` functions to describe the modifications.

With the invocation of the `saImmOmCcbApply_3()` function, the manager indicates that the set of configuration changes is now complete and needs to be applied to the SA Forum Information Model. This call will initiate a procedure in which the IMM Service attempts to apply the modifications to the running configuration:

(1) The IMM Service serializes simultaneous CCB application requests, so that only one of them may enter the critical region at a time. The CCB that enters the critical region moves to the **transaction start state**.

(2) In the transaction start state, the IMM Service will call the `saImmOiCcbValidateCallback()` function on all involved CCB validators, which moves the CCB into the **transaction validate state**.

(3) If any of the CCB validators returns an error for the CCB, the IMM Service enters the **abort transaction state** to abort the current CCB. In this state,

- the IMM Service invokes the `saImmOiCcbAbortCallback()` functions of all involved CCB validators,
- the `saImmOmCcbApply_3()` function call returns with an error,
- the current CCB exits the critical region, and the IMM Service proceeds with the next CCB application request, if there is one pending.

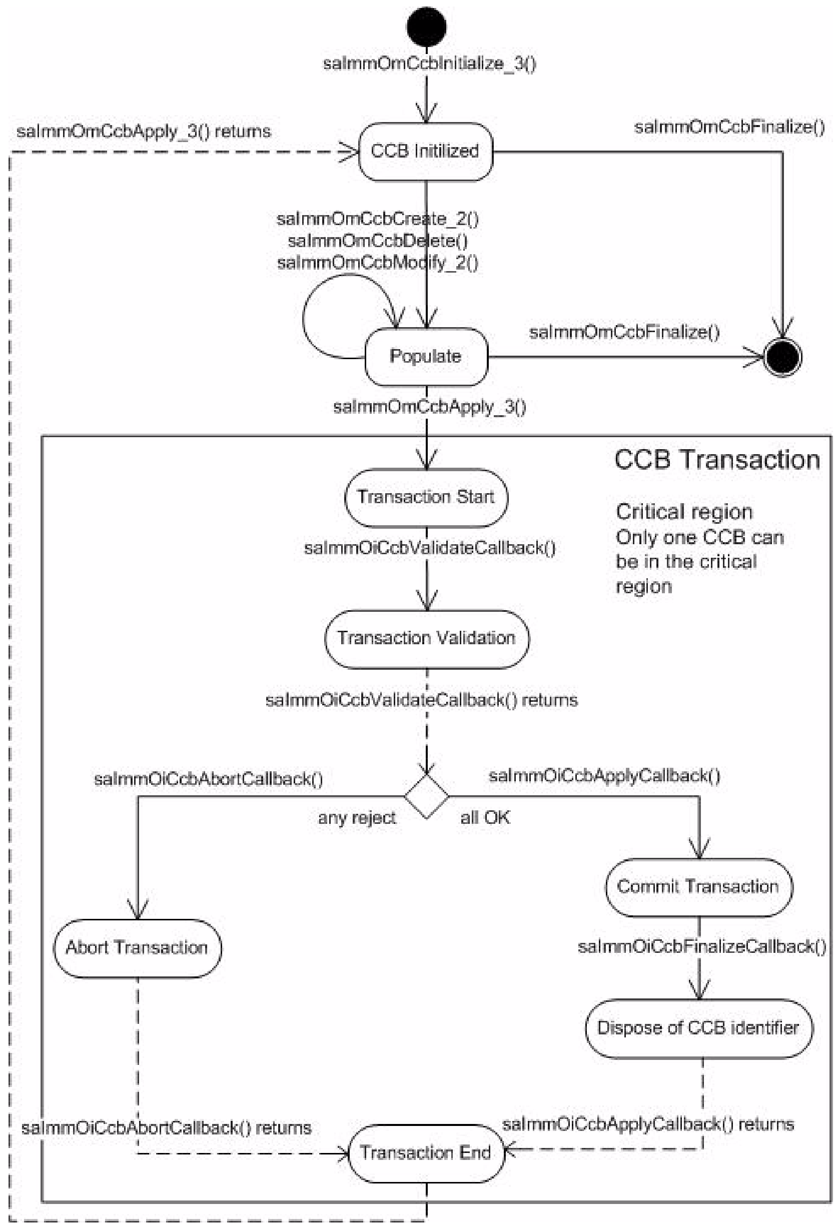
After the IMM Service exits the abort transaction state, all change requests associated with the CCB are removed, and the CCB is returned to the populate state, where it will remain until used for a new set of changes or finalized.

(4) If all callbacks from (2) return successfully, the CCB enters the **commit transaction state**, and the IMM Service starts notifying all relevant CCB appliers by invoking their `saImmOiCcbApplyCallback()` functions in the order specified by their rank in the IMM Service configuration (see Section 5.6 on page 150).

(5) After all the CCB applier callbacks have returned,

- the IMM Service invokes the `saImmOiCcbFinalizeCallback()` function of all involved CCB validators to release the CCB identifier and any associated state,
- the `saImmOmCcbApply_3()` function call returns successfully, which notifies the manager that the CCB has been successfully applied, and
- the current CCB exits the critical region, all changes associated with it are removed, and it is returned to the populate state, where it will remain until used to make a new set of changes or finalized.

FIGURE 2 CCB State Machine


saImmOmCcbInitialize_3()
saImmOmCcbApply_3() returns
CCB Initilized
saImmOmCcbFinalize()
saImmOmCcbCreate_2()
saImmOmCcbDelete()
saImmOmCcbModify_2()
Populate
saImmOmCcbFinalize()
saImmOmCcbApply_3()
CCB Transaction
Transaction Start
Critical region
Only one CCB can be in the critical region
saImmOiCcbValidateCallback()
Transaction Validation
saImmOiCcbValidateCallback() returns
saImmOiCcbAbortCallback()
saImmOiCcbApplyCallback()
any reject
all OK
Commit Transaction
Abort Transaction
saImmOiCcbFinalizeCallback()
Dispose of CCB identifier
saImmOiCcbAbortCallback() returns
saImmOiCcbApplyCallback() returns
Transaction End

A CCB validator process may also be a CCB applier for the same object. However, a CCB validator process may also validate other parts of the configuration of which it is not the CCB applier.

A CCB that is in the critical region is termed an **in-progress CCB**.

3.2 Object Naming

The Distinguished Name (DN) of an object (also simply called the object name) is constructed by prefixing the DN of the object's parent in the IMM tree hierarchy with the Relative Distinguished Name (RDN) of the object. The ',' character is used as a separator between the RDN of the object and the DN of its parent as follows:

Object_DN = "Object_RDN,Parent_Object_DN"

Objects that are immediately under the root of the IMM hierarchy have a DN that is equal to their RDN.

Each object must have one and only one attribute which is used to build the object RDN as follows:

Object_RDN = "RDN_attribute_name=RDN_attribute_value"

3.3 Internal Persistent Repository

The IMM Service maintains a copy of all its persistent entities (class definitions and persistent objects with their persistent attributes) within an **internal persistent repository** kept on stable storage. The storage holding the IMM Service persistent repository must be highly available, which implies storage replication. The nature of this internal repository is implementation-specific.

During startup of the IMM Service, the contents of its internal repository may be overwritten (or initialized if the internal repository was empty) from the contents of an XML file. It is implementation-specific how the XML file is provided to the IMM Service at startup. The XML file must conform to the **IMM XML Schema Definition** (see [4]). Such an XML file may be the result of the `SA_IMM_ADMIN_EXPORT` administrative operation (see Section 6.4.1 on page 174). If the XML file contains the description of non-persistent objects or attributes, these objects and attributes are ignored. The configuration parameter `saImmRepositoryInit` of the `SaImmMngt` object class (see Section 7.2 on page 177) specifies whether to overwrite or not the contents of the IMM Service internal repository at startup of the IMM Service.

When the IMM Service starts (for example, at the initial cluster startup or after a full cluster power-off), it contains only the class definitions and persistent objects with their persistent attributes that are present in its internal repository. Non-persistent runtime objects must be re-created by object implementers. The values of non-persistent runtime attributes (cached or not) will be obtained from the object implementers.

3.4 Unavailability of the IMM Service API on a Non-Member Node

This section describes the behavior of the IMM Service API from the point of view of a regular application process (as opposed to a middleware process implementing an AIS Service).

The behavior of the IMM Service API used by middleware processes that implement AIS Services is not specified and is left implementation-dependent. Processes implementing other AIS Services may need to access the IMM Service when a node that is not in the cluster membership is started up or shutdown.

The IMM Service does not provide service to regular application processes on cluster nodes that are not in the cluster membership (see [5]).

The following subsection describes the behavior of the IMM Service under various conditions that cause the IMM Service to be unavailable on a cluster node. Section 3.4.2 on page 32 contains guidelines for IMM Service implementers for dealing with a temporary unavailability of the service.

3.4.1 A Member Node Leaves or Rejoins the Cluster Membership

If the cluster node has left the cluster membership (see [5]) or is being administratively evicted from the cluster membership, the IMM Service behaves as follows towards processes residing on that cluster node and using or attempting to use the service:

- Calls to `saImmOmInitialize_3()` and `saImmOiInitialize_3()` will fail with `SA_AIS_ERR_UNAVAILABLE`.
- All IMM Service APIs that are invoked by the process and that operate on handles already acquired by the process will fail with `SA_AIS_ERR_UNAVAILABLE` with the following exceptions, assuming that the handle `immHandle` or the handle `immOiHandle` has already been acquired:
  - ⇒ The `saImmOmAdminOperationInvokeAsync_3()` function may return `SA_AIS_OK` or `SA_AIS_ERR_UNAVAILABLE`, depending on the service implementation. If it returns `SA_AIS_OK`, the `saImmOmAdminOperationInvokeCallback()` callback function of the process will be called and will also return `SA_AIS_ERR_UNAVAILABLE` in the error parameter; otherwise, the callback will not be called.
  - ⇒ The `saImmOmFinalize()` and `saImmOiFinalize()` functions, which are used to free the object management or object implementer library handles and all resources associated with these handles.
- An outstanding callback `saImmOmAdminOperationInvokeCallback()` will return `SA_AIS_ERR_UNAVAILABLE` in the error parameter.

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

When the cluster node leaves the membership, the IMM Service executing on the remaining nodes of the cluster behaves as if all processes that were using the IMM Service on the leaving cluster node had been terminated. In particular, if a process on the leaving cluster node was registered as an object implementer, the IMM Service will unregister it automatically (see Section 5.4.2 on page 132).

3.4.2 Guidelines for IMM Service Implementers

The implementation of the IMM Service must leverage the SA Forum Cluster Membership Service (see [5]) to determine the membership status of a cluster node for the case explained in Section 3.4.1 on page 31 before returning `SA_AIS_ERR_UNAVAILABLE`. If the Cluster Membership Service considers a cluster node as a member of the cluster but the IMM Service experiences difficulty in providing service to its clients because of transport, communication, or other issues, it must respond with `SA_AIS_ERR_TRY_AGAIN`.

4 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**. For its usage, refer to Section 4.8 on page 85.

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

The `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

The `SaImmClassCategoryT` type 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

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

```
#define SA_IMM_ATTR_MULTI_VALUE       0x00000001
#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 of type `SA_IMM_ATTR_SASTRINGT` or `SA_IMM_ATTR_SANAMET` 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 configuration 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 that has a default value.

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. This flag is automatically set by the IMM Service when the `SA_IMM_ATTR_PERSISTENT` flag is set.

4.2.6 SaImmAttrValueT

The `SaImmAttrValueT` type is used to represent the values of object attributes.

```
typedef void *SaImmAttrValueT;
```

4.2.7 SaImmAttrDefinitionT_2

The `SaImmAttrDefinitionT_2` type is used to specify the characteristics of an attribute belonging to a particular object class.

```
typedef struct {
     SaImmAttrNameT   attrName;
     SaImmValueTypeT  attrValueType;
     SaImmAttrFlagsT  attrFlags;
     SaImmAttrValueT  attrDefaultValue;
} SaImmAttrDefinitionT_2;
```

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.
- `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. A default value shall only be provided for configuration and persistent runtime attributes. Must be set to NULL if there is no default value for this attribute.

4.2.8 SaImmAttrValuesT_2

The `SaImmAttrValuesT_2` type is used to specify the values of one attribute of an object.

```
typedef struct {
	SaImmAttrNameT   attrName;
	SaImmValueTypeT  attrValueType;
	SaUint32T        attrValuesNumber;
	SaImmAttrValueT *attrValues;
} SaImmAttrValuesT_2;
```

The `attrName` field indicates the attribute name, the `attrValueType` field the type of the attribute, and the `attrValuesNumber` field the number of attribute values contained in the array of value descriptors to which `attrValues` points.

In order to be present within an object, an attribute must have at least one value. Optional attributes that have no value are not present in objects.

4.2.9 SaImmAttrModificationTypeT

The `SaImmAttrModificationTypeT` type 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 intention is to remove an attribute without specifying all its values, the SA_IMM_ATTR_REPLACE enum can be used.

- 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.

The SaImmAttrModificationTypeT type is used to specify the modification to apply on an object attribute.

4.2.10 SaImmAttrModificationT_2

```
typedef struct {
      SaImmAttrModificationTypeT  modType;
      SaImmAttrValuesT_2          modAttr;
} SaImmAttrModificationT_2;
```

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, or to be 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

The SaImmScopeT type 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

The `SaImmSearchOptionsT` is used to specify various options when performing searches amongst 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 holding 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 (see the `attributeNames` parameter in Section 4.5.1 on page 62).

4.2.13 SaImmSearchParametersT_2

The `SaImmSearchParametersT_2` type is used to provide the criteria parameters used for search operations.

```
typedef struct {
      SaImmAttrNameT   attrName;
      SaImmValueTypeT  attrValueType;
      SaImmAttrValueT  attrValue;
} SaImmSearchOneAttrT_2;
```

The `SaImmSearchOneAttrT_2` type contains the attribute description for `SA_IMM_SEARCH_ONE_ATTR` search operations. The fields `attrName` and `attrValue` specify the attribute name and value being searched for. The `attrValueType` field indicates the type of value that 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 to which `attrName` points, 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 to which `attrValue` points.

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 attribute values.

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

```
typedef union {
      SaImmSearchOneAttrT_2 searchOneAttr;
} SaImmSearchParametersT_2;
```

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_3

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

```
#define SA_IMM_CCB_ALLOW_ABSENT_VALIDATORS       0x00000001
#define SA_IMM_CCB_ALLOW_ABSENT_APPLIERS         0x00000002
typedef SaUint64T SaImmCcbFlagsT_3;
```

`SA_IMM_CCB_ALLOW_ABSENT_VALIDATORS`—if this flag is specified, the CCB can hold changes for objects for which CCB validators have been specified but are currently not registered. If this flag is not set, all specified validators must be registered for all objects that are changed in the CCB.

`SA_IMM_CCB_ALLOW_ABSENT_APPLIERS`—if this flag is specified, the CCB can hold changes for objects for which CCB appliers have been specified but are currently not registered. If this flag is not set, all specified appliers must be registered for all objects that are changed in the CCB.

4.2.15 SaImmCcbIdT

```
typedef SaUint64T SaImmCcbIdT;
```

This type is used to represent a CCB identifier associated with a particular configuration change bundle (CCB).

4.2.16 SaImmContinuationIdT

```
typedef SaUint64T SaImmContinuationIdT;
```

The type `SaImmContinuationIdT` is used to identify a particular invocation of an administrative operation on an IMM object. Its scope is cluster-wide, and it must be unique on a per-IMM object basis. For more details, refer to Section 4.9 on page 100.

4.2.17 SaImmAdminOperationIdT

The `SaImmAdminOperationIdT` type 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.

The IMM Service is not aware of the valid range of operation identifiers of an object class.

```
typedef SaUint64T SaImmAdminOperationIdT;
```

4.2.18 SaImmAdminOperationParamsT_2

The `SaImmAdminOperationParamsT_2` type is used to specify the parameters of an administrative operation performed on an object.

```
typedef struct {
        SaStringT        paramName;
        SaImmValueTypeT  paramType;
        SaImmAttrValueT  paramBuffer;
} SaImmAdminOperationParamsT_2;
```

The `paramName` field indicates the name of the parameter. The `paramType` field indicates the type of the parameter. The `paramBuffer` field contains the parameter value.

4.2.19 SaImmNotificationMinorIdT

```
typedef enum {
        SA_IMM_NTFID_OP_START               = 1,
        SA_IMM_NTFID_OP_END                 = 2,
        SA_IMM_NTFID_CCB_APPLY_START        = 3,
        SA_IMM_NTFID_CCB_APPLY_END          = 4
} SaImmNotificationMinorIdT;
```

This type provides the values for the `minorId` field of notification class Identifiers used by the IMM Service.

4.2.20 SaImmAdditionalInfoIdT

```
typedef enum {
     SA_IMM_AI_ADMIN_OPERATION_ID     = 1,
     SA_IMM_AI_ADMIN_OPERATION_PARAM  = 2,
     SA_IMM_AI_ADMIN_OPERATION_RESULT = 3,
     SA_IMM_AI_CCB_ID                 = 4,
     SA_IMM_AI_CCB_RETURN_VALUE       = 5
} SaImmAdditionalInfoIdT;
```

This type provides identifiers for the data that is part of the additional information portion of notifications sent by the IMM Service.

4.2.21 SaImmCallbacksT_3

The `SaImmCallbacksT_3` structure defines the set of callbacks a process can provide to the IMM Service at initialization time.

```
typedef struct {
     SaImmOmAdminOperationInvokeCallbackT_3
          saImmOmAdminOperationInvokeCallback;
} SaImmCallbacksT_3;
```

4.2.22 IMM Service Object Attributes

The following #define directives are used to refer to the name of attributes of objects in the SA Forum Information Model.

```
#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.

The two preceding attributes are single-value attributes and their value is of type `SA_IMM_ATTR_SASTRINGT`.

```
#define SA_IMM_ATTR_VALIDATOR_NAME
     "saImmAttrValidatorName"
```

When an object has CCB validators, the IMM Service stores the names of the CCB validators in one multi-valued attribute of the object. The name of this attribute is specified by the constant `SA_IMM_ATTR_VALIDATOR_NAME`, and it is of type `SA_IMM_ATTR_SASTRINGT`. This attribute does not exist in objects having no CCB validators.

```
#define SA_IMM_ATTR_APPLIER_NAME
     "saImmAttrApplierName"
```

When an object has CCB appliers, the IMM Service stores the names of the CCB appliers in one multi-valued attribute of the object. The name of this attribute is specified by the constant `SA_IMM_ATTR_APPLIER_NAME`, and it is of type `SA_IMM_ATTR_SASTRINGT`. This attribute does not exist in objects having no CCB appliers.

```
#define SA_IMM_ATTR_RUNTIME_OWNER_NAME
     "saImmAttrRuntimeOwnerName"
```

When an object has a runtime owner, the IMM Service stores the name of the runtime owner in one single-valued attribute of the object. The name of this attribute is specified by the constant `SA_IMM_ATTR_RUNTIME_OWNER_NAME`, and it is of type `SA_IMM_ATTR_SASTRINGT`. This attribute does not exist in objects having no runtime owner.

All these attributes are runtime attributes, and for persistent objects (that is, configuration and persistent runtime objects), these attributes are persistent runtime attributes; otherwise, they are non-persistent runtime attributes.

4.2.23 SaImmRepositoryInitModeT

```
typedef enum {
     SA_IMM_KEEP_REPOSITORY     = 1,
     SA_IMM_INIT_FROM_FILE      = 2
} SaImmRepositoryInitModeT;
```

The values of `SaImmRepositoryInitModeT` specify how the IMM Service initializes its internal repository when the IMM Service starts up.

- `SA_IMM_KEEP_REPOSITORY`: at startup, the IMM Service keeps the contents of its internal repository.

- `SA_IMM_INIT_FROM_FILE`: at startup, the IMM Service must overwrite the contents of its internal repository with the contents of an XML file. The location of this initial XML file is implementation-dependent.

4.3 Library Life Cycle

4.3.1 saImmOmInitialize_3()

Prototype

```
SaAisErrorT saImmOmInitialize_3(
     SaImmHandleT *immHandle,
     const SaImmCallbacksT_3 *immCallbacks,
     SaVersionT *version
);
```

Parameters

`immHandle` - [`out`] A pointer to the handle which identifies this particular initialization of the IMM Service and which is to be returned by the IMM Service. This handle provides access to the object management APIs of the IMM Service. The `SaImmHandleT` type is defined in Section 4.2.1 on page 33.

`immCallbacks` - [`in`] If `immCallbacks` is set to NULL, no callback is registered; if `immCallbacks` is not set to NULL, it is a pointer to an `SaImmCallbacksT_3` structure which contains the callback functions of the process that the IMM Service may invoke. Only non-NULL callback functions in this structure will be registered. The `SaImmCallbacksT_3` type is defined in Section 4.2.21 on page 43.

`version` - [`in/out`] As an input parameter, `version` is a pointer to a structure containing the required IMM Service version. In this case, `minorVersion` is ignored and should be set to 0x00.
As an output parameter, `version` is a pointer to a structure containing the version actually supported by the IMM Service. The `SaVersionT` type is defined in [2].

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 pointed to by `immHandle` is returned by the IMM Service 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_3()` function and before invoking `saImmOmFinalize()` to finalize the handle `immHandle` (see Section 4.3.4 on page 52), the IMM Service automatically finalizes this handle and any other handles that have been acquired using the handle `immHandle` when the IMM Service detects the death of the process.

If the implementation supports the version of the Information Model Management Service API specified by the `releaseCode` and `majorVersion` fields of the structure pointed to by the `version` parameter, `SA_AIS_OK` is returned. In this case, the structure pointed to by the `version` parameter is set by this function to:

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

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

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

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

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

Return Values

`SA_AIS_OK` - The function completed successfully.

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

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

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

`SA_AIS_ERR_INVALID_PARAM` - A parameter is not set correctly.

`SA_AIS_ERR_NO_MEMORY` - Either the 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 provided in the structure to which the `version` parameter points is not compatible with the version of the Information Model Management Service implementation.

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

See Also

`saImmOmSelectionObjectGet()`, `saImmOmDispatch()`, `saImmOmFinalize()`

4.3.2 saImmOmSelectionObjectGet()

Prototype

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

Parameters

`immHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOmInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmHandleT` type is defined in Section 4.2.1 on page 33.

`selectionObject` - [out] A pointer to the operating system handle that the process can use to detect pending callbacks. The `SaSelectionObjectT` type is defined in [2].

Description

This function returns the operating system handle associated with the handle `immHandle`. The invoking process can use the operating system 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 operating system handle returned by `saImmOmSelectionObjectGet()` is valid until `saImmOmFinalize()` is successfully 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).

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle immHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOmInitialize_3(), saImmOmDispatch(), saImmOmFinalize()

4.3.3 saImmOmDispatch()

Prototype

```
SaAisErrorT saImmOmDispatch(
     SaImmHandleT immHandle,
     SaDispatchFlagsT dispatchFlags
);
```

Parameters

immHandle - [in] The handle which was obtained by a previous invocation of the saImmOmInitialize_3() function and which identifies this particular initialization of the Information Model Management Service. The SaImmHandleT type is defined in Section 4.2.1 on page 33.

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. These flags are values of the SaDispatchFlagsT enumeration type, which is described in [2].

Description

In the context of the calling thread, this function invokes 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. This value is also returned if this function is being invoked with `dispatchFlags` set to `SA_DISPATCH_ALL` or `SA_DISPATCH_BLOCKING`, and the handle `immHandle` has been finalized.

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

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

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

`SA_AIS_ERR_BAD_HANDLE` - The handle `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_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmInitialize_3()`, `saImmOmSelectionObjectGet()`, `saImmOmFinalize()`

4.3.4 saImmOmFinalize()

Prototype

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

Parameters

`immHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOmInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmHandleT` type is defined in Section 4.2.1 on page 33.

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_3()` before it invokes this function. A process must invoke this function once for each handle it acquired by invoking `saImmOmInitialize_3()`.

If the `saImmOmFinalize()` function completes successfully, it releases all resources acquired when `saImmOmInitialize_3()` 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 handle `immHandle` and the selection object associated with it are no longer valid.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

See Also

saImmOmInitialize_3()

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_2()

Prototype

```
SaAisErrorT saImmOmClassCreate_2(
     SaImmHandleT immHandle,
     const SaImmClassNameT className,
     SaImmClassCategoryT classCategory,
     const SaImmAttrDefinitionT_2 *const *attrDefinitions
);
```

Parameters

`immHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOmInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmHandleT` type is defined in Section 4.2.1 on page 33.

`className` - [in] The name of the object class to create. The `SaImmClassNameT` type is defined in Section 4.2.2 on page 34.

`classCategory` - [in] Category of the object class. The `SaImmClassCategoryT` type is defined in Section 4.2.4 on page 34.

`attrDefinitions` - [in] Pointer to a NULL-terminated array of pointers to definitions of the class attributes. The `SaImmAttrDefinitionT_2` type is defined in Section 4.2.7 on page 36.

Description

This function creates a new object class with the 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 refers to a NULL or zero length attribute name, an invalid value type, an invalid default attribute value, or a set of attribute flags that 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.

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmInitialize_3()`

4.4.2 saImmOmClassDescriptionGet_2()

Prototype

```
SaAisErrorT saImmOmClassDescriptionGet_2(
      SaImmHandleT immHandle,
      const SaImmClassNameT className,
      SaImmClassCategoryT *classCategory,
      SaImmAttrDefinitionT_2 ***attrDefinitions
);
```

Parameters

`immHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOmInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmHandleT` type is defined in Section 4.2.1 on page 33.

`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 34.

`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 34.

`attrDefinitions` - [out] Pointer to a pointer to a NULL-terminated array of pointers to definitions of the class attributes. The `SaImmAttrDefinitionT_2` type is defined in Section 4.2.7 on page 36.

Description

This function returns a description of the object class identified by the 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_2()` 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 - No object class exists with a name identical to className.

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

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle immHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOmInitialize_3(), saImmOmClassCreate_2(), saImmOmClassDescriptionMemoryFree_2()

4.4.3 saImmOmClassDescriptionMemoryFree_2()

Prototype

```
SaAisErrorT saImmOmClassDescriptionMemoryFree_2(
     SaImmHandleT immHandle,
     SaImmAttrDefinitionT_2 **attrDefinitions
);
```

Parameters

`immHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOmInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmHandleT` type is defined in Section 4.2.1 on page 33.

`attrDefinitions` - [in] Pointer to a NULL-terminated array of pointers to attribute definitions to be freed. The `SaImmAttrDefinitionT_2` type is defined in Section 4.2.7 on page 36.

Description

This function deallocates the memory that was allocated by a previous call to the `saImmOmClassDescriptionGet_2()` function; this deallocation includes

- the memory areas containing the attribute definitions which are referred to by the pointers held in the NULL-terminated array referred to by `attrDefinitions` and
- the memory of the NULL-terminated array of pointers referred to by `attrDefinitions`.

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.

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

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle immHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOmInitialize_3(), saImmOmClassCreate_2(), saImmOmClassDescriptionGet_2()

4.4.4 saImmOmClassDelete()

Prototype

```
SaAisErrorT saImmOmClassDelete(
      SaImmHandleT immHandle,
      const SaImmClassNameT className
);
```

Parameters

immHandle - [in] The handle which was obtained by a previous invocation of the saImmOmInitialize_3() function and which identifies this particular initialization of the Information Model Management Service. The SaImmHandleT type is defined in Section 4.2.1 on page 33.

className - [in] Name of the object class to be deleted. The SaImmClassNameT type is defined in Section 4.2.2 on page 34.

Description

This function deletes the object class whose name is className, provided no objects of this class exist.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

`SA_AIS_ERR_BAD_HANDLE` - The handle `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` - No object class exists 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.

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmInitialize_3()`, `saImmOmClassCreate_2()`

4.5 Object Search

The API functions in this section are used to perform **object search**, that is, to search for particular objects in the SA Forum Information Model and also to obtain the values of some of their attributes.

When the object search is initialized, a valid CCB identifier (a nonzero value) can be supplied. If it is supplied, the object search will behave as if the changes proposed that far by the CCB in question had already been applied to the SA Forum Information Model. When these object search API functions are used during validation of a CCB (refer to Section 5.5 on page 144), it is guaranteed that no changes of other pending CCBs can affect the object search, because no two CCBs are allowed to be in the validate state (see Section 3.1 on page 26) at the same time. The `saImmOmSearchNext_2()` function returns objects, their attributes, and values of these attributes modified according to the changes proposed that far by the CCB in question.
If zero is provided instead of a valid CCB identifier when the object search is initialized, the object search refers to the current state of the SA Forum Information Model. Pending CCB changes, that is, CCB changes that have not yet been applied, are invisible to the object search.

To facilitate the management of the memory allocated by the IMM Service library to return the results of the search, the search is performed by using 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 by invoking the finalize API.

Every object which was created before the invocation of the `saImmOmSearchInitialize_3()` function and 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_2()` search iterator. No other guarantees are made: objects that are created after the iteration is initialized, or modified, or deleted before the iteration is finalized, may or may not be returned by the search iterator.

4.5.1 saImmOmSearchInitialize_3()

Prototype

```
SaAisErrorT saImmOmSearchInitialize_3(
      SaImmHandleT immHandle,
      SaImmCcbIdT ccbId,
      const SaNameT *rootName,
      SaImmScopeT scope,
      SaImmSearchOptionsT searchOptions,
      const SaImmSearchParametersT_2 *searchParam,
      const SaImmAttrNameT *attributeNames,
      SaImmSearchHandleT *searchHandle
);
```

Parameters

`immHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOmInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmHandleT` type is defined in Section 4.2.1 on page 33.

`ccbId` - [in] A valid CCB identifier or zero. The `SaImmCcbIdT` type is defined in Section 4.2.15 on page 41.

`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. The `SaNameT` type is defined in [2].

`scope` - [in] Scope of the search. The `SaImmScopeT` type is defined in Section 4.2.11 on page 38.

`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 39.

`searchParam` - [in] A pointer to the search parameters according to the search criteria specified in `searchOptions`. The `SaImmSearchParametersT_2` type is defined in Section 4.2.13 on page 40.

`attributeNames` - [in] Pointer to a 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. The `attributeNames` pointer must be set to NULL otherwise.
The `SaImmAttrNameT` type is defined in Section 4.2.2 on page 34.

`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 33.

Description

This function initializes a search operation limited to a set of targeted objects identified by the `scope` parameter and the name to which the `rootName` parameter points.

If the `ccbId` parameter is zero, the object search refers to the current state of the SA Forum Information Model. Pending CCB changes, that is, CCB changes that have not yet been applied, are invisible to the object search.
If the `ccbId` parameter specifies a valid CCB identifier (a nonzero value), the object search operation behaves as if the modifications proposed that far in the CCB identified by the `ccbId` parameter have already been applied to the SA Forum Information Model.

The targeted set of objects is determined as follows:

- If `scope` is `SA_IMM_SUBLEVEL`, the scope of the operation is the object having the name to which `rootName` points and its direct children.
- If `scope` is `SA_IMM_SUBTREE`, the scope of the operation is the object having the name to which `rootName` points and the entire subtree rooted at that object.
- `SA_IMM_ONE` is not a valid value for the `scope` parameter.

If the `SA_IMM_SEARCH_ONE_ATTR` option is not set in the `searchOptions` parameter, the `searchOptions` parameter must be set to NULL. In this case, no selection criteria is applied for the search, and all objects in the defined scope will be retrieved by the search operation.

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. In particular, this error is returned if the specified `ccbId` parameter is nonzero and unknown to the IMM Service.

`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` - The name to which `rootName` points is not the name of an existing object.

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmInitialize_3()`

4.5.2 saImmOmSearchNext_2()

Prototype

```
SaAisErrorT saImmOmSearchNext_2(
      SaImmSearchHandleT searchHandle,
      SaNameT *objectName,
      SaImmAttrValuesT_2 ***attributes
);
```

Parameters

`searchHandle` - [in] Handle returned by `saImmOmSearchInitialize_3()`. The `SaImmSearchHandleT` type is defined in Section 4.2.1 on page 33.

`objectName` - [out] Pointer to the name of the next object matching the search criteria. The `SaNameT` type is defined in [2].

`attributes` - [out] Pointer to a pointer to a NULL-terminated array of pointers to data structures holding the names and values of the attributes (of the object whose name is pointed to by `objectName`) that were selected when the search was initialized. The `SaImmAttrValuesT_2` type is defined in Section 4.2.8 on page 37.

Description

This function is used to obtain the next object matching the search criteria that was specified in the corresponding `saImmOmSearchInitialize_3()` call.

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_2()` or `saImmOmSearchFinalize()` for the same search handle.

If the handle `searchHandle` was not obtained by specifying `SA_IMM_SEARCH_GET_ALL_ATTR` or `SA_IMM_SEARCH_GET_SOME_ATTR` in the `searchOptions` parameter of the corresponding `saImmOmSearchInitialize_3()` call, no attribute names and values will be returned by this call, and the pointer to which the `attributes` parameter refers is set to NULL.

Only the attribute name is returned (`attrValuesNumber` is set to 0 and `attrValues` is set to NULL in the `SaImmAttrValuesT_2` data structure referred to by the corresponding entry in the array whose address is referred to by the `attributes` parameter) if one of the attributes requested by the search

- has no value or
- is a non-persistent runtime attribute, and no runtime owner is registered for the 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 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_2()`.

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `searchHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmInitialize_3()`, `saImmOmSearchInitialize_3()`, `saImmOmSearchFinalize()`

4.5.3 saImmOmSearchFinalize()

Prototype

```
SaAisErrorT saImmOmSearchFinalize(
     SaImmSearchHandleT searchHandle
);
```

Parameters

`searchHandle` - [in] Handle returned by `saImmOmSearchInitialize_3()`. The `SaImmSearchHandleT` type is defined in Section 4.2.1 on page 33.

Description

This function finalizes the search initialized by a previous call to `saImmOmSearchInitialize_3()`. It frees all memory previously allocated by that search, in particular, the memory used to return attribute names and values in the previous `saImmOmSearchNext_2()` 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.

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `searchHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmInitialize_3()`, `saImmOmSearchInitialize_3()`, `saImmOmSearchNext_2()`

4.6 Object Access

The API functions in this section are used to perform **object access**, that is, 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.

When the object access is initialized, a valid CCB identifier (a nonzero value) can be supplied. If it is supplied, the object access will behave as if the changes proposed that far by the CCB in question had already been applied to the SA Forum Information Model. When these object access API functions are used during validation of a CCB (refer to Section 5.5 on page 144), it is guaranteed that no changes of other pending CCBs can affect the object access, because no two CCBs are allowed to be in the validate state (see Section 3.1 on page 26) at the same time. The `saImmOmAccessorGet_2()` function returns the values of the configuration attributes modified according to the changes proposed that far by the CCB in question.
If zero is provided instead of a valid CCB identifier when the object access is initialized, the object access applies to the current state of the SA Forum Information Model. Pending CCB changes, that is, CCB changes that have not yet been applied, are invisible to the object access.

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_3()

Prototype

```
SaAisErrorT saImmOmAccessorInitialize_3(
     SaImmHandleT immHandle,
     SaImmCcbIdT ccbId,
     SaImmAccessorHandleT *accessorHandle
);
```

Parameters

`immHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOmInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmHandleT` type is defined in Section 4.2.1 on page 33.

`ccbId` - `[in]` A valid CCB identifier or zero. The `SaImmCcbIdT` type is defined in Section 4.2.15 on page 41.

`accessorHandle` - `[out]` Pointer to the object accessor handle. The `SaImmAccessorHandleT` type is defined in Section 4.2.1 on page 33.

Description

This function initializes an object accessor and returns the handle pointed to by the `accessorHandle` parameter for further references to the object accessor.

If the `ccbId` parameter is zero, the object access refers to the current state of the SA Forum Information Model. Pending CCB changes, that is, CCB changes that have not yet been applied, are invisible to the object access.
If the `ccbId` parameter specifies a valid CCB identifier (a nonzero value), the object access operation behaves as if the modifications proposed that far in the CCB identified by the `ccbId` parameter have already been applied to the SA Forum Information Model.

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, this error is returned if the specified `ccbId` parameter is nonzero and unknown to the IMM Service.

`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 invoked function is not supported in the version specified in the call to initialize this instance of the IMM Service library.

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmInitialize_3()`

4.6.2 saImmOmAccessorGet_2()

Prototype

```
SaAisErrorT saImmOmAccessorGet_2(
      SaImmAccessorHandleT accessorHandle,
      const SaNameT *objectName,
      const SaImmAttrNameT *attributeNames,
      SaImmAttrValuesT_2 ***attributes
);
```

Parameters

`accessorHandle` - [in] Object accessor handle. The `SaImmAccessorHandleT` type is defined in Section 4.2.1 on page 33.

`objectName` - [in] Pointer to the name of the object being accessed. The `SaNameT` type is defined in [2].

`attributeNames` - [in] Pointer to a NULL-terminated array of attribute names for which values must be returned. The `SaImmAttrNameT` type is defined in Section 4.2.2 on page 34.

`attributes` - [out] Pointer to a pointer to a NULL-terminated array of pointers to data structures containing the name and values of the attributes being accessed. The `SaImmAttrValuesT_2` type is defined in Section 4.2.8 on page 37.

Description

This function uses the object accessor referred to by the `accessorHandle` parameter 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 runtime owner for the object, only the attribute name is returned (`attrValuesNumber` is set to 0 and `attrValues` is set to NULL in the `SaImmAttrValuesT_2` 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_2()` 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` - The name to which `objectName` points is not the name of an existing object, or any of the names specified by `attributeNames` does not exist for the object identified by the name to which `objectName` points.

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;

- the cluster node has rejoined the cluster membership, but the handle `accessorHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmAccessorInitialize_3()`

4.6.3 saImmOmAccessorFinalize()

Prototype

```
SaAisErrorT saImmOmAccessorFinalize(
     SaImmAccessorHandleT accessorHandle
);
```

Parameters

`accessorHandle` - [in] Object accessor handle. The `SaImmAccessorHandleT` type is defined in Section 4.2.1 on page 33.

Description

This function finalizes the object accessor referred to by the `accessorHandle` parameter and deallocates all memory previously allocated for this object accessor. In particular, this function frees the memory used to return the object attribute names and values during the previous invocation of `saImmOmAccessorGet_2()`.

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_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `accessorHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmAccessorInitialize_3()`

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 registered runtime owner. 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 (by invoking the `saImmOmAdminOwnerClear()` function) without acting under the name of their current administrative owner.

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 which was obtained by a previous invocation of the `saImmOmInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmHandleT` type is defined in Section 4.2.1 on page 33.

`adminOwnerName` - [`in`] Name of the administrative owner. The `SaImmAdminOwnerNameT` type is defined in Section 4.2.2 on page 34.

`releaseOwnershipOnFinalize` - [`in`] This parameter specifies how to release administrative ownerships that were acquired with the newly initialized handle `ownerHandle` when this handle is finalized. The `SaBoolT` type is defined in [2].

`ownerHandle` - [`out`] Pointer to the handle for the administrative owner. The `SaImmAdminOwnerHandleT` type is defined in Section 4.2.1 on page 33.

Description

This function initializes a handle for an administrative owner whose name is specified by `adminOwnerName`. All objects owned by an administrative owner have the attribute whose name is defined by the constant `SA_IMM_ATTR_ADMIN_OWNER_NAME` set to the name of the administrative owner. 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 the administrative ownership on these 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 `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_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmInitialize_3()`, `saImmOmAdminOwnerSet()`, `saImmOmAdminOwnerFinalize()`

4.7.2 saImmOmAdminOwnerSet()

Prototype

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

Parameters

`ownerHandle` - [in] Administrative owner handle. The `SaImmAdminOwnerHandleT` type is defined in Section 4.2.1 on page 33.

`objectNames` - [in] Pointer to a NULL-terminated array of pointers to object names. The `SaNameT` type is defined in [2].

`scope` - [`in`] Scope of the operation. The `SaImmScopeT` type is defined in Section 4.2.11 on page 38.

Description

This function sets the administrative owner identified by `ownerHandle` as the owner of the set of objects identified by the `scope` and the `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 having names specified by `objectNames`.
- If `scope` is `SA_IMM_SUBLEVEL`, the scope of the operation are the objects having names specified by `objectNames` and their direct children.
- If `scope` is `SA_IMM_SUBTREE`, the scope of the operation are the objects having names specified 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 that was 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 names specified 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`.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `ownerHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmAdminOwnerInitialize()`, `saImmOmAdminOwnerRelease()`, `saImmOmAdminOwnerClear()`

4.7.3 saImmOmAdminOwnerRelease()

Prototype

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

Parameters

`ownerHandle` - [in] Administrative owner handle. The `SaImmAdminOwnerHandleT` type is defined in Section 4.2.1 on page 33.

`objectNames` - [in] Pointer to a NULL-terminated array of pointers to object names. The `SaNameT` type is defined in [2].

`scope` - [in] Scope of the operation. The `SaImmScopeT` type is defined in Section 4.2.11 on page 38.

Description

This function releases the administrative owner of the set of objects identified 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 having names specified by `objectNames`.
- If `scope` is `SA_IMM_SUBLEVEL`, the scope of the operation are the objects having names specified by `objectNames` and their direct children.
- If `scope` is `SA_IMM_SUBTREE`, the scope of the operation are the objects having names specified 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, and the continuation identifiers registered for these objects (see Section 4.2.16 on page 41) are all cleared.

The operation fails if an administrative operation is currently in progress on one of the targeted objects. An administrative operation is considered to be **in progress** on an object if the `saImmOiAdminOperationCallback()` object implementer's callback has been invoked for that operation, and the registered runtime owner is still registered but has not yet called `saImmOiAdminOperationResult()` to provide the operation results. The operation also fails if a change request for one of the targeted objects is included in a CCB that has not been finalized.

If the operation fails, the administrative owner of all objects in the given scope remains unchanged.

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 names specified 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.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle ownerHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOmAdminOwnerInitialize(), saImmOmAdminOwnerSet()

4.7.4 saImmOmAdminOwnerFinalize()

Prototype

```
SaAisErrorT saImmOmAdminOwnerFinalize(
     SaImmAdminOwnerHandleT ownerHandle
);
```

Parameters

ownerHandle - [in] Administrative owner handle. The SaImmAdminOwnerHandleT type is defined in Section 4.2.1 on page 33.

Description

This function releases ownerHandle. If ownerHandle has been initialized with the releaseOwnershipOnFinalize option set to SA_FALSE, this function neither affects registered continuation identifiers of any object nor releases the administrative ownership 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 and clears all continuation identifiers registered for these objects.

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.

SA_AIS_ERR_BUSY - ownerHandle has been initialized with the releaseOwnershipOnFinalize option set to SA_TRUE, and 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.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle ownerHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOmAdminOwnerInitialize(), saImmOmCcbInitialize_3()

4.7.5 saImmOmAdminOwnerClear()

Prototype

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

Parameters

`immHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOmInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmHandleT` type is defined in Section 4.2.1 on page 33.

`objectNames` - [in] Pointer to a NULL-terminated array of pointers to object names. The `SaNameT` type is defined in [2].

`scope` - [in] Scope of the operation. The `SaImmScopeT` type is defined in Section 4.2.11 on page 38.

Description

This function clears the administrative owner of the set of objects identified 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 having names specified by `objectNames`.
- If `scope` is `SA_IMM_SUBLEVEL`, the scope of the operation are the objects having names specified by `objectNames` and their direct children.
- If `scope` is `SA_IMM_SUBTREE`, the scope of the operation are the objects having names specified 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, and the continuation identifiers registered for these objects are all cleared.

The operation fails if an administrative operation is currently in progress on one of the targeted objects (for the term “in progress”, see Section 4.7.3 on page 79), 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 all objects in the given scope remains unchanged.

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 names specified 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.

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmAdminOwnerInitialize()`, `saImmOmAdminOwnerSet()`, `saImmOmAdminOwnerRelease()`

4.8 Configuration Changes

All changes of IMM Service configuration objects are performed in the context of **configuration change bundles** (**CCB**). A CCB is associated with a single administrative owner, and all objects changed by the CCB must have the same administrative owner as the CCB. Once a CCB has been initialized, change requests can be added to the CCB. A **change request** can be a creation, a deletion, or a modification. A CCB that has been initialized, but not yet applied, is called a **pending CCB,** and it may contain any number of pending change requests. 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).

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

The IMM Service does not prevent applications from reading (by invoking `saImmOmSearchNext_2()` or `saImmOmAccessorGet_2()`) the attribute values of the objects modified by a CCB while a CCB is being applied. Therefore, it may happen, for example, that a search operation returns 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_2()` or `saImmOmAccessorGet_2()` for a <u>particular</u> object <u>must all</u> 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 one or several administrative operations.

The application of a CCB can succeed only if all specified CCB validators and CCB appliers are registered for all the objects that are changed by the CCB. The CCB fails if any object changed by the CCB has a validator or an applier specified for which there is no process currently registered with the IMM Service. This requirement can be overruled if the object manager application sets the `SA_IMM_CCB_ALLOW_ABSENT_VALIDATORS` or `SA_IMM_CCB_ALLOW_ABSENT_APPLIERS` flags when it initializes the CCB. These flags indicate that any registration may be missing among the CCB validators or CCB appliers, respectively.

A CCB is aborted under the following conditions:

- The `saImmOmCcbFinalize()` function is called before the CCB is applied.

- A CCB validator does not respond in time or rejects the changes contained in the CCB by returning an error to the `saImmOiCcbValidateCallback()` function.
- A CCB validator process that is required to validate the CCB changes or a CCB applier process that is required to apply the CCB changes explicitly unregisters (or the process exits) before all CCB validators have approved the CCB changes.

The CCB is not aborted if the process that called the `saImmOmCcbApply_3()` function exits before this function completes. Once the CCB has been successfully validated by the current validators, it will not be aborted, even if a new validator registers while the CCB is being applied.

None of the handles used during the processing of the CCB should be finalized before the CCB is finalized. Finalizing a handle in such a situation is considered a usage error. However, it is guaranteed that either all changes contained in the CCB have been applied persistently to the SA Forum Information Model, or none of them have been applied.

4.8.1 saImmOmCcbInitialize_3()

Prototype

```
SaAisErrorT saImmOmCcbInitialize_3(
     SaImmAdminOwnerHandleT ownerHandle,
     SaImmCcbFlagsT_3 ccbFlags,
     SaImmCcbHandleT *ccbHandle,
     SaImmCcbIdT *ccbId
);
```

Parameters

`ownerHandle` - [in] Administrative owner handle. The `SaImmAdminOwnerHandleT` type is defined in Section 4.2.1 on page 33.

`ccbFlags` - [in] CCB flags. The `SaImmCcbFlagsT_3` type is defined in Section 4.2.14 on page 41.

`ccbHandle` - [out] Pointer to the CCB handle. The `SaImmCcbHandleT` type is defined in Section 4.2.1 on page 33.

`ccbId` - [out] Pointer to the CCB identifier. The `SaImmCcbIdT` type is defined in Section 4.2.15 on page 41.

Description

This function initializes a new CCB and returns both a handle (`ccbHandle`) and a CCB identifier for it (`ccbId`). The value of `ccbId` is global and unique within the IMM Service. The same value is supplied to CCB validators and CCB appliers in the corresponding callbacks, and it can be used in the functions to initialize an object search (see Section 4.5 on page 61) and to initialize an object access (see Section 4.6 on page 69).
The lifetime of the returned CCB identifier, `ccbId`, is tied to the lifetime of the returned `ccbHandle`, that is, this CCB identifier only ceases to exist when the `ccbHandle` is finalized.

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).

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `ownerHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmAdminOwnerInitialize()`

4.8.2 saImmOmCcbObjectCreate_2()

Prototype

```
SaAisErrorT saImmOmCcbObjectCreate_2(
     SaImmCcbHandleT ccbHandle,
     const SaImmClassNameT className,
     const SaNameT *parentName,
     const SaImmAttrValuesT_2 *const *attrValues
);
```

Parameters

`ccbHandle` - [in] CCB handle. The `SaImmCcbHandleT` type is defined in Section 4.2.1 on page 33.

`className` - [in] Object name class. The `SaImmClassNameT` type is defined in Section 4.2.2 on page 34.

`parentName` - [in] Pointer to the name of the parent of the new object. The `SaNameT` type is defined in [2].

`attrValues` - [in] Pointer to a NULL-terminated array of pointers to attribute descriptors. The `SaImmAttrValuesT_2` type is defined in Section 4.2.8 on page 37.

Description

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

The attributes specified by the array to which `attrValues` refers must match the object class definition. Only configuration and persistent runtime attributes can be specified by this array.

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

The creation will only be persistently performed when the CCB is applied.

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

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,
- the parent object referred to by the parentName parameter and some of its ancestors are non-persistent objects.
- all of the configuration attributes required at object creation are not provided by the caller,
- or the attrValues parameter includes:
  - non-persistent runtime attributes,
  - attributes with values that do not match the defined value type for the attribute, and
  - 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.

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

- The name to which the parentName parameter points 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.
- The SA_IMM_ATTR_VALIDATOR_NAME attribute of the parent object contains the name of a CCB validator that is currently not registered, or the object class referred to by the className parameter requires at least one CCB validator for which no CCB validator process is currently registered.
- The SA_IMM_ATTR_APPLIER_NAME attribute of the parent object contains the name of a CCB applier that is currently not registered, or the object class referred to by the className parameter requires at least one CCB applier for which no CCB applier process is currently registered.

SA_AIS_ERR_EXIST - An object with the same name already exists.

SA_AIS_ERR_FAILED_OPERATION - The operation failed because the CCB has been aborted. The CCB is now empty.

SA_AIS_ERR_NAME_TOO_LONG - The size of the new object's DN is greater than SA_MAX_NAME_LENGTH.

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

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle ccbHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOmCcbInitialize_3(), saImmOmCcbApply_3()

4.8.3 saImmOmCcbObjectDelete()

Prototype

```
SaAisErrorT saImmOmCcbObjectDelete(
     SaImmCcbHandleT ccbHandle,
     const SaNameT *objectName
);
```

Parameters

`ccbHandle` - [`in`] CCB handle. The `SaImmCcbHandleT` type is defined in Section 4.2.1 on page 33.

`objectName` - [`in`] Pointer to the object name. The `SaNameT` type is defined in [2].

Description

This function adds to the CCB identified by its handle `ccbHandle` a request to delete the configuration object designated by the name to which the `objectName` parameter points and the entire subtree of configuration objects rooted at that object.

This operation fails if one of the targeted objects is not a configuration object that is administratively owned by the administrative owner of the CCB. It also fails if one of the targeted objects has some registered continuation identifiers.

The deletion will only be persistently performed when the CCB is applied.

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.

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

- at least one of the targeted objects is not a configuration object that is owned by the administrative owner of the CCB;
- at least one of the targeted objects has some registered continuation identifiers;

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

- The name to which the objectName parameter points is not the name of an existing object.
- The SA_IMM_ATTR_VALIDATOR_NAME attribute of the object contains the name of a CCB validator that is currently not registered.
- The SA_IMM_ATTR_APPLIER_NAME attribute of the object contains the name of a CCB applier that is currently not registered.

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_ERR_FAILED_OPERATION - The operation failed because the CCB has been aborted. The CCB is now empty.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle ccbHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOmCcbInitialize_3(), saImmOmCcbApply_3()

4.8.4 saImmOmCcbObjectModify_2()

Prototype

```
SaAisErrorT saImmOmCcbObjectModify_2(
      SaImmCcbHandleT ccbHandle,
      const SaNameT *objectName,
      const SaImmAttrModificationT_2 *const *attrMods
);
```

Parameters

`ccbHandle` - [in] CCB handle. The `SaImmCcbHandleT` type is defined in Section 4.2.1 on page 33.

`objectName` - [in] Pointer to the name of the object to be modified. The `SaNameT` type is defined in [2].

`attrMods` - [in] Pointer to a NULL-terminated array of pointers to descriptors of the modifications to perform. The `SaImmAttrModificationT_2` type is defined in Section 4.2.10 on page 38.

Description

This function adds to the CCB identified by its handle `ccbHandle` a request to modify configuration attributes of a configuration 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 persistently performed when the CCB is applied.

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.

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 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.

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

- The name to which the objectName parameter points is not the name of an existing object.
- One or more attribute names specified by the attrMods parameter are not valid for the object class.
- The SA_IMM_ATTR_VALIDATOR_NAME attribute of the object contains the name of a CCB validator that is currently not registered.
- The SA_IMM_ATTR_APPLIER_NAME attribute of the object contains the name of a CCB applier that is currently not registered.

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

SA_AIS_ERR_FAILED_OPERATION - The operation failed because the CCB has been aborted. The CCB is now empty.

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

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle ccbHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOmCcbInitialize_3(), saImmOmCcbApply_3()

4.8.5 saImmOmCcbApply_3()

Prototype

```
SaAisErrorT saImmOmCcbApply_3(
     SaImmCcbHandleT ccbHandle,
     SaNtfCorrelationIdsT *correlationIds
);
```

Parameters

ccbHandle - [in] CCB handle. The SaImmCcbHandleT type is defined in Section 4.2.1 on page 33.

correlationIds - [in/out] Pointer to the correlation identifiers associated with the CCB being applied. The rootCorrelationId and parentCorrelationId fields are in parameters and hold the root and parent correlation identifiers, respectively. These correlation identifiers are included by the IMM Service in its notifications triggered by the invocation of this API. The notificationId field is an out parameter that holds the notification identifier of the notification that the IMM Service sends to report that the CCB has been applied. The SaNtfCorrelationIdsT type is defined in [6].

Description

This function initiates the persistent application of 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.

In the first step to apply the changes contained in the CCB persistently, the IMM Service invokes the `saImmOiCcbValidateCallback()` function of all CCB validators involved in the change requests contained in the CCB (see Section 5.5.1 on page 146) to validate the CCB (see Section 5.5 on page 144). If the validation succeeds, the IMM Service applies all changes in the CCB persistently to the SA Forum Information Model and then invokes the `saImmOiCcbApplyCallback()` functions of all the CCB appliers of all objects affected by the CCB (see Section 5.6.1 on page 151) to deploy the changes.

This operation fails if the administrative ownership of an object targeted by this CCB has changed since the change was added to the CCB, and the new administrative owner of the object is no longer the administrative owner of the CCB.

If this function returns with `SA_AIS_OK`, the CCB changes have been applied to the SA Forum Information Model and deployed by the CCB appliers; if it returns with `SA_AIS_ERR_DEPLOYMENT`, the CCB has been successfully applied to the SA Forum Information Model, it is, however, unspecified whether the CCB changes have been completely or only partially deployed or not deployed at all. For all other return values, the CCB has not been applied.

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. The CCB changes have been applied to the SA Forum Information Model and deployed by the CCB appliers.

`SA_AIS_ERR_DEPLOYMENT` - The CCB has been successfully applied to the SA Forum Information Model; however, it is unspecified whether the CCB changes have been completely or only partially deployed or not deployed at all. This value is also returned if `ccbHandle` is invalidated after the CCB changes have been applied to the SA Forum Information Model.

`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.

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_ERR_FAILED_OPERATION - The operation failed because the CCB has been aborted. The CCB is now empty.

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

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle ccbHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOmCcbInitialize_3(), saImmOmCcbObjectCreate_2(), saImmOmCcbObjectDelete(), saImmOmCcbObjectModify_2(), SaImmOiCcbValidateCallbackT, SaImmOiCcbApplyCallbackT_3

4.8.6 saImmOmCcbFinalize()

Prototype

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

Parameters

`ccbHandle` - [in] CCB handle. The `SaImmCcbHandleT` type is defined in Section 4.2.1 on page 33.

Description

This function finalizes the CCB identified by `ccbHandle`, which also implies that the CCB identifier associated with it (that is, the one returned in the corresponding `saImmOmCcbInitialize_3()` call) is no longer valid.

If the `saImmOmCcbFinalize()` function is called after changes have been added to the CCB and before the `saImmOmCcbApply_3()` function is invoked, the CCB is aborted, and all change requests contained in the CCB are removed. Invoking the `saImmOmCcbFinalize()` function while the CCB is being applied is considered an error. However, it is guaranteed that either all changes have been applied persistently to the SA Forum Information Model, or none of 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 `ccbHandle` is invalid, since it is corrupted, uninitialized, or has already been finalized.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle ccbHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOmCcbInitialize_3()

4.9 Administrative Operations Invocation

Processes can invoke **administrative operations** on IMM objects by using the `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()` API functions.

The IMM Service transfers the administrative operation to the registered runtime owner process by invoking its `saImmOiAdminOperationCallback()` registered callback, passing along all parameters provided to the `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()` API functions.

If the invoking process exits (due to a failure, for example) before the administrative operation completes, the IMM Service allows another process to carry over the invocation and wait for its result by invoking the `saImmOmAdminOperationContinue()` or `saImmOmAdminOperationContinueAsync()` API functions. These functions are called **continuation functions**. Saying in this section that a process **carries over** an administrative operation means that the process invokes the appropriate IMM Service API functions to wait for the result of an administrative operation that has been correctly initiated. Note that carrying over an administrative operation does not affect the outcome of the administrative operation.
The administrative operation may have completed when a continuation function is called. In this case, the continuation function will just fetch the result of the administrative operation that has been buffered by the IMM Service.
A runtime owner is not aware of the continuation functions, the support of which is entirely handled by the IMM Service.

In order for an administrative operation to be carried over, the original invoker of the administrative operation must provide a nonzero **continuation identifier**. The continuation identifier must be unique on a per-object basis. It is the responsibility of the process that initiates the administrative operation to store the continuation identifier in a location where a process that may need to carry over the operation can access it. The location where a continuation identifier is stored is not specified by the IMM Service and is application-specific; checkpoints or files may be used to store continuation identifiers.
The IMM Service registers a particular continuation identifier with an object when an administrative operation is invoked on the object by a call to `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()`. The continuation identifier will stay registered with the object until explicitly cleared with `saImmOmAdminOperationContinuationClear()`, or until the administrative ownership on the object that was in effect at the time of the invocation of

`saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()`is released.
As long as a continuation identifier stays registered with the object, it is said to be a **registered continuation identifier**, and the IMM Service shall keep any result of the associated administrative operation available.
Continuation identifiers are not persistent, and they are all cleared when the IMM Service is terminated.

The IMM Service does not allow concurrent continuation operations for the same continuation identifier. As a consequence, `saImmOmAdminOperationContinue()` and `saImmOmAdminOperationContinueAsync()` will fail and return an SA_AIS_ERR_EXIST error if

- the administrative owner handle that was used when the continuation identifier for an object was first provided in an invocation of `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()` is still valid, or if
- the administrative owner handle that was used when the continuation identifier for an object was last provided in an invocation of any of the two continuation functions is still valid.

4.9.1 saImmOmAdminOperationInvoke_3(), saImmOmAdminOperationInvokeAsync_3()

Prototype

```
SaAisErrorT saImmOmAdminOperationInvoke_3(
     SaImmAdminOwnerHandleT ownerHandle,
     const SaNameT *objectName,
     SaImmContinuationIdT continuationId,
     SaNtfCorrelationIdsT *correlationIds,
     SaImmAdminOperationIdT operationId,
     const SaImmAdminOperationParamsT_2 *const *params,
     SaAisErrorT *operationReturnValue,
     SaTimeT timeout
);
```

Parameters

`ownerHandle` - [in] Administrative owner handle.
The `SaImmAdminOwnerHandleT` type is defined in Section 4.2.1 on page 33.

`objectName` - [in] Pointer to the object name. The `SaNameT` type is defined in [2].

`continuationId` - [in] Continuation identifier for this particular invocation of the administrative operation. In case `ownerHandle` is finalized before the process retrieved the result of the operation, the result of the operation may be obtained by specifying another valid administrative owner handle in an invocation of one of the `saImmOmAdminOperationContinue()` or `saImmOmAdminOperationContinueAsync()` functions.
The `continuationId` parameter must be set to 0 if the invocation shall not be carried over at a later point of time. The `SaImmContinuationIdT` type is defined in Section 4.2.16 on page 41.

`correlationIds` - [in/out] Pointer to the correlation identifiers associated with the administrative operation. The `rootCorrelationId` and `parentCorrelationId` fields are `in` parameters and hold the root and parent correlation identifiers, respectively. These correlation identifiers are included by the IMM Service in its notifications triggered by the invocation of this API. The `notificationId` field is an `out` parameter that holds the notification identifier of the notification that the IMM Service sends to report the invocation of the administrative operation. The `SaNtfCorrelationIdsT` type is defined in [6].

`operationId` - [in] Identifier of the administrative operation.
The `SaImmAdminOperationIdT` type is defined in Section 4.2.17 on page 42.

`params` - [in] Pointer to a NULL-terminated array of pointers to parameter descriptors. The `SaImmAdminOperationParamsT_2` type is defined in Section 4.2.18 on page 42.

`operationReturnValue` - [out] Pointer to the value returned by the registered runtime owner for the invoked operation. This value is specific to the administrative operation being performed, and it is valid only if the `saImmOmAdminOperationInvoke_3()` function returns `SA_AIS_OK`. For more details about this value, refer to the description of the administrative operation in question in the documentation of the object implementer that is currently the registered runtime owner. The `SaAisErrorT` type is defined in [2].

`timeout` - [in] The `saImmOmAdminOperationInvoke_3()` invocation is considered to have failed if it does not complete by the time specified.
The `SaTimeT` type is defined in [2].

Prototype

```
SaAisErrorT saImmOmAdminOperationInvokeAsync_3(
      SaImmAdminOwnerHandleT ownerHandle,
      SaInvocationT invocation,
      const SaNameT *objectName,
      SaImmContinuationIdT continuationId,
      const SaNtfCorrelationIdsT *correlationIds,
      SaImmAdminOperationIdT operationId,
      const SaImmAdminOperationParamsT_2 *const *params
);
```

Parameters

`ownerHandle` - [in] Administrative owner handle.
The `SaImmAdminOwnerHandleT` type is defined in Section 4.2.1 on page 33.

`invocation` - [in] Used to match this invocation of `saImmOmAdminOperationInvokeAsync_3()` with the corresponding invocation of the `SaImmOmAdminOperationInvokeCallbackT_3` callback.
The `SaInvocationT` type is defined in [2].

`objectName` - [in] Pointer to the object name. The `SaNameT` type is defined in [2].

`continuationId` - [in] Continuation identifier for this particular invocation of the administrative operation. In case `ownerHandle` is finalized before the process retrieved the result of the operation, the result of the operation may be obtained by specifying another valid administrative owner handle in an invocation of one of the `saImmOmAdminOperationContinue()` or `saImmOmAdminOperationContinueAsync()` functions.
The `continuationId` parameter must be set to 0 if the operation shall not be carried over at a later point of time. The `SaImmContinuationIdT` type is defined in Section 4.2.16 on page 41.

`correlationIds` - [in] Pointer to the correlation identifiers associated with the administrative operation. The `rootCorrelationId` and `parentCorrelationId` fields are `in` parameters and hold the root and parent correlation identifiers, respectively. These correlation identifiers are included by the IMM Service in its notifications triggered by the invocation of this API. The `notificationId` field is not used. The `SaNtfCorrelationIdsT` type is defined in [6].

`operationId` - [in] Identifier of the administrative operation.
The `SaImmAdminOperationIdT` type is defined in Section 4.2.17 on page 42.

`params` - [in] Pointer to a NULL-terminated array of pointers to parameter descriptors. The `SaImmAdminOperationParamsT_2` type is defined in Section 4.2.18 on page 42.

Description

Using the IMM Service as an intermediary, these two functions request the registered runtime owner of the object designated by the name to which `objectName` points to perform an administrative operation characterized by `operationId` on that object. Administrative operations can be performed on configuration and runtime objects.

Each descriptor pointed to by an element of the array of pointers to which the `params` parameter points represents an input parameter of the administrative operation to execute.

The function `saImmOmAdminOperationInvoke_3()` is the synchronous variant and returns only when the registered runtime owner has successfully completed the execution of the administrative operation, or when an error has been detected by the IMM Service or the registered runtime owner.

The function `saImmOmAdminOperationInvokeAsync_3()` is the asynchronous variant; it returns as soon as the IMM Service has registered the request to be transmitted to the registered runtime owner. If the IMM Service detects an error while registering the request, an error is immediately returned, and no further invocation of the `saImmOmAdminOperationInvokeCallback()` callback function of the registered runtime owner must be expected for this invocation of `saImmOmAdminOperationInvokeAsync_3()`. If no error is detected by the IMM Service while registering the request, the invocation of `saImmOmAdminOperationInvokeAsync_3()` completes successfully, and a later invocation of the `saImmOmAdminOperationInvokeCallback()` callback function will occur to indicate the success or failure of the administrative operation on the target object.

If the administrative owner handle `ownerHandle` becomes finalized before the process could retrieve the result of the administrative operation (returned by `saImmOmAdminOperationInvoke_3()` or passed to the `saImmOmAdminOperationInvokeCallback()` callback function of the process), the current process or another process may invoke one of the functions `saImmOmAdminOperationContinue()` or `saImmOmAdminOperationContinueAsync()` on a valid administrative owner handle to carry over the operation to retrieve its results, if necessary.

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 corresponding previous invocation of saImmOmInitialize_3() to initialize the IMM Service and obtain the IMM Service handle (with which the handle ownerHandle was obtained by invoking saImmOmAdminOwnerInitialize()) was incomplete, since the SaImmOmAdminOperationInvokeCallbackT_3 callback function was missing. This return value applies only to the saImmOmAdminOperationInvokeAsync_3() 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 the name to which objectName points is not owned by the administrative owner associated with ownerHandle.

SA_AIS_ERR_NOT_EXIST - The name to which the objectName parameter points is not the name of an existing object, or there is no registered runtime owner for this object.

SA_AIS_ERR_EXIST - The object designated by the name to which objectName points has already a registered continuation identifier identical to continuationId.

SA_AIS_ERR_BUSY - The object designated by the name to which objectName points is already the target of a change request in a CCB.

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

`SA_AIS_ERR_FAILED_OPERATION` - The operation failed due to a problem with the registered runtime owner.

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `ownerHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmAdminOwnerInitialize()`, `SaImmOmAdminOperationInvokeCallbackT_3`, `saImmOmAdminOperationContinue()`, `saImmOmAdminOperationContinueAsync()`, `saImmOmAdminOperationContinueClear()`

4.9.2 SaImmOmAdminOperationInvokeCallbackT_3

Prototype

```
typedef void (*SaImmOmAdminOperationInvokeCallbackT_3) (
     SaInvocationT invocation,
     SaNtfIdentifierT notificationId,
     SaAisErrorT operationReturnValue,
     SaAisErrorT error
);
```

Parameters

`invocation` - [in] Used to match this callback invocation to the corresponding previous invocation of either `saImmOmAdminOperationInvokeAsync_3()` or `saImmOmAdminOperationContinueAsync()`, depending on which of these functions was called last. The `SaInvocationT` type is defined in [2].

`notificationId` - [in] Holds the notification identifier of the notification that the IMM Service sends to report the invocation of the administrative operation. The `SaNtfIdentifierT` type is defined in [6].

`operationReturnValue` - [in] Value returned by the registered runtime owner for the administrative operation requested in the corresponding previous invocation of either `saImmOmAdminOperationInvokeAsync_3()` or `saImmOmAdminOperationContinueAsync()`, depending on which of these functions was called last.
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.
The `SaAisErrorT` type is defined in [2].

`error` - [in] Indicates whether the IMM Service succeeded or not to invoke the registered runtime owner.
The `SaAisErrorT` type is defined in [2].
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 either `saImmOmAdminOperationInvokeAsync_3()` or `saImmOmAdminOperationContinueAsync()` (depending on which of these functions was called last) 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_BAD_OPERATION` - The object designated by the name to which the `objectName` parameter points in the corresponding invocation of either `saImmOmAdminOperationInvokeAsync_3()` or `saImmOmAdminOperationContinueAsync()` (depending on which of these functions was called last) is not owned by the administrative owner associated with `ownerHandle`.

- `SA_AIS_ERR_NOT_EXIST` - The name to which the `objectName` parameter points in the corresponding invocation of either `saImmOmAdminOperationInvokeAsync_3()` or `saImmOmAdminOperationContinueAsync()` (depending on which of these functions was called last) is not the name of an existing object, or there is no registered runtime owner for this object.
- `SA_AIS_ERR_EXIST` - Two cases must be distinguished:
  - This callback has been requested by the `saImmOmAdminOperationInvokeAsync_3()` call: the object designated by the name to which the `objectName` parameter points in the `saImmOmAdminOperationInvokeAsync_3()` call has already a registered continuation identifier identical to `continuationId`.
  - This callback has been requested by the `saImmOmAdminOperationContinueAsync()` call: the object designated by the name to which the `objectName` parameter points in the `saImmOmAdminOperationContinueAsync()` call has already a registered continuation identifier identical to `continuationId`, and the administrative owner handle specified for this object in a preceding call to one of the following functions (depending on which of these four functions was called last) has not yet been finalized:
    - either `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()`, or
    - either `saImmOmAdminOperationContinue()` or `saImmOmAdminOperationContinueAsync()`
- `SA_AIS_ERR_BUSY` - The object designated by the name to which the `objectName` parameter points in the corresponding invocation of either `saImmOmAdminOperationInvokeAsync_3()` or `saImmOmAdminOperationContinueAsync()` (depending on which of these functions was called last) is already the target of a change request in a CCB.
- `SA_AIS_ERR_FAILED_OPERATION` - The operation failed due to a problem with the registered runtime owner.
- `SA_AIS_ERR_VERSION` - The invoked function is not supported in the version specified in the call to initialize this instance of the IMM Service library. This return code applies only if this call was triggered by a previous invocation of either `saImmOmAdminOperationInvokeAsync_23()` or `saImmOmAdminOperationContinueAsync()`.
- `SA_AIS_ERR_UNAVAILABLE` - The operation requested by either the corresponding `saImmOmAdminOperationContinueAsync()` call or the corresponding `saImmOmAdminOperationInvokeAsync_3()` call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `ownerHandle` specified in either the corresponding `saImmOmAdminOperationContinueAsync()` call or the corresponding `saImmOmAdminOperationInvokeAsync_3()` call was acquired before the cluster node left the cluster membership.

Description

The IMM Service invokes this callback function when the operation requested by the corresponding invocation of either `saImmOmAdminOperationInvokeAsync_3()` or `saImmOmAdminOperationContinueAsync()` (depending on which of these functions was called last) completes successfully, or an error is detected.

This callback is invoked in the context of a thread calling `saImmOmDispatch()` with the handle `immHandle` that was used to initialize the `ownerHandle` specified in one of the corresponding functions `saImmOmAdminOperationInvokeAsync_3()` or `saImmOmAdminOperationContinueAsync()`, depending on which of these functions was called last.

Return Values

None

See Also

`saImmOmAdminOwnerInitialize()`, `saImmOmDispatch()`, `saImmOmAdminOperationInvokeAsync_3()`, `saImmOmAdminOperationContinue()`, `saImmOmAdminOperationContinueAsync()`, `saImmOmAdminOperationContinueClear()`

4.9.3 saImmOmAdminOperationContinue(), saImmOmAdminOperationContinueAsync()

Prototype

```
SaAisErrorT saImmOmAdminOperationContinue(
     SaImmAdminOwnerHandleT ownerHandle,
     const SaNameT *objectName,
     SaImmContinuationIdT continuationId,
     SaAisErrorT *operationReturnValue
);
```

Parameters

`ownerHandle` - [in] Administrative owner handle.
The `SaImmAdminOwnerHandleT` type is defined in Section 4.2.1 on page 33.

`objectName` - [in] Pointer to the object name. The `SaNameT` type is defined in [2].

`continuationId` - [in] Identifies the corresponding previous invocation of `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()`.
The `SaImmContinuationIdT` type is defined in Section 4.2.16 on page 41.

`operationReturnValue` - [out] Pointer to the value returned by the registered runtime owner for the operation requested by the corresponding previous call to `saImmOmAdminOperationInvoke_3()` or to `saImmOmAdminOperationInvokeAsync_3()`. The value returned by the registered runtime owner is specific to the administrative operation being performed, and it is valid only if the `saImmOmAdminOperationContinue()` function returns `SA_AIS_OK`. For more details about this value, refer to the object implementer administrative operation description. The `SaAisErrorT` type is defined in [2].

Prototype

```
SaAisErrorT saImmOmAdminOperationContinueAsync(
      SaImmAdminOwnerHandleT ownerHandle,
      SaInvocationT invocation,
      const SaNameT *objectName,
      SaImmContinuationIdT continuationId
);
```

Parameters

`ownerHandle` - [in] Administrative owner handle.
The `SaImmAdminOwnerHandleT` type is defined in Section 4.2.1 on page 33.

`invocation` - [in] Used to match this invocation of `saImmOmAdminOperationContinueAsync()` with the corresponding invocation of the `SaImmOmAdminOperationInvokeCallbackT_3` callback.
The `SaInvocationT` type is defined in [2].

`objectName` - [in] Pointer to the object name. The `SaNameT` type is defined in [2].

`continuationId` - [in] Identifies the corresponding previous invocation of `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()`.
The `SaImmContinuationIdT` type is defined in Section 4.2.16 on page 41.

Description

These two functions allow a process to carry over the invocation of an administrative operation that had been initiated with a particular administrative handle but did not complete before the handle was finalized (explicitly or as a side effect of the process termination).

The process carrying over the operation may invoke a synchronous or asynchronous continuation function regardless of whether the respective administrative operation was initiated by invoking `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()`.

The function `saImmOmAdminOperationContinue()` is the synchronous variant and returns only when the registered runtime owner has successfully completed the execution of the administrative operation, or when an error has been detected by the IMM Service or the registered runtime owner.

The function `saImmOmAdminOperationContinueAsync()` is the asynchronous variant; it returns as soon as the IMM Service has registered the request. If the IMM Service detects an error while registering the request, an error is immediately returned, and no further invocation of the `SaImmOmAdminOperationInvokeCallbackT_3` callback must be expected for this invocation of `saImmOmAdminOperationContinueAsync()`. If no error is detected by the IMM Service while registering the request, the invocation of `saImmOmAdminOperationInvokeAsync_3()` completes successfully, and the `SaImmOmAdminOperationInvokeCallbackT_3` callback will be invoked later to indicate the success or failure of the administrative operation on the target object.

The object name pointed to by `objectName` and the continuation identifier `continuationId` must be the same that were supplied in a corresponding previous invocation of `saImmOmAdminOperationInvoke_3()`, `saImmOmAdminOperationInvokeAsync_3()`.

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_INIT` - The corresponding previous invocation of `saImmOmInitialize_3()` to initialize the IMM Service and obtain the IMM Service handle (with which the handle `ownerHandle` was obtained by invoking `saImmOmAdminOwnerInitialize()`) was incomplete, since the `SaImmOmAdminOperationInvokeCallbackT_3` callback function was missing. This return value only applies to the `saImmOmAdminOperationContinueAsync()` 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 the name to which the `objectName` parameter points is not owned by the administrative owner associated with `ownerHandle`.

`SA_AIS_ERR_NOT_EXIST` - This error is returned if one of the following conditions apply:

- The name to which the `objectName` parameter points is not the name of an existing object, or there is no registered runtime owner for this object.
- The `continuationId` parameter is not a valid continuation identifier (that is, it is not a registered continuation identifier) for the object whose name is pointed to by the `objectName` parameter.

`SA_AIS_ERR_EXIST` - The object designated by the name to which the `objectName` parameter points has already a registered continuation identifier identical to `continuationId`, and the administrative owner handle specified for this object in the last call to one of the following functions (depending on which of these four functions was called last) has not yet been finalized:

- either `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()`, or
- either `saImmOmAdminOperationContinue()` or `saImmOmAdminOperationContinueAsync()`

`SA_AIS_ERR_BUSY` - The object designated by the name to which the `objectName` parameter points is already the target of a change request in a CCB.

`SA_AIS_ERR_FAILED_OPERATION` - The operation failed due to a problem with the registered runtime owner.

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `ownerHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmAdminOwnerInitialize()`, `saImmOmAdminOperationInvoke_3()`, `saImmOmAdminOperationInvokeAsync_3()`, `SaImmOmAdminOperationInvokeCallbackT_3`, `saImmOmAdminOperationContinueClear()`

4.9.4 saImmOmAdminOperationContinuationClear()

Prototype

```
SaAisErrorT saImmOmAdminOperationContinuationClear(
        SaImmAdminOwnerHandleT ownerHandle,
        const SaNameT *objectName,
        SaImmContinuationIdT continuationId
);
```

Parameters

`ownerHandle` - [in] Administrative owner handle.
The `SaImmAdminOwnerHandleT` type is defined in Section 4.2.1 on page 33.

`objectName` - [in] Pointer to the object name. The `SaNameT` type is defined in [2].

`continuationId` - [in] The continuation identifier that was supplied in the corresponding previous invocation of `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()`.
The `SaImmContinuationIdT` type is defined in Section 4.2.16 on page 41.

Parameters

Description

This function instructs the IMM Service to clear all information kept to allow the continuation of the administrative operation identified by `continuationId` for the object whose name is pointed to by `objectName` and the administrative owner identified by `ownerHandle`. After successful completion of this function, the `continuationId` identifier is cleared, that is, it is no longer a registered continuation identifier.

The object name pointed to by `objectName` and the continuation identifier `continuationId` must be the same that were supplied in the corresponding previous invocation of `saImmOmAdminOperationInvoke_3()`, `saImmOmAdminOperationInvokeAsync_3()`.

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_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 the name to which the `objectName` parameter points is not owned by the administrative owner associated with `ownerHandle`.

`SA_AIS_ERR_NOT_EXIST` - This error is returned if one of the following conditions apply:

- The name to which the `objectName` parameter points is not the name of an existing object, or there is no registered runtime owner for this object.
- The `continuationId` parameter is not a valid continuation identifier (that is, it is not a registered continuation identifier) for the object whose name is pointed to by the `objectName` parameter.

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `ownerHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOmAdminOperationInvoke_3()`, `saImmOmAdminOperationInvokeAsync_3()`, `SaImmOmAdminOperationInvokeCallbackT_3`, `saImmOmAdminOperationContinue()`, `saImmOmAdminOperationContinueAsync()`

5 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 Handles

The following handles are used by IMM Service object implementer API functions:

```
typedef SaUint64T SaImmOiHandleT;
typedef SaUint64T SaImmOiCcbIteratorHandleT;
```

5.2.2 SaImmOiImplementerNameT

The `SaImmOiImplementerNameT` type 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 SaImmOiRoleFlagsT

The `SaImmOiRoleFlagsT` type is used to specify the role or roles of an object implementer. A value of this type is one of the flags described further down or a logical *or* of the SA_IMM_ROLE_CCB_VALIDATOR and SA_IMM_ROLE_CCB_APPLIER flags.

```
#define SA_IMM_ROLE_RUNTIME_OWNER 0x00000001
#define SA_IMM_ROLE_CCB_VALIDATOR 0x00000002
#define SA_IMM_ROLE_CCB_APPLIER   0x00000004

typedef SaUint32T SaImmOiRoleFlagsT;
```

The meaning of the flags listed above is:

- `SA_IMM_ROLE_RUNTIME_OWNER` - if this flag is specified in a call to `saImmOiObjectImplementerSet_3()` or `saImmOiClassImplementerSet_3()`, the implementer name associated with the specified object implementer handle will become the name of the runtime owner of the corresponding objects.
- `SA_IMM_ROLE_CCB_VALIDATOR` - if this flag is specified in a call to `saImmOiObjectImplementerSet_3()` or `saImmOiClassImplementerSet_3()`, the implementer name associated with the specified object implementer handle will become the name of a CCB validator of the corresponding objects.
- `SA_IMM_ROLE_CCB_APPLIER` - if this flag is specified in a call to `saImmOiObjectImplementerSet_3()` or `saImmOiClassImplementerSet_3()`, the implementer name associated with the specified object implementer handle will become the name of a CCB applier of the corresponding objects.

5.2.4 SaImmOiCcbIteratorOptionT

The `SaImmOiCcbIteratorOptionT` type is used to specify the information that should be returned for the objects in the CCB.

```
#define SA_IMM_CCB_ITERATOR_REGISTERED_OBJECTS        0x01
#define SA_IMM_CCB_ITERATOR_OBJECT_NAME_ONLY          0x02
#define SA_IMM_CCB_ITERATOR_MODIFIED_ATTRIBUTES       0x04
typedef SaUint64T SaImmOiCcbIteratorOptionT;
```

The values of this type definition have the following meaning:

`SA_IMM_CCB_ITERATOR_REGISTERED_OBJECTS` - only return objects for which the invoking process is a registered CCB validator or CCB applier. If `SA_IMM_CCB_ITERATOR_REGISTERED_OBJECTS` is not set, all objects in the CCB are returned.

The two options `SA_IMM_CCB_ITERATOR_OBJECT_NAME_ONLY` and `SA_IMM_CCB_ITERATOR_MODIFIED_ATTRIBUTES` are mutually exclusive (only one of them can be specified). If none of these two options is specified, the object name and all its attributes are returned for each object.

5.2.5 SaImmOiObjectChangeT

The `SaImmOiObjectChangeT` type is used to specify how a returned object has been changed.

```
typedef enum {
      SA_IMM_OBJECT_CREATE  = 1,
      SA_IMM_OBJECT_DELETE  = 2,
      SA_IMM_OBJECT_MODIFY  = 3
} SaImmOiObjectChangeT;
```

The values of this type definition have the following meaning:

- `SA_IMM_OBJECT_CREATE` - indicates that the object has been created.
- `SA_IMM_OBJECT_DELETE` - indicates that the object has been deleted.
- `SA_IMM_OBJECT_MODIFIED` - indicates that some attribute of the object has been modified (add, delete, or replace).

5.2.6 SaImmOiObjectTraverseT

The `SaImmOiObjectTraverseT` is used to indicate how the SA Forum Information Model should be searched for changes.

```
typedef enum {
      SA_IMM_OBJECT_ENTER        = 1,
      SA_IMM_OBJECT_CONTINUE     = 2
} SaImmOiObjectTraverseT;
```

The values of this type definition have the following meaning:

- `SA_IMM_OBJECT_ENTER` - this value is used to request the iterator to search the subtree below the last returned object.

- SA_IMM_OBJECT_CONTINUE - this value is used to request the iterator not to enter the subtree below the last returned object.

5.2.7 SaImmOiCallbacksT_3

The SaImmOiCallbacksT_3 structure defines the set of callbacks a process implementing IMM Service objects can provide to the IMM Service at initialization time.

```
typedef struct {
      SaImmOiAdminOperationCallbackT_3
            saImmOiAdminOperationCallback;
      SaImmOiCcbAbortCallbackT_3
            saImmOiCcbAbortCallback;
      SaImmOiCcbApplyCallbackT_3
            saImmOiCcbApplyCallback;
      SaImmOiCcbValidateCallbackT
            saImmOiCcbValidateCallback;
      SaImmOiCcbFinalizeCallbackT
            saImmOiCcbFinalizeCallback;
      SaImmOiRtAttrUpdateCallbackT
            saImmOiRtAttrUpdateCallback;
} SaImmOiCallbacksT_3;
```

5.3 Library Life Cycle

5.3.1 saImmOiInitialize_3()

Prototype

```
SaAisErrorT saImmOiInitialize_3(
        SaImmOiHandleT *immOiHandle,
        const SaImmOiCallbacksT_3 *immOiCallbacks,
        SaVersionT *version
);
```

Parameters

`immOiHandle` - [`out`] A pointer to the handle which identifies this particular initialization of the IMM Service and which is to be returned by the IMM Service. This handle provides access to the object implementer APIs of the IMM Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`immOiCallbacks` - [`in`] If `immOiCallbacks` is set to NULL, no callback is registered; if `immOiCallbacks` is not set to NULL, it is pointer to an `SaImmOiCallbacksT_3` structure which contains 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_3` type is defined in Section 5.2.7 on page 120.

`version` - [`in/out`] As an input parameter, `version` is a pointer to a structure containing the required Information Model Management Service version. In this case, `minorVersion` is ignored and should be set to 0x00.
As an output parameter, `version` is a pointer to a structure containing the version actually supported by the Information Model Management Service. The `SaVersionT` type is defined in [2].

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 IMM Service object implementer functionality. The handle pointed to by `immOiHandle` is returned by the IMM Service as the reference to this association between the process and the IMM Service. The process uses this handle in subsequent communication with the IMM 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_3()` function and before invoking `saImmOiFinalize()` to finalize the handle `immOiHandle` (see Section 5.3.4 on page 126), the IMM Service automatically finalizes this handle when the death of the process is detected.

If the implementation supports the version of the IMM Service object implementer API specified by the `releaseCode` and `majorVersion` fields of the structure pointed to by the `version` parameter, `SA_AIS_OK` is returned. In this case, the structure pointed to by the `version` parameter is set by this function to:

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

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

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

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

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

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

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

See Also

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

5.3.2 saImmOiSelectionObjectGet()

Prototype

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

Parameters

immOiHandle - [in] The handle which was obtained by a previous invocation of the saImmOiInitialize_3() function and which identifies this particular initialization of the Information Model Management Service. The SaImmOiHandleT type is defined in Section 5.2.1 on page 117.

selectionObject - [out] A pointer to the operating system handle that the invoking process can use to detect pending callbacks. The SaSelectionObjectT type is defined in [2].

Description

This function returns the operating system handle associated with the handle immOiHandle. The invoking process can use the operating system 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 operating system handle returned by saImmOiSelectionObjectGet() is valid until saImmOiFinalize() is successfully 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).

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle immOiHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOiInitialize_3(), saImmOiDispatch(), saImmOiFinalize()

5.3.3 saImmOiDispatch()

Prototype

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

Parameters

immOiHandle - [in] The handle which was obtained by a previous invocation of the saImmOiInitialize_3() function and which identifies this particular initialization of the Information Model Management Service. The SaImmOiHandleT type is defined in Section 5.2.1 on page 117.

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. These flags are values of the SaDispatchFlagsT enumeration type, which is described in [2].

Description

In the context of the calling thread, this function invokes 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. This value is also returned if this function is being invoked with dispatchFlags set to SA_DISPATCH_ALL or SA_DISPATCH_BLOCKING, and the handle immOiHandle has been finalized.

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

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

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

`SA_AIS_ERR_BAD_HANDLE` - The handle `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_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immOiHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOiInitialize_3()`, `saImmOiSelectionObjectGet()`, `saImmOiFinalize()`

5.3.4 saImmOiFinalize()

Prototype

```
SaAisErrorT saImmOiFinalize(
     SaImmOiHandleT immOiHandle
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

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_3()` before it invokes this function. A process must invoke this function once for each handle it acquired by invoking `saImmOiInitialize_3()`.

This function does not release the associations established between object classes or objects and the implementer name that 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.

If the `saImmOiFinalize()` function completes successfully, it releases all resources acquired when `saImmOiInitialize_3()` 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.

If a process terminates, the Information Model Management Service implicitly finalizes all associations (handles) with the Information Model Management Service that were initialized by the process, as described in the preceding paragraph.

After `saImmOiFinalize()` returns successfully, the handle `immOiHandle` and the selection object associated with it are no longer valid.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

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

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

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

See Also

`saImmOiInitialize_3()`

5.4 Object Implementer API

With each object in the SA Forum Information Model, the IMM Service associates a set of processes that perform the different tasks required for the correct deployment, administration, and status update of this object, that is, for implementing it. The associations are created through names (termed **implementer names**) that are set for the object. A process may then register for a name and become an implementer of the objects for which this name is set. The different tasks of implementing an object are grouped into roles:

(1) **runtime owner**: A process having this role can create, delete, and modify runtime objects. Such a process is termed the runtime owner of the particular object. As a runtime object is only created by its runtime owner, the IMM Service can automatically set the implementer name for the object and for this role when the object is created. The runtime owner of a configuration object is responsible for maintaining the runtime attributes of the object and for carrying out the administrative operations issued on the object. Such a process indicates this role with respect to the object by registering with the IMM Service as a runtime owner for the object. At any given time, only a single process in the entire cluster can assume this role for an object regardless of whether the object is a configuration or runtime object. The name of the runtime owner is contained in the `SA_IMM_ATTR_RUNTIME_OWNER_NAME` attribute of the object.

(2) **CCB applier**: A process which interprets a configuration object and deploys any part of the configuration that the object reflects in the SAF system is termed a CCB applier of that configuration object. Such a process indicates this role with respect to the object by registering with the IMM Service as a CCB applier for the object. As a result, it is informed of any configuration modifications applied to this object. At any given time, one or more processes in the cluster can assume this role for an object. CCB appliers 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 list of CCB appliers is contained in the `SA_IMM_ATTR_APPLIER_NAME` attribute of the object.

(3) **CCB validator**: A process which validates any proposed configuration change for a configuration object is termed a CCB validator of the particular object. At any given time, one or more processes in the entire cluster can assume this role for an object. Whenever a CCB is to be applied that impacts this configuration object, all CCB validator processes associated with the object are requested by the IMM Service to validate the proposed changes. CCB validators 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 list of CCB validators is contained in the `SA_IMM_ATTR_VALIDATOR_NAME` attribute of the object.

Any process having these roles is termed an **object implementer**. In a few occurrences in this document, the term “object implementer” is also used to refer to any process having successfully obtained an object implementer handle, even if the process has not yet a role associated to it. The context clarifies the intended meaning.

The IMM Service keeps records of all object implementers and their associated roles with respect to every object.

A runtime object has only a runtime owner, and it is always the object implementer that created the object. For configuration objects, the implementer name for a certain role is set, and it remains associated with the object until explicitly released. The description refers to this association by saying that the object “has an implementer” or, for the particular roles, “has a runtime owner”, “has a CCB validator”, or “has a CCB applier”. This association and these terms apply even if the process that held the corresponding role for this implementer name (called the **registered object implementer**) clears the implementer name associated with its object implementer handle.
This feature 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.
The process of registration for an object implementer role, its release, and impact on in-progress CCBs is specified in the related functions and in
Section 5.5 on page 144.

Similar to the term registered object implementer, the more specific terms **registered runtime owner**, **registered CCB validator,** and **registered CCB applier** are also used to refer to these processes.

An object implementer handle can be associated with only one implementer name.

Typical use cases consist of enforcing system-wide constraints. For an illustration, see the example in Appendix A.
For sequence diagrams that illustrate the detailed sequence of API operations for configuration changes using CCBs, refer to Appendix B.

5.4.1 saImmOiImplementerSet()

Prototype

```
SaAisErrorT saImmOiImplementerSet(
      SaImmOiHandleT immOiHandle,
      const SaImmOiImplementerNameT implementerName
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`implementerName` - [in] Name of the object implementer. The `SaImmOiImplementerNameT` type is defined in Section 5.2.2 on page 117.

Description

This function sets the implementer name specified in the `implementerName` parameter for the handle `immOiHandle`. This function also registers the invoking process as an object implementer having the name which 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 that have an implementer name identical to `implementerName`.
Also, at any given time one `immOiHandle` handle can be associated with at most one object implementer name. If a process wants to register for multiple implementer names, it must obtain a separate `immOiHandle` handle for each of them by repeated initialization of the object implementer library.

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.

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_INIT - The corresponding previous invocation of saImmOiInitialize_3() to initialize the IMM Service object implementer library and obtain the handle specified by the immOiHandle parameter was incomplete, since one or more of the following callback functions, depending on the role currently associated with the object implementer name implementerName, were missing:

- implementerName parameter contains a name which is currently associated with the runtime owner role: SaImmOiRtAttrUpdateCallbackT and SaImmOiAdminOperationCallbackT_3;
- implementerName parameter contains a name which is currently associated with the CCB applier role: SaImmOiCcbApplyCallbackT_3;
- implementerName parameter contains a name which is currently associated with the CCB validator role: SaImmOiCcbValidateCallbackT, SaImmOiCcbFinalizeCallbackT, and SaImmOiCcbAbortCallbackT_3.

SA_AIS_ERR_EXIST - An object implementer with the same name is already registered with the IMM Service or an object implementer name is already set for the handle immOiHandle.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle immOiHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOiInitialize_3(), saImmOiImplementerClear(), SaImmOiRtAttrUpdateCallbackT, SaImmOiAdminOperationCallbackT_3, SaImmOiCcbApplyCallbackT_3, SaImmOiCcbValidateCallbackT, SaImmOiCcbFinalizeCallbackT, SaImmOiCcbAbortCallbackT_3

5.4.2 saImmOiImplementerClear()

Prototype

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

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

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 that 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, has already been finalized, or it is not associated with an implementer name.

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immOiHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOiInitialize_3()`, `saImmOiImplementerSet()`, `saImmOiSelectionObjectGet()`, `saImmOiDispatch()`, `saImmOiFinalize()`, `SaImmOiRtAttrUpdateCallbackT`, `SaImmOiAdminOperationCallbackT_3`, `SaImmOiCcbApplyCallbackT_3`, `SaImmOiCcbValidateCallbackT`, `SaImmOiCcbFinalizeCallbackT`, `SaImmOiCcbAbortCallbackT_3`

5.4.3 saImmOiClassImplementerSet_3()

Prototype

```
SaAisErrorT saImmOiClassImplementerSet_3(
     SaImmOiHandleT immOiHandle,
     SaImmOiRoleFlagsT role,
     const SaImmClassNameT className
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`role` - [in] Role of this object implementer. The `SaImmOiRoleFlagsT` type is defined in Section 5.2.3 on page 117.

`className` - [in] Object class name. The `SaImmClassNameT` type is defined in Section 4.2.2 on page 34.

Description

An object implementer whose name is associated with the handle `immOiHandle` invokes this function to inform the IMM Service that the object implementer assumes the role or roles specified by the `role` parameter for all the objects that are instances of the object class whose name is specified by the `className` parameter (existing objects as well as objects that will be created in the future).

If this operation succeeds and the `role` parameter contains the `SA_IMM_ROLE_RUNTIME_OWNER` flag, the current process becomes the runtime owner of all objects of the object class whose name is specified by `className`; additionally, for each targeted object, the IMM Service adds to the object's `SA_IMM_ATTR_RUNTIME_OWNER_NAME` attribute the implementer name (if not already present) associated with the handle `immOiHandle`.
This operation fails if the object class whose name is specified by the `className` parameter or an object of this class has already a runtime owner whose name is different from the implementer name associated with the handle `immOiHandle`.

If this operation succeeds and the `role` parameter contains the `SA_IMM_ROLE_CCB_APPLIER` flag, the current process becomes a CCB applier of all objects of the object class whose name is specified by `className`; additionally, for each targeted object, the IMM Service adds to the object's `SA_IMM_ATTR_APPLIER_NAME` attribute the implementer name (if not yet present) associated with the handle `immOiHandle`.

If this operation succeeds and the `role` parameter contains the `SA_IMM_ROLE_CCB_VALIDATOR` flag, the current process becomes a CCB validator of all objects of the object class whose name is specified by `className`; additionally, for each targeted object, the IMM Service adds to the object's `SA_IMM_ATTR_VALIDATOR_NAME` attribute the implementer name (if not yet present) 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_INIT - The corresponding previous invocation of saImmOiInitialize_3() to initialize the IMM Service object implementer library and obtain the handle specified by the immOiHandle parameter was incomplete, since one or more of the following callback functions, depending on the role parameter, were missing:

- role parameter contains the SA_IMM_ROLE_RUNTIME_OWNER flag: SaImmOiRtAttrUpdateCallbackT and SaImmOiAdminOperationCallbackT_3;
- role parameter contains the SA_IMM_ROLE_CCB_APPLIER flag: SaImmOiCcbApplyCallbackT_3;
- role parameter contains the SA_IMM_ROLE_CCB_VALIDATOR flag: SaImmOiCcbValidateCallback, SaImmOiCcbFinalizeCallbackT, and SaImmOiCcbAbortCallbackT_3.

SA_AIS_ERR_BAD_OPERATION - The className parameter specifies the name of a runtime object class.

SA_AIS_ERR_NOT_EXIST - The className parameter does not specify the name of an existing class.

SA_AIS_ERR_EXIST - The object class whose name is specified by the className parameter or an object of that class has already a runtime owner whose name is different from the implementer name associated with the handle immOiHandle.

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

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle immOiHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOiInitialize_3(), SaImmOiRtAttrUpdateCallbackT, SaImmOiAdminOperationCallbackT_3, SaImmOiCcbApplyCallbackT_3, SaImmOiCcbValidateCallbackT, SaImmOiCcbFinalizeCallbackT, SaImmOiCcbAbortCallbackT_3

5.4.4 saImmOiClassImplementerRelease_3()

Prototype

```
SaAisErrorT saImmOiClassImplementerRelease_3(
        SaImmOiHandleT immOiHandle,
        SaImmOiRoleFlagsT role,
        const SaImmClassNameT className
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`role` - [in] Role of this object implementer. The `SaImmOiRoleFlagsT` type is defined in Section 5.2.3 on page 117.

`className` - [in] Object class name. The `SaImmClassNameT` type is defined in Section 4.2.2 on page 34.

Description

An object implementer whose name is associated with the handle `immOiHandle` invokes this function to inform the IMM Service that the object implementer no longer holds the role or roles specified by the `role` parameter for all the objects that are instances of the object class whose name is specified by the `className` parameter (existing objects as well as objects that will be created in the future).

An object implementer can release just some of the roles for which it has registered. It may, for example, register as both a CCB validator and a CCB applier, and then, at a later time, unregister as a CCB applier. It will then remain a CCB validator.

If this operation succeeds, and the CCB validator, CCB applier, or runtime owner role is to be released, the IMM Service removes the implementer name from the attribute `SA_IMM_ATTR_VALIDATOR_NAME`, `SA_IMM_ATTR_APPLIER_NAME`, and `SA_IMM_ATTR_RUNTIME_OWNER_NAME`, respectively. If no more implementer names remain in any of these attribute, the attribute is removed.

If this operation succeeds, and the `role` parameter contains the `SA_IMM_ROLE_RUNTIME_OWNER` flag, the IMM Service removes all non-persistent cached runtime attributes from all objects of that class.

In any of the following cases, this operation fails.

- for the class whose name is specified by `className`, the invoking process does not hold at least one of the roles to be released;
- one or more objects affected by this operation are taking part in an in-progress CCB operation, and one of the roles to be released is that of the CCB validator;
- one or more of the affected objects are taking part in an in-progress administrative operation, and one of the roles to be released is that of the runtime 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 `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 the name of a runtime object class.

`SA_AIS_ERR_NOT_EXIST` - The name specified by the `className` parameter is not the name of an existing object class, or the invoking process does not hold at least one of the roles to be released for the class whose name is specified by `className`.

`SA_AIS_ERR_BUSY` - This value is returned if

- one or more objects affected by this operation are taking part in an in-progress CCB operation, and one of the roles to be released is that of the CCB validator, or
- one or more of the affected objects are taking part in an in-progress administrative operation, and one of the roles to be released is that of the runtime owner.

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immOiHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOiInitialize_3()`, `saImmOiClassImplementerSet_3()`

5.4.5 saImmOiObjectImplementerSet_3()

Prototype

```
SaAisErrorT saImmOiObjectImplementerSet_3(
      SaImmOiHandleT immOiHandle,
      SaImmOiRoleFlagsT role,
      const SaNameT *objectName,
      SaImmScopeT scope
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`role` - [in] Role of this object implementer. The `SaImmOiRoleFlagsT` type is defined in Section 5.2.3 on page 117.

`objectName` - [in] Pointer to the object name. The `SaNameT` type is defined in [2].

`scope` - [in] Scope of the operation. The `SaImmScopeT` type is defined in Section 4.2.11 on page 38.

Description

An object implementer whose name is associated with the handle immOiHandle invokes this function to inform the IMM Service that the object implementer assumes the role or roles specified by the role parameter for the objects identified 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 the name to which objectName points.
- If scope is SA_IMM_SUBLEVEL, the scope of the operation is the object designated by the name to which objectName points and its direct children.
- If scope is SA_IMM_SUBTREE, the scope of the operation is the object designated by the name to which objectName points and the entire subtree rooted at that object.

If this operation succeeds, and the role parameter contains the SA_IMM_ROLE_RUNTIME_OWNER flag, the current process becomes the runtime owner of the targeted objects; additionally, for each targeted object, the IMM Service adds the implementer name associated with the handle immOiHandle (if not yet present) to the SA_IMM_ATTR_RUNTIME_OWNER_NAME attribute of the object. This operation fails if one of the targeted objects has already a runtime owner whose name is different from the implementer name associated with the handle immOiHandle.

If this operation succeeds and the role parameter contains the flag SA_IMM_ROLE_CCB_VALIDATOR or SA_IMM_ROLE_CCB_APPLIER, the IMM Service adds the implementer name associated with the handle immOiHandle (if not yet present) to the SA_IMM_ATTR_VALIDATOR_NAME or SA_IMM_ATTR_APPLIER_NAME attribute of the targeted objects, respectively. Additionally, the current process becomes for each of the targeted objects a CCB validator or a CCB applier, respectively.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

`SA_AIS_ERR_BAD_HANDLE` - The handle `immOiHandle` is invalid, since it is corrupted, uninitialized, has already been finalized, or it is not associated with an implementer name.

`SA_AIS_ERR_INIT` - The corresponding previous invocation of `saImmOiInitialize_3()` to initialize the IMM Service object implementer library and obtain the handle specified by the `immOiHandle` parameter was incomplete, since one or more of the following callback functions, depending on the `role` parameter, were missing:

- `role` parameter contains the `SA_IMM_ROLE_RUNTIME_OWNER` flag: `SaImmOiRtAttrUpdateCallbackT` and `SaImmOiAdminOperationCallbackT_3`;
- `role` parameter contains the `SA_IMM_ROLE_CCB_APPLIER flag`: `SaImmOiCcbApplyCallbackT_3`;
- `role` parameter contains the `SA_IMM_ROLE_CCB_VALIDATOR flag`: `SaImmOiCcbValidateCallbackT`, `SaImmOiCcbFinalizeCallbackT`, and `SaImmOiCcbAbortCallbackT_3`.

`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 name to which the `objectName` parameter points is not the name of an existing object.

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

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immOiHandle` was acquired before the cluster node left the cluster membership.

See Also

saImmOiInitialize_3(), saImmOiObjectImplementerRelease_3(), SaImmOiRtAttrUpdateCallbackT, SaImmOiAdminOperationCallbackT_3, SaImmOiCcbApplyCallbackT_3, SaImmOiCcbValidateCallbackT, SaImmOiCcbFinalizeCallbackT, SaImmOiCcbAbortCallbackT_3

5.4.6 saImmOiObjectImplementerRelease_3()

Prototype

```
SaAisErrorT saImmOiObjectImplementerRelease_3(
      SaImmOiHandleT immOiHandle,
      SaImmOiRoleFlagsT role,
      const SaNameT *objectName,
      SaImmScopeT scope
);
```

Parameters

immOiHandle - [in] The handle which was obtained by a previous invocation of the saImmOiInitialize_3() function and which identifies this particular initialization of the Information Model Management Service. The SaImmOiHandleT type is defined in Section 5.2.1 on page 117.

role - [in] Role of this object implementer. The SaImmOiRoleFlagsT type is defined in Section 5.2.3 on page 117.

objectName - [in] Pointer to the object name. The SaNameT type is defined in [2].

scope - [in] Scope of the operation. The SaImmScopeT type is defined in Section 4.2.11 on page 38.

Description

An object implementer whose name is associated with the handle immOiHandle invokes this function to inform the IMM Service that the object implementer no longer holds the role or roles specified by the role parameter for the set of objects identified by scope and the name to which objectName points.

An object implementer can release just some of the roles for which it has registered. It may, for example, register as both CCB validator and CCB applier, and then, at a later time, unregister as CCB applier. It will then remain CCB validator.

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 the name to which `objectName` points.
- If `scope` is `SA_IMM_SUBLEVEL`, the scope of the operation is the object designated by the name to which `objectName` points and its direct children.
- If `scope` is `SA_IMM_SUBTREE`, the scope of the operation is the object designated by the name to which `objectName` points and the entire subtree rooted at that object.

If this operation succeeds and the `role` parameter contains the flag `SA_IMM_ROLE_CCB_VALIDATOR`, `SA_IMM_ROLE_CCB_APPLIER`, or `SA_IMM_ROLE_RUNTIME_OWNER`, the IMM Service removes the implementer name associated with the handle `immOiHandle` from the `SA_IMM_ATTR_VALIDATOR_NAME`, `SA_IMM_ATTR_APPLIER_NAME`, or `SA_IMM_ATTR_RUNTIME_OWNWR_NAME` attribute of all the targeted objects, respectively. If no more implementer names remain in any of these attributes, the attribute is removed.

If this operation succeeds and the `role` parameter contains the `SA_IMM_ROLE_RUNTIME_OWNER` flag, the IMM Service removes all non-persistent cached runtime attributes from all the targeted objects.

In any of the following cases, this operation fails.

- for one or more of the targeted objects, the invoking process does not hold at least one of the roles to be released;
- one or more objects affected by this operation are taking part in an in-progress CCB operation, and one of the roles to be released is that of the CCB validator;
- one or more of the affected objects are taking part in an in-progress administrative operation, and one of the roles to be released is that of the runtime 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 `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 name to which the `objectName` parameter points is not the name of an existing object, or the invoking process does not hold at least one of the roles to be released for one or more of the targeted objects.

`SA_AIS_ERR_BUSY` - This value is returned if

- one or more objects affected by this operation are taking part in an in-progress CCB operation, and one of the roles to be released is that of the CCB validator, or
- one or more of the affected objects are taking part in an in-progress administrative operation, and one of the roles to be released is that of the runtime owner.

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immOiHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOiInitialize_3()`, `saImmOiClassImplementerSet_3()`

5.5 CCB Validator Callbacks

When the user of the object management API requests the IMM Service to apply all change requests contained in a CCB by invoking the `saImmOmCcbApply_3()` function, the CCB enters the critical region, and the IMM Service applies the CCB in two steps:

1. In the first step, the IMM Service indicates to each CCB validator that validates at least one object for which the CCB holds one or more changes that the CCB is now complete and that the CCB validator must perform a local and global validation of the entire set of CCB changes (the meaning of these terms is explained further down). This indication is done by invoking the `saImmOiCcbValidateCallback()` callback function of each CCB validator. If any of the CCB validators returns an error or does not respond within the required time limit (specified by the `saImmOiTimeout` attribute in the `SaImmMngt` object class, shown in Section 7.2 on page 177), the attempt to apply the CCB fails, and the following actions are performed:
   - The IMM Service informs all CCB validators affected by the CCB that the CCB is aborted by invoking the `saImmOiCcbAbortCallback()` callback function of each CCB validator. When this callback of a CCB validator is invoked, the CCB validator shall dispose of the corresponding CCB identifier (as well as of any associated state), as the IMM Service may re-use the same CCB identifier to designate another set of changes later.
   - The `saImmOmCcbApply_3()` function returns an error, and the CCB leaves the critical section.
2. If all CCB validators agree with the proposed changes, the IMM Service applies the changes, and it then invokes the `saImmOiCcbApplyCallback()` callback function of each CCB applier as described in Section 5.6 on page 150 to inform them that the CCB has been applied.
   Subsequently, the IMM Service invokes the `saImmOiCcbFinalizeCallback()` callback function of all CCB validators before it returns to the caller of the `saImmOmCcbApply_3()` function. When its `saImmOiCcbFinalizeCallback()` callback is invoked, the CCB validator shall dispose of the corresponding CCB identifier (as well as of any associated state) for the same reason given in 1.).

Each CCB-related callback is invoked with a CCB identifier as a parameter.

The same CCB initialized with `saImmOmCcbInitialize_3()` may hold changes validated by different CCB validators. The IMM Service guarantees that the CCB identifiers passed to the different CCB validators are identical, meaning that the scope of the CCB identifier is global to the entire cluster.

A CCB validator shall obtain the changes contained in a CCB by invoking the CCB iterator functions described in Section 5.7 on page 153. Using an appropriate `immHandle`, it can also invoke the object search API functions described in Section 4.5 on page 61 and the object access API functions of Section 4.6 on page 69.

All changes are applied to the SA Forum Information Model by the IMM Service and deployed by the CCB appliers as a single transaction. Thus, the validation only considers the new state (resulting from all proposed changes) that needs to be validated prior to the application of the CCB. As a consequence, CCB validators are requested to perform two types of validation, local and global validation, when their `saImmOiCcbValidateCallback()` is invoked:

- **Local validation** consists of type and constraints checking of an attribute or an object, and it excludes any dependency checking, such as validating the impact or the consistency of the modification with respect to other attributes or objects.
- **Global validation** consists of making sure that the configuration of the SA Forum Information Model, as it would appear if all the modifications in the CCB were applied, is consistent and valid from the CCB validator's perspective.

Note: If the data model allows for a configuration that a CCB applier cannot accept, perhaps due to memory constraints, there must be a CCB validator which has validation code to guard against such configuration changes.

When a change request is added to the CCB, the IMM Service checks that the SA Forum 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.

The Information Model Management Service may also perform a local validation of the object attributes against the specification of the object class to which the object belongs.

If a CCB validator or a CCB applier either registers or unregisters itself while a CCB is in progress, and the CCB holds changes for objects this CCB validator or CCB applier implements, the IMM Service aborts the CCB, unless the CCB was initialized with the appropriate `SA_IMM_CCB_ALLOW_ABSENT_VALIDATORS` and `SA_IMM_CCB_ALLOW_ABSENT_APPLIERS` flags set. Note that if the IMM Service has applied the changes contained in the successfully validated CCB to the SA Forum Information Model (that is, the changes have been persisted by the IMM Service), the CCB cannot be aborted under any circumstances.

Appendix B provides sequence diagrams that show the detailed sequence of API operations for configuration changes using CCBs.

5.5.1 SaImmOiCcbValidateCallbackT

Prototype

```
typedef SaAisErrorT (*SaImmOiCcbValidateCallbackT)(
     SaImmOiHandleT immOiHandle,
     SaImmCcbIdT ccbId
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`ccbId` - [in] CCB identifier. The `SaImmCcbIdT` type is defined in Section 4.2.15 on page 41.

Description

The IMM Service invokes this callback function to inform CCB validators that the CCB identified by `ccbId` is now complete. The invoked process must perform a local and a global validation (see Section 5.5 on page 144) of the configuration, as it would appear if the CCB had been applied to the SA Forum Information Model and deployed in the SA Forum system by CCB appliers, and it must additionally ensure that no errors will be generated when these changes are effectively applied.

If all CCB validators that implement and validate objects changed by the CCB agree with the changes (they return `SA_AIS_OK`), the IMM Service will apply the changes to the SA Forum Information Model and then invoke the `saImmOiCcbApplyCallback()` callback functions of the CCB appliers to notify them that the CCB has been applied.

If any CCB validator fails to return from the `saImmOiCcbValidateCallback()` function within the time interval specified by the `saImmOiTimeout` attribute (defined in the `SaImmMngt` object class, shown in Section 7.2 on page 177), the in-progress CCB is failed by the IMM Service.

The IMM Service invokes this callback in the context of a thread of a CCB validator calling `saImmOiDispatch()` with the handle `immOiHandle` that was specified when the process invoked `saImmOiImplementerSet()`, `saImmOiObjectImplementerSet_3()`, or `saImmOiClassImplementerSet_3()` to become a registered CCB validator for one or more objects or classes of objects changed by the CCB identified by the `ccbId` parameter.

Return Values

`SA_AIS_OK` - The function completed successfully. The global validation was successful and the CCB validator agrees to apply the CCB.

`SA_AIS_ERR_NO_MEMORY` - The CCB validator is out of memory and cannot allocate the memory required to later apply all requested changes.

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

`SA_AIS_ERR_BAD_OPERATION` - The validation by the CCB validator of all change requests contained in the CCB failed.

See Also

`saImmOmCcbApply_3()`, `saImmOiInitialize_3()`, `saImmOiDispatch()`, `saImmOiImplementerSet()`, `saImmOiClassImplementerSet_3()`, `saImmOiObjectImplementerSet_3()`

5.5.2 SaImmOiCcbAbortCallbackT_3

Prototype

```
typedef void (*SaImmOiCcbAbortCallbackT_3)(
     SaImmOiHandleT immOiHandle,
     SaImmCcbIdT ccbId
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`ccbId` - [in] CCB identifier. The `SaImmCcbIdT` type is defined in Section 4.2.15 on page 41.

Description

The IMM Service invokes this callback function to inform CCB validators that the CCB identified by `ccbId` is aborted, so that they can dispose of the CCB identifier `ccbId` and of any associated state.

The IMM Service invokes this callback in the context of a thread of a CCB validator calling `saImmOiDispatch()` with the handle `immOiHandle` that was specified when the process invoked `saImmOiImplementerSet()`, `saImmOiObjectImplementerSet_3()`, or `saImmOiClassImplementerSet_3()` to become a registered CCB validator for one or more objects or classes of objects changed by the CCB identified by the `ccbId` parameter.

Return Values

None

See Also

`saImmOmCcbApply_3()`, `saImmOiInitialize_3()`, `saImmOiDispatch()`, `saImmOiImplementerSet()`, `saImmOiClassImplementerSet_3()`, `saImmOiObjectImplementerSet_3()`

5.5.3 SaImmOiCcbFinalizeCallbackT

Prototype

```
typedef void (*SaImmOiCcbFinalizeCallbackT)(
        SaImmOiHandleT immOiHandle,
        SaImmCcbIdT ccbId
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_23()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`ccbId` - [in] CCB identifier. The `SaImmCcbIdT` type is defined in Section 4.2.15 on page 41.

Description

The IMM Service invokes this callback function to inform CCB validators that the CCB identified by `ccbId` has been applied, so that they can dispose of the CCB identifier `ccbId` and of any associated state.

The IMM Service invokes this callback in the context of a thread of a CCB validator calling `saImmOiDispatch()` with the handle `immOiHandle` that was specified when the process invoked `saImmOiImplementerSet()`, `saImmOiObjectImplementerSet_3()`, or `saImmOiClassImplementerSet_3()` to become a registered CCB validator for one or more objects or classes of objects changed by the CCB identified by the `ccbId` parameter.

Return Values

None

See Also

`saImmOmCcbApply_3()`, `saImmOiInitialize_3()`, `saImmOiDispatch()`, `saImmOiImplementerSet()`, `saImmOiClassImplementerSet_3()`, `saImmOiObjectImplementerSet_3()`

5.6 CCB Applier Callback

The IMM Service invokes the `saImmOiCcbApplyCallback()` callback function of a CCB applier to notify the CCB applier that the configuration of objects in which it is interested has been modified.

The IMM Service invokes this callback after a modification has been approved, that is, after all CCB validators have returned successfully from their `saImmOiCcbValidateCallback()` function, and the CCB has been applied to the SA Forum Information Model.

The IMM Service invokes the `saImmOiCcbApplyCallback()` function of all CCB appliers according to their rank (`saImmCcbApplierRank` configuration attribute in the `SaImmCcbApplier` object class, see Section 7.2 on page 177). The lower the value of this attribute, the higher the rank. CCB appliers with a lower rank are invoked after all CCB appliers with a higher rank have completed the operation. CCB appliers with the same rank are invoked in an arbitrary order. A CCB applier with no configured rank value is invoked after all CCB appliers with configured rank have completed the operation.

To ensure that the system is in a consistent state, no new CCB is allowed to enter the critical region until all CCB appliers of the current CCB have completed their operation, and until all modifications of the current CCB have been deployed. However, to avoid infinite blocking by a CCB applier, the IMM Service waits for the return of the `saImmOiCcbApplyCallback()` callback function at most the time specified by the `saImmOiTimeout` attribute (defined in the `SaImmMngt` object class, shown in Section 7.2 on page 177).

The CCB identifier provided in the `SaImmOiCcbApplyCallbackT_3` function has the same value as the CCB identifier provided in the associated CCB validator callbacks (see Section 5.5 on page 144).

A CCB applier can obtain the configuration changes by invoking functions of the CCB iterator API (see Section 5.7). Using an appropriate `immHandle`, it can also invoke the object search API functions (see Section 4.5 on page 61), or the object access API functions (see Section 4.6 on page 69).

If a process acting as a CCB applier exits or simply unregisters during the application of the CCB, and another process (or the same process, if it simply unregistered) registers for the same CCB applier role while the application of the CCB is still in progress, the IMM Service invokes the `saImmOiCcbApplyCallback()` on the newly registered process.
If no process registers for the particular CCB applier role within this time interval, the new CCB applier (whenever it registers subsequently) will need to obtain the configuration as it would do at initial startup via the object management API.
As by default CCB appliers must be registered for all the objects changed in a CCB

for the CCB change to succeed, no subsequent CCB involving objects of the unregistered appliers will succeed unless the SA_IMM_CCB_ALLOW_ABSENT_APPLIERS flag is set for the CCB.

As the IMM Service may re-use the same CCB identifier to designate another set of changes later, a CCB applier shall dispose of the corresponding CCB identifier (as well as of any associated state) after it responds to the saImmOiCcbApplyCallback() callback call.

Appendix B provides sequence diagrams that explain the detailed sequence of API operations for configuration changes using CCBs.

5.6.1 SaImmOiCcbApplyCallbackT_3

Prototype

```
typedef void (*SaImmOiCcbApplyCallbackT_3)(
      SaImmOiHandleT immOiHandle,
      const SaNtfCorrelationIdsT *correlationIds,
      SaImmCcbIdT ccbId
);
```

Parameters

immOiHandle - [in] The handle which was obtained by a previous invocation of the saImmOiInitialize_3() function and which identifies this particular initialization of the Information Model Management Service. The SaImmOiHandleT type is defined in Section 5.2.1 on page 117.

correlationIds – [in] Pointer to the correlation identifiers associated with the CCB being applied. The rootCorrelationId and parentCorrelationId fields are in parameters. The rootCorrelationId field holds the root correlation identifier that has been provided by the invoker of the saImmOmCcbApply_3() function. The parentCorrelationId field holds the notification identifier of the notification that the IMM Service sends to report that the CCB is being applied. The notificationId field is not used. The SaNtfCorrelationIdsT type is defined in [6].

ccbId - [in] CCB identifier. The SaImmCcbIdT type is defined in Section 4.2.15 on page 41.

Description

The IMM Service invokes this callback function to inform a CCB applier that the CCB identified by `ccbId` has been applied by the IMM Service.

All configuration changes have already been validated by the CCB validators in previous calls to their `saImmOiCcbValidateCallback()` callback functions.

Each CCB applier is responsible for determining the effect of the configuration changes.

If any CCB applier fails to return from the `saImmOiCcbApplyCallback()` function within the time interval specified by the `saImmOiTimeout` attribute (defined in the `SaImmMngt` object class, shown in Section 7.2 on page 177), the IMM Service returns `SA_AIS_ERR_DEPLOYMENT` to the `saImmOmCcbApply_3()` call, but this does not invalidate the CCB. The changes are persisted in the SA Forum Information Model maintained by the IMM Service (see paragraph in Section 5.6 on page 150).

The IMM Service invokes this callback in the context of a thread of a CCB applier process calling `saImmOiDispatch()` with the handle `immOiHandle` that was specified when the process invoked `saImmOiImplementerSet()`, `saImmOiObjectImplementerSet_3()`, or `saImmOiClassImplementerSet_3()` to become a registered CCB applier for one or more objects or classes of objects changed by the CCB identified by the `ccbId` parameter.

Return Values

None

See Also

`saImmOmCcbApply_3()`, `SaImmOiCcbValidateCallbackT`, `saImmOiImplementerSet()`, `saImmOiClassImplementerSet_3()`, `saImmOiObjectImplementerSet_3()`, `saImmOiDispatch()`

5.7 CCB Iterator API

The API functions in this section are used to iterate through configuration changes associated with a given CCB identifier. 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 using a search iterator.

What objects and values should be returned can be specified when the iterator is initialized. An object implementer may select to receive only objects in the CCB for which it is a registered object implementer. For each found object the returned values can be either the name of the modified object alone, or the object name together with its attributes. If the latter option is requested, one can specify whether all the attributes or only the modified ones are returned.

Each invocation of the `saImmOiCcbIteratorNext()` function returns the next modified object in the SA Forum Information Model that matches the iterator options.

The `traverse` parameter specifies the direction of the search for this next modified object, namely whether the subtree of the last found object is searched or if the iteration continues at the same level or above.

When these iterator API functions are used during the validation or commit phase of a CCB, it is guaranteed that no changes of other pending CCBs can affect the object search, because no two CCBs are allowed to be in the critical region, that is, in the validate or commit phases (see Section 3.1 on page 26) at the same time.

The iteration is terminated by invoking the finalize function.

The CCB iterator API should only be used by a CCB validator during the validation phase, or by a CCB applier during the commit phase. For this API, the CCB identifier becomes valid when the `saImmOmCcbApply_3()` function is invoked by an object management process and becomes invalid when this function returns, that is, as soon as the CCB has been applied. In other words, the API is available while the CCB is in the critical region and for the CCB in the critical region.

When the IMM Service returns from the `saImmOmCcbApply_3()` invocation, it finalizes any CCB iterator handle that is still initialized for the CCB.

5.7.1 saImmOiCcbIteratorInitialize()

Prototype

```
SaAisErrorT saImmOiCcbIteratorInitialize(
     SaImmOiHandleT immOiHandle,
     SaImmCcbIdT ccbId,
     SaImmOiCcbIteratorOptionT iteratorOptions,
     SaImmOiCcbIteratorHandleT *iteratorHandle
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`ccbId` - [in] CCB identifier. The `SaImmCcbIdT` type is defined in Section 4.2.15 on page 41.

`iteratorOptions` - [in] Specifies what objects are searched and which attribute values must be returned for each found object. The `SaImmOiCcbIteratorOptionT` type is defined in Section 5.2.4 on page 118.

`iteratorHandle` - [out] Pointer to the iterator handle used later to iterate through the SA Forum Information Model to search for configuration changes. The `SaImmOiCcbIteratorHandleT` type is defined in Section 5.2.1 on page 117.

Description

This function initializes an iterator for changes induced by the CCB specified by the `ccbId` parameter.
The `iteratorOptions` parameter specifies which information is returned for each modified object.

If this function completes successfully, the `iteratorHandle` parameter points to the iterator handle to be used in the other functions of the iterator API.

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, this error is returned if the specified `ccbId` parameter is unknown to the IMM Service or invalid to be used for iteration, as the CCB is not yet complete.

`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 invoked function is not supported in the version specified in the call to initialize this instance of the IMM Service library.

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immOiHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOiInitialize_3()`, `saImmOiCcbIteratorNext()`, `saImmOiCcbIteratorFinalize()`

5.7.2 saImmOiCcbIteratorNext()

Prototype

```
SaAisErrorT saImmOiCcbIteratorNext(
        SaImmOiCcbIteratorHandleT iteratorHandle,
        SaImmOiObjectTraverseT traverse,
        SaImmOiObjectChangeT *objectChange,
        SaNameT *objectName,
        SaImmAttrValuesT_2 ***attributes
);
```

Parameters

`iteratorHandle` - [in] Iterator handle returned by the `saImmOiCcbIteratorInitialize()` function. The `SaImmOiCcbIteratorHandleT` type is defined in Section 5.2.1 on page 117.

`traverse` - [in] Specifies whether the subtree of the previously found object should be entered or not when iterating for modified objects. This parameter is ignored in the first invocation of this function subsequent to the invocation of the corresponding `saImmOiCcbIteratorInitialize()` function. The `SaImmOiObjectTraverseT` type is defined in Section 5.2.6 on page 119.

`objectChange` - [out] Pointer to the change status of the found object. The `SaImmOiObjectChangeT` is defined in Section 5.2.5 on page 119.

`objectName` - [out] In the first invocation of this function after the initialization of the iterator, this parameter points to the name of the first object modified by the CCB. In subsequent invocations of this function, this parameter points to the name of the object found next with respect to the last found object in the direction indicated by the `traverse` parameter. The `SaNameT` type is defined in [2].

`attributes` - [out] Pointer to a pointer to a NULL-terminated array of pointers to data structures holding the names and values of attributes of the object whose name is pointed to by `objectName`. The attributes were selected when the search was initialized. The `SaImmAttrValuesT_2` type is defined in Section 4.2.8 on page 37.

Description

This function is used to obtain the next object (or the first object if this is the first invocation of this function subsequent to the invocation of the corresponding `saImmOiCcbIteratorInitialize()` function) that has been modified and matches the `iteratorOptions` specified in the `saImmOiCcbIteratorInitialize()` function when performing a depth-first search of the configuration tree for modified objects.

Each object is reported once. The status of the object is reported with respect to its last committed status, regardless of the number of changes the object went through to reach this status. For example, if the object was newly created, and then some of its attributes were modified, the object is reported as newly created, but the attributes referred to by the `attributes` parameter will contain the latest values cumulating all the subsequent modifications.

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 `saImmOiCcbIteratorNext()` or `saImmOiCcbIteratorFinalize()` for the same iterator handle.

If the handle `iteratorHandle` was obtained by specifying `SA_IMM_CCB_ITERATOR_OBJECT_NAME_ONLY` in the `iteratorOptions` parameter of the corresponding `saImmOiCcbIteratorInitialize()` call, no attribute names and values will be returned by this call, and the pointer to which the `attributes` parameter refers is 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 `iteratorHandle` 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_NOT_EXIST` - All objects matching the iteration criteria either have already been returned to the calling process, or they have been explicitly skipped by specifying `SA_IMM_OBJECT_CONTINUE` in the `traverse` parameter. The caller can now invoke the `saImmOiCcbIteratorFinalize()` function.

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `iteratorHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOiInitialize_3()`, `saImmOiCcbIteratorInitialize()`, `saImmOiCcbIteratorFinalize()`

5.7.3 saImmOiCcbIteratorFinalize()

Prototype

```
SaAisErrorT saImmOiCcbIteratorFinalize(
     SaImmOiCcbIteratorHandleT iteratorHandle
);
```

Parameters

`iteratorHandle` - [in] Iterator handle returned by the `saImmOiCcbIteratorInitialize()` function. The `SaImmOiCcbIteratorHandleT` type is defined in Section 5.2.1 on page 117.

Description

This function finalizes the CCB iteration initialized by a previous call to the `saImmOiCcbIteratorInitialize()` function. It frees all memory previously allocated by that iteration and, in particular, the memory used to return attribute names and values in the previous `saImmOiCcbIteratorNext()` invocation.

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

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `iteratorHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOiInitialize_3()`, `saImmOiCcbIteratorInitialize()`, `saImmOiCcbIteratorNext()`

5.8 Runtime Owner APIs

As has been explained in Section 5.4 on page 128, a runtime owner is responsible for managing runtime objects and runtime attributes of either configuration or runtime objects and for carrying out administrative operations issued on these objects. The next two sections describe the corresponding APIs.

5.8.1 Runtime Objects Management API

The set of functions contained in this section are used by a registered runtime owner 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-cached runtime attributes are not accessible when a runtime owner is not registered for the objects to which these attributes belong.

A runtime attribute whose value is cached by the IMM Service must be updated by its runtime owner whenever the attribute's value changes. The value of non-cached attributes must be updated by the runtime owner only when the IMM Service requests such an update by invoking the runtime owner's `saImmOiRtAttrUpdateCallback`() 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 by using the object management API should typically not be cached.

5.8.1.1 saImmOiRtObjectCreate_2()

Prototype

```
SaAisErrorT saImmOiRtObjectCreate_2(
     SaImmOiHandleT immOiHandle,
     const SaImmClassNameT className,
     const SaNameT *parentName,
     const SaImmAttrValuesT_2 *const *attrValues
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`className` - [in] Object class name. The `SaImmClassNameT` type is defined in Section 4.2.2 on page 34.

`parentName` - [in] Pointer to the name of the parent of the new object. The `SaNameT` type is defined in [2].

`attrValues`- [in] Pointer to a NULL-terminated array of pointers to attribute descriptors. The `SaImmAttrValuesT_2` type is defined in Section 4.2.8 on page 37.

Description

This function creates a new IMM Service runtime object.

The new object is created as a child of the object designated by the name to which `parentName` points. If `parentName` is set to NULL, the new object is created as a top level object.

The attributes referred to by the pointers in the array of pointers to which the `attrValues` parameter points must match the object class definition. These attributes can only be cached runtime attributes. One and only one of these attributes must have 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_RUNTIME_OWNER_NAME` must not be specified by the `attrValues` descriptors, as these attributes are automatically set by the IMM Service.

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

The invoking process becomes the registered runtime owner of the new 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 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 the name of a configuration object class,
- there is no valid RDN attribute specified for the new object,
- some cached attributes do not have values,
- the class referred by the className parameter includes a persistent attribute but the parent object indicated by the parentName parameter and some of its ancestors are non-persistent objects,
- the attrValues parameter includes:
  - attributes with values that do not match the defined value type for the attribute,
  - multiple values for a single-valued attribute, and
  - non-cached runtime attributes.

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 name to which the parentName parameter points 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 the object class designated by the name className.

SA_AIS_ERR_EXIST - An object with the same name already exists.

`SA_AIS_ERR_NAME_TOO_LONG` - The size of the new object's DN is greater than `SA_MAX_NAME_LENGTH`.

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immOiHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOiInitialize_3()`

5.8.1.2 saImmOiRtObjectDelete()

Prototype

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

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`objectName` - [in] Pointer to the object name. The `SaNameT` type is defined in [2].

Description

This function deletes the object designated by the name to which the `objectName` parameter points 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 - This value is returned due to one or more of the following reasons:

- at least one of the targeted objects is a configuration object;
- at least one of the targeted object is a runtime object not implemented by the invoking process.

SA_AIS_ERR_NOT_EXIST - The name to which the objectName parameter points is not the name of an existing object.

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle immOiHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOiInitialize_3()

5.8.1.3 saImmOiRtObjectUpdate_2()

Prototype

```
SaAisErrorT saImmOiRtObjectUpdate_2(
        SaImmOiHandleT immOiHandle,
        const SaNameT *objectName,
        const SaImmAttrModificationT_2 *const *attrMods
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`objectName` - [in] Pointer to the name of the updated object. The `SaNameT` type is defined in [2].

`attrMods` - [in] Pointer to a NULL-terminated array of pointers to descriptors of the modifications to perform. The `SaImmAttrModificationT_2` type is defined in Section 4.2.10 on page 38.

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_RUNTIME_OWNER_NAME` must not be modified.

This operation fails and returns the `SA_AIS_ERR_BAD_OPERATION` error code 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 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 targeted object is not implemented by the invoking process.

`SA_AIS_ERR_NOT_EXIST` - The name to which the `objectName` parameter points is not the name of an existing object, or one or more attribute names specified by the `attrMods` parameter are not valid for the object class.

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

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

`SA_AIS_ERR_UNAVAILABLE` - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle `immOiHandle` was acquired before the cluster node left the cluster membership.

See Also

`saImmOiInitialize_3()`

5.8.1.4 SaImmOiRtAttrUpdateCallbackT

Prototype

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

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`objectName` - [in] Pointer to the name of the object for which the update is requested. The `SaNameT` type is defined in [2].

`attributeNames` - [in] Pointer to a NULL-terminated array of attribute names for which values must be updated. The `SaImmAttrNameT` type is defined in Section 4.2.2 on page 34.

Description

The IMM Service invokes this callback function to request a runtime owner to update the values of some attributes of an IMM Service object. These attributes are attributes whose values are not cached by the IMM Service. The target object is identified by the name to which `objectName` points. The process must use the `saImmOiRtObjectUpdate_2()` 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_2()` call to set the attribute value to the empty set.

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

The IMM Service invokes this callback in the context of a thread of a runtime owner calling `saImmOiDispatch()` with the handle `immOiHandle` that was specified when the process invoked `saImmOiRtObjectCreate_2()` or `saImmOiImplementerSet()` and became the registered runtime owner for the object to which the `objectName` parameter points.

Return Values

`SA_AIS_OK` - The function completed successfully.

`SA_AIS_ERR_NO_MEMORY` - The runtime owner is out of memory and cannot provide the service.

`SA_AIS_ERR_NO_RESOURCES` - The runtime owner is out of required resources (other than memory) to provide the service.

`SA_AIS_ERR_FAILED_OPERATION` - The runtime owner failed to update the requested attributes due to an error occurring in the `saImmOiRtObjectUpdate_2()` invocation.

See Also

`saImmOiInitialize_3()`, `saImmOiDispatch()`, `saImmOiRtObjectCreate_2()`, `saImmOiImplementerSet()`

5.8.2 Administrative Operations

5.8.2.1 SaImmOiAdminOperationCallbackT_3

Prototype

```
typedef void (*SaImmOiAdminOperationCallbackT_3) (
      SaImmOiHandleT immOiHandle,
      SaInvocationT invocation,
      const SaNtfCorrelationIdsT *correlationIds,
      const SaNameT *objectName,
      SaImmAdminOperationIdT operationId,
      const SaImmAdminOperationParamsT_2 *const *params
);
```

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`invocation` - [in] Used to match this invocation of `SaImmOiAdminOperationCallbackT_3` with the corresponding invocation of `saImmOiAdminOperationResult()`. The `SaInvocationT` type is defined in [2].

`correlationIds` – [in] Pointer to the correlation identifiers associated with the administrative operation. The `rootCorrelationId` and `parentCorrelationId` fields are `in` parameters. The `rootCorrelationId` field holds the root correlation identifier that has been provided by the invoker of the administrative operation. The `parentCorrelationId` field holds the notification identifier of the notification that the IMM Service sends to report the invocation of the administrative operation. The `notificationId` field is not used. The `SaNtfCorrelationIdsT` type is defined in [6].

`objectName` - [in] Pointer to the object name. The `SaNameT` type is defined in [2].

`operationId` - [in] Identifier of the administrative operation. The `SaImmAdminOperationIdT` type is defined in Section 4.2.17 on page 42.

`params` - [in] Pointer to a NULL-terminated array of pointers to parameter descriptors. The `SaImmAdminOperationParamsT_2` type is defined in Section 4.2.18 on page 42.

Description

The IMM Service invokes this callback function to request a runtime owner to execute an administrative operation on the object designated by the name to which `objectName` points. The administrative operation identified by the `operationId` parameter has been initiated by an invocation of the `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()` functions.

Each element referred to by a pointer of the array of pointers to which the `params` parameter points represents an input parameter of the administrative operation to execute.

The IMM Service invokes this callback in the context of a thread of a runtime owner process calling `saImmOiDispatch()` with the handle `immOiHandle` that was specified when the process invoked `saImmOiImplementerSet()`, `saImmOiObjectImplementerSet()`, or `saImmOiClassImplementerSet()` for configuration objects or `saImmOiRtObjectCreate_2()` for runtime objects to become the registered runtime owner for the object to which the `objectName` parameter points.

The runtime owner indicates the success or failure of the administrative operation by invoking the `saImmOiAdminOperationResult()` function. The `saImmOiAdminOperationResult()` function can be invoked from the callback itself or outside the callback by any thread of the process that initialized the `immOiHandle`.

Return Values

None

See Also

`saImmOiInitialize_3()`, `saImmOmAdminOperationInvoke_3()`, `saImmOmAdminOperationInvokeAsync_3()`, `saImmOiAdminOperationResult()`, `saImmOiImplementerSet()`, `saImmOiObjectImplementerSet()`, `saImmOiClassImplementerSet()`, `saImmOiRtObjectCreate_2()`

5.8.2.2 saImmOiAdminOperationResult()

Prototype

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

Parameters

`immOiHandle` - [in] The handle which was obtained by a previous invocation of the `saImmOiInitialize_3()` function and which identifies this particular initialization of the Information Model Management Service. The `SaImmOiHandleT` type is defined in Section 5.2.1 on page 117.

`invocation` - [in] Used to match this invocation of `saImmOiAdminOperationResult()` with the previous corresponding invocation of the `SaImmOiAdminOperationCallbackT_3` callback. The `SaInvocationT` type is defined in [2].

`result` - [in] Result of the execution of the administrative operation. The `SaAisErrorT` type is defined in [2].

Description

An object implementer invokes this function to inform the IMM Service about the result of the execution of an administrative operation requested by the IMM Service by an invocation of the object implementer's `saImmOiAdminOperationCallback()` callback.

This function can be called only by the process for which its `saImmOiAdminOperationCallback()` callback has been invoked.

If the runtime owner exits or simply unregisters during the execution of the administrative operation and another process (or the same process, if it simply unregistered) registers for the same runtime owner role within the time interval specified by the `saImmOiTimeout` attribute (defined in the `SaImmMngt` object class, shown in Section 7.2 on page 177), the IMM Service invokes the `saImmOiAdminOperationCallback()` callback on the newly registered process. If no process registers within this time interval, the IMM Service— depending on how the administrative operation was initiated— either returns the

SA_AIS_ERR_TIMEOUT error to the corresponding invocation of the saImmOmAdminOperationInvoke_3() function or invokes the saImmOmAdminOperationInvokeCallback() function with the error parameter set to SA_AIS_ERR_TIMEOUT.

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_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node due to one of the two reasons:

- the cluster node has left the cluster membership;
- the cluster node has rejoined the cluster membership, but the handle immOiHandle was acquired before the cluster node left the cluster membership.

See Also

saImmOiInitialize_3(), SaImmOiAdminOperationCallbackT_3

6 IMM Service Administration API

This section describes the administrative API functions that the IMM Service exposes on behalf of itself to a system administrator. These API functions are described using a ‘C’ API syntax. The main clients of this administrative API are system management applications.

6.1 Administrative Operations on the IMM Service

Administrative operations on the IMM Service can be carried out using the IMM Service API functions `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()` (refer to Section 4.9 on page 100) on an object that represents the IMM Service and for which the IMM Service is the object implementer.

Return values are passed in the `operationReturnValue` parameter (see Section 4.9.1 on page 102).

6.2 Include File and Library Name

The following IMM Service header file containing declarations of data types and function prototypes must be included in the source of an application using the IMM Service Administration API:

```
#include <saImm.h>
```

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

```
libSaImm.so
```

6.3 Type Definitions

The specification of IMM Service Administration API requires the following type.

6.3.1 SaImmMngtAdminOperationT

```
typedef enum {
      SA_IMM_ADMIN_EXPORT = 1
} SaImmMngtAdminOperationT;
```

6.4 IMM Service Administration API

6.4.1 SA_IMM_ADMIN_EXPORT

Parameters

`operationId` - [in] = `SA_IMM_ADMIN_EXPORT`

`objectName` - [in] = The LDAP name of the object of class `SaImmService` that represents the IMM Service. The DN of this object is "`safRdn=immManagement`,`safApp=safImmService`".
For SA Forum naming conventions and rules, see [2].

`params` - [in] A pointer to a NULL-terminated array of pointers to parameter descriptors. The first parameter descriptor has the following format:

```
params[0].paramName = SA_IMM_ADMIN_EXPORT;
params[0].paramType = SA_IMM_ATTR_SASTRINGT;
params 0].paramBuffer = filePathname;
```

`filePathname` is the standard relative POSIX pathname of the file to which the IMM contents must be exported. This pathname is relative to an implementation defined root directory. The type of this parameter is `SaStringT`, defined in [2].

Description

This administrative operation requests the IMM Service to export all its persistent contents (class definitions as well as persistent objects and attributes) into a file whose relative pathname is specified by the `filePathname` parameter.

The persistent contents will be stored into the file according to the IMM XML Schema Definition (see [4]).

The `saImmExportFileUri` attribute of the `SaImmMngt` IMM configuration class (see Section 7.2 on page 177) shall be used to retrieve the file after the export operation completed.

operationReturnValue

`SA_AIS_OK` - The operation completely successfully.

`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 operation cannot be provided at this time. The caller may retry later. This error generally should be returned in cases where the requested administrative operation is valid but not currently possible.

`SA_AIS_ERR_INVALID_PARAM` - A parameter is not set correctly.

`SA_AIS_ERR_NO_MEMORY` - The IMM Service or a library is out of memory and cannot provide the service.

`SA_AIS_ERR_NO_RESOURCES` - There are insufficient resources to carry out the operation.

`SA_AIS_ERR_NOT_SUPPORTED` - This administrative operation is not supported by the type of entity denoted by the name to which `objectName` points.

See Also

-

7 IMM Service UML Information Model

The IMM Service Information Model is described in UML and has been organized in a UML class diagram.

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

The IMM Service UML class diagram has two object classes, which show the contained attributes and the administrative operations applicable on these classes (if any).

7.1 DN Formats for the IMM Service UML Class Diagram

Table 3 DN Formats for Objects of the IMM Service Class Diagram

Object Class	DN Formats for Objects of the Class
`SaImmMngt`	“`safRdn=immManagement`, `safApp=safImmService`”
`SaImmCcbApplier`	“`safCcbApplier=`...,`safRdn=immManagement`, `safApp=safImmService`

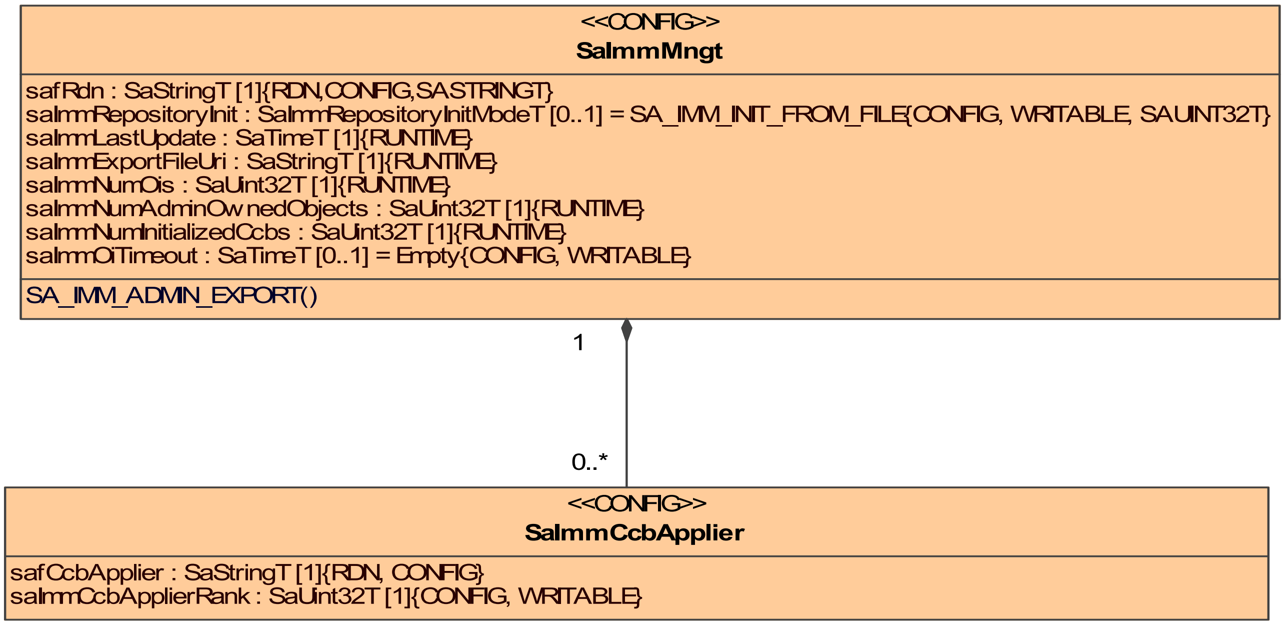
7.2 IMM Service UML Classes

The `SaImmMngt` configuration object class exports all IMM global attributes and administrative operations.

The `SaImmCcbApplier` configuration object class specifies a list of CCB appliers, each with a rank. The rank indicates the order that the IMM Service uses to invoke the apply CCB callbacks of these CCB appliers. For details, see Section 5.6 on page 150.

FIGURE 3 shows these two classes. A description of each attribute of these classes is found in the XMI file (see [3]). For additional details, refer to the SA Forum Overview document ([1]).

FIGURE 3 IMM Service UML Classes


<<CONFIG>>
SaImmMngt
safRdn : SaStringT [1]{RDN,CONFIG,SASTRINGT}
saImmRepositoryInit : SaImmRepositoryInitModeT [0..1] = SA_IMM_INIT_FROM_FILE{CONFIG, WRITABLE, SAUINT32T}
saImmLastUpdate : SaTimeT [1]{RUNTIME}
saImmExportFileUri : SaStringT [1]{RUNTIME}
saImmNumOis : SaUint32T [1]{RUNTIME}
saImmNumAdminOwnedObjects : SaUint32T [1]{RUNTIME}
saImmNumInitializedCcbs : SaUint32T [1]{RUNTIME}
saImmOiTimeout : SaTimeT [0..1] = Empty{CONFIG, WRITABLE}
SA_IMM_ADMIN_EXPORT()
1
0..*
<<CONFIG>>
SaImmCcbApplier
safCcbApplier : SaStringT [1]{RDN, CONFIG}
saImmCcbApplierRank : SaUint32T [1]{CONFIG, WRITABLE}

8 IMM Service Alarms and Notifications

The Information Model Management Service produces alarms and notifications to convey important information regarding the operational and functional state of the objects under its control to an administrator or a management system.

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

The vehicle to be used for producing alarms and notifications is the Notification Service of the Service Availability^{TM} Forum (abbreviated as NTF, see [6]), and hence the various notifications are partitioned into categories, as described in this service.

In some cases, this specification uses the word “Unspecified” for values of attributes that the vendor is at liberty to set to whatever makes sense in the vendor’s context, and the SA Forum has no specific recommendation regarding such values. Such values are generally optional from the CCITT Recommendation X.733 perspective (see [7]).

8.1 Setting Common Attributes

The following attributes of the notifications presented in Section 8.2 on page 181 are not shown in their description, as the generic description presented here applies to all of them:

- Notification Id - Depending on the Notification Service function used to send the notification, this attribute is either implicitly set by the Notification Service or provided by the caller.
- Notifying Object - DN of the entity generating the notification. This name must conform to the SA Forum AIS naming convention and must contain at least the `safApp` RDN value portion of the DN set to the specified standard RDN value of the SA Forum AIS Service generating the notification, that is, `safImmService`. For details on the AIS naming convention, refer to the Overview document ([1]).

The following notes apply to all IMM Service notifications presented in Section 8.2 on page 181:

- Correlated Notifications - Correlation ids are supplied to correlate notifications that have been generated because of a related cause. The correlated notifications attribute should include

- in the first position the root notification identifier of the related tree of notifications as described in the Notification Service specification (see [6]);
- in the second position the parent notification identifier of the same tree;
- in the third position the notification identifier of the sibling notification, if any. This sibling notification is the opening pair of the current notification such as the alarm that is being cleared or the start of an administrative operation or a configuration change that has ended.

If any of these notifications is unknown, the `SA_NTF_IDENTIFIER_UNUSED` value must be used. This value may be omitted in trailing positions.

- Notification Class Identifier - The `vendorId` field of the `SaNtfClassIdT` data structure must be set to `SA_NTF_VENDOR_ID_SAF`, and the `majorId` field must be set to `SA_SVC_IMM` (as defined in the `SaServicesT` enumeration in [2]) for all notifications that follow the standard formats described in this specification. The `minorId` field will vary based on the specific notification.

8.2 Information Model Management Service Notifications

The following subsections describe the notifications that an Information Model Management Service implementation shall produce.

8.2.1 Information Model Management Service Alarms

The Information Model Management Service does not issue any alarms at the time of publication of this specification.

8.2.2 Information Model Management Service Notifications of Miscellaneous Type

8.2.2.1 Administrative Operation Start

Description

The IMM Service sends the following notification when the `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()` functions are called.

The additional information field contains the administrative operation identifier (`operationId` parameter) and the administrative operation parameters (values referred to by the `params` parameter), if any.

Table 4 Administrative Operation Start

NTF Attribute Name	Mandatory/ Optional	Specified Value
Event Type	Mandatory	`SA_NTF_ADMIN_OPERATION_START`
Notification Object	Mandatory	LDAP DN of the object on which the administrative operation is invoked
Notification Class Identifier	NTF-Internal	`minorId` = `SA_IMM_NTFID_OP_START`, see Section 4.2.19 on page 42
Event Time	Mandatory	Time when either the `saImmOmAdminOperationInvoke_3()` or the `saImmOmAdminOperationInvokeAsync_3()` functions was invoked
Number of Correlated Notifications	Mandatory	0, 1, or 2
Correlated Notifications	Mandatory	`rootCorrelationId` and `parentCorrelationId` passed to either `saImmOmAdminOperationInvoke_3()` or to `saImmOmAdminOperationInvokeAsync_3()`
Number of Elements in Additional Information Array	Mandatory	At least 1
Additional Information `additionalInfo[0]`	Mandatory	{`SA_IMM_AI_ADMIN_OPERATION_ID`, `SA_NTF_VALUE_UINT64`, `operationId` parameter passed to `saImmOmAdminOperationInvoke_3()` or to `saImmOmAdminOperationInvokeAsync_3()`}
Additional Information `additionalInfo[i]`	Optional	{`SA_IMM_AI_ADMIN_OPERATION_ID`, `SA_NTF_VALUE_xxx`, value referred to by the `params` parameter passed to `saImmOmAdminOperationInvoke_3()` or to `saImmOmAdminOperationInvokeAsync_3()`}

8.2.2.2 Administrative Operation End

Description

The IMM Service sends the following notification after the runtime owner provided a result for the operation previously invoked by the `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()` functions. The IMM Service also sends this notification to inform about any kind of error that may be returned by the aforementioned functions and by the `saImmOmAdminOperationInvokeCallback()` function.

The first additional information field contains the administrative operation identifier. The second one contains the result of the administrative operation, which can be:

- the return value of the `saImmOmAdminOperationInvoke_3()` function, if this function was invoked;
- the return value of the `saImmOmAdminOperationInvokeAsync_3()` function, if this function was invoked, and the corresponding callback is not invoked;
- the value returned in the `error` parameter of the `saImmOmAdminOperationInvokeCallback()` function, if this function was invoked.

Table 5 Administrative Operation End

NTF Attribute Name	Mandatory/ Optional	Specified Value
Event Type	Mandatory	`SA_NTF_ADMIN_OPERATION_END`
Notification Object	Mandatory	LDAP DN of the object on which the administrative operation is invoked
Notification Class Identifier	NTF-Internal	`minorId` = `SA_IMM_NTFID_OP_END`, see Section 4.2.19 on page 42
Event Time	Mandatory	Time when the runtime owner provided a response, or when the IMM Service detected an error
Number of Correlated Notifications	Mandatory	3
Correlated Notifications	Mandatory	`rootCorrelationId` and `parentCorrelationId` passed to either `saImmOmAdminOperationInvoke_3()` or to `saImmOmAdminOperationInvokeAsync_3()` and additionally the notification identifier of the corresponding `SA_NTF_ADMIN_OPERATION_START` notification
Number of Elements in Additional Information Array	Mandatory	2
Additional Information `additionalInfo`[0]	Mandatory	{`SA_IMM_AI_ADMIN_OPERATION_ID`, `SA_NTF_VALUE_UINT64`, `operationId` parameter passed to `saImmOmAdminOperationInvoke_3()` or to `saImmOmAdminOperationInvokeAsync_3()`}
Additional Information `additionalInfo`[1]	Mandatory	{`SA_IMM_AI_ADMIN_OPERATION_RESULT`, `SA_NTF_VALUE_UINT64(SaAisErrorT)`, as explained above in the description section of this notification}

8.2.2.3 Configuration Update Start

Description

The IMM Service sends the following notification when the `saImmOmCcbApply_3()` function is invoked.

Table 6 Configuration Update Start

NTF Attribute Name	Mandatory/ Optional	Specified Value
Event Type	Mandatory	`SA_NTF_CONFIG_UPDATE_START`
Notification Object	Mandatory	Empty
Notification Class Identifier	NTF-Internal	`minorId` = `SA_IMM_NTFID_CCB_APPLY_START`, see Section 4.2.19 on page 42
Event Time	Mandatory	Time when the `saImmOmCcbApply_3()` function was invoked
Number of Correlated Notifications	Mandatory	0, 1, or 2
Correlated Notifications	Mandatory	`rootCorrelationId` and `parentCorrelationId` passed to `saImmOmCcbApply_3()`
Number of Elements in Additional Information Array	Mandatory	1
Additional Information `additionalInfo`[0]	Mandatory	{`SA_IMM_AI_CCB_ID`, `SA_NTF_VALUE_UINT64`, value of the CCB identifier returned by the `saImmOmCcbInitialize_3()` function}

8.2.2.4 Configuration Update End

Description

The IMM Service sends the following notification when the saImmOmCcbApply_3() function returns.

.

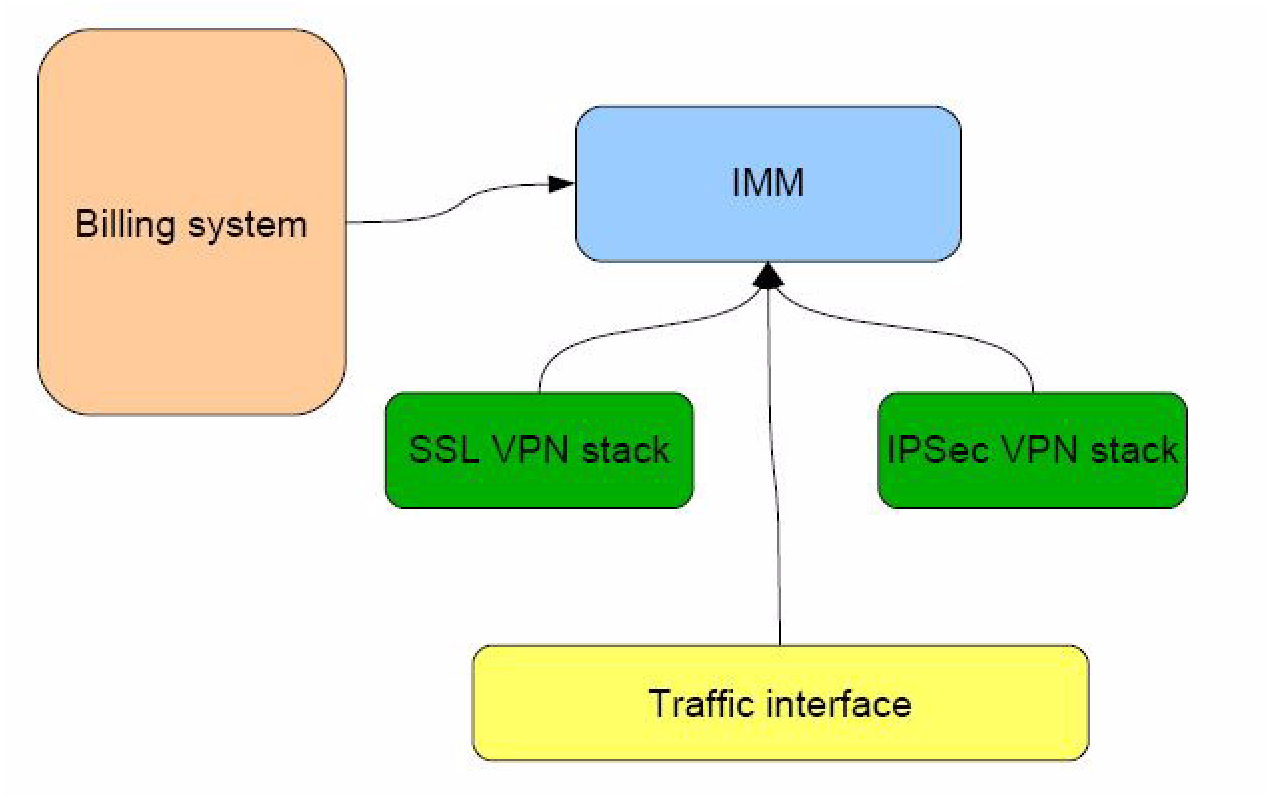
Table 7 Configuration Update End

NTF Attribute Name	Mandatory/ Optional	Specified Value
Event Type	Mandatory	SA_NTF_CONFIG_UPDATE_END
Notification Object	Mandatory	Empty
Notification Class Identifier	NTF-Internal	minorId = SA_IMM_NTFID_CCB_APPLY_END, see Section 4.2.19 on page 42
Event Time	Mandatory	Time when the saImmOmCcbApply_3() function returned
Number of Correlated Notifications	Mandatory	3
Correlated Notifications	Mandatory	rootCorrelationId and parentCorrelationId passed to saImmOmCcbApply_3() and additionally the notification identifier of the corresponding SA_NTF_CONFIG_UPDATE_START notification
Number of Elements in Additional Information Array	Mandatory	2
Additional Information additionalInfo[0]	Mandatory	{SA_IMM_AI_CCB_ID, SA_NTF_VALUE_UINT64, value of the CCB identifier returned by the saImmOmCcbInitialize_3() function}
Additional Information additionalInfo[1]	Mandatory	{SA_IMM_AI_CCB_RETURN_VALUE, SA_NTF_VALUE_UINT64(SaAisErrorT), return value of the saImmOmCcbApply_3() function}

Appendix A Example Use Case

This example shows how the IMM Service APIs can be used by an application.

FIGURE 4 Example of Using the IMM Service to Change the Configuration of a Real Application



(a) System Setup

The example consists of four Object Implementers (OIs), an SSL VPN stack, an IPSec VPN stack, a traffic interface, and a billing engine.

(b) Global Constraints

A global constraint is that a billing account must always be configured for all VPNs, and that the billing account must also be configured in the billing engine.

(c) Shared Configuration Data

The IP address of the traffic interface is a shared configuration data.
The traffic interface needs the IP address to properly configure the interface, the two VPN stacks need the IP address to bind the listening socket to the proper address.

(d) Registrations

The four OIs will register as CCB validators and CCB appliers for their corresponding objects in the IMM. The traffic interface OI must be configured with a higher rank as a CCB applier than the two VPN stacks, so that it can apply the changes ahead of them.

The billing engine will register as CCB Validator for the billing account settings in the SSL and IPSec VPN stacks. It will be invoked whenever any of those settings are created, deleted, or modified.

The SSL and IPSec VPN stacks register as CCB appliers for the IP address configuration of the traffic interface.

(e) Configuration Change: Change IP Address

Suppose that a manager decides to change the IP address of the management interface, then the following will happen:

1. The manager creates a CCB with the new IP address of the traffic interface.
2. The manager calls the apply function to indicate that no more modifications will be included in the CCB.
3. The IMM Service invokes the `saImmOiCcbValidateCallback()` callback of the traffic interface, so that the traffic interface can validate that the new IP address does not conflict with the rest of its configuration.
4. The IMM Service calls the apply function to take the new configuration into active duty. The new IP address is communicated to all CCB appliers in an order given by the rank of the CCB applier in a configured list, that is, the traffic interface, which has been configured to be notified before the VPN stacks, will be notified first, and the two VPN stacks after that. The notification ordering is important as the traffic interface needs to be updated first, because the VPN stacks cannot bind to the new address unless the interface has already been changed.
5. When called back, all CCB appliers invoke the iterator functions to obtain their new configuration. The new IP address is communicated to all CCB appliers in an order given by the rank of this CCB applier in a configured list, that is, the traffic interface, which has been configured to be notified before the VPN stacks, will be notified first, and the two VPN stacks after that.
   The CCB appliers now modify their internal states to use the new IP address. The notification ordering is important as the traffic interface needs to be updated first, because the VPN stacks cannot bind to the new address unless the interface has already been changed.

(f) Configuration Change: Create New VPN

Suppose the manager creates a new IPSec VPN. The following will happen:

1. The manager creates a new CCB with the new IPSec VPN.
2. The manager calls the apply function to indicate that no more modifications will be added to the CCB.
3. The IMM Service invokes callbacks, so that the IPSec stack can validate the whole IPSec configuration. The IMM Service also invokes the callbacks of the billing engine, so that the billing engine can validate its settings.
4. The IMM Service calls the apply function to make the new configuration the new running configuration.
5. The IPsec stack invokes the iterator functions to obtain the new configuration; the IPsec stack finally updates its internal state.

Appendix B Sequence Diagrams

The following sequence diagrams show some important scenarios. For simplicity, the suffix “Callback” and any version suffix like “_3” are not shown in the diagrams.

FIGURE 5 Successful CCB, Independent CCB Appliers and Validators

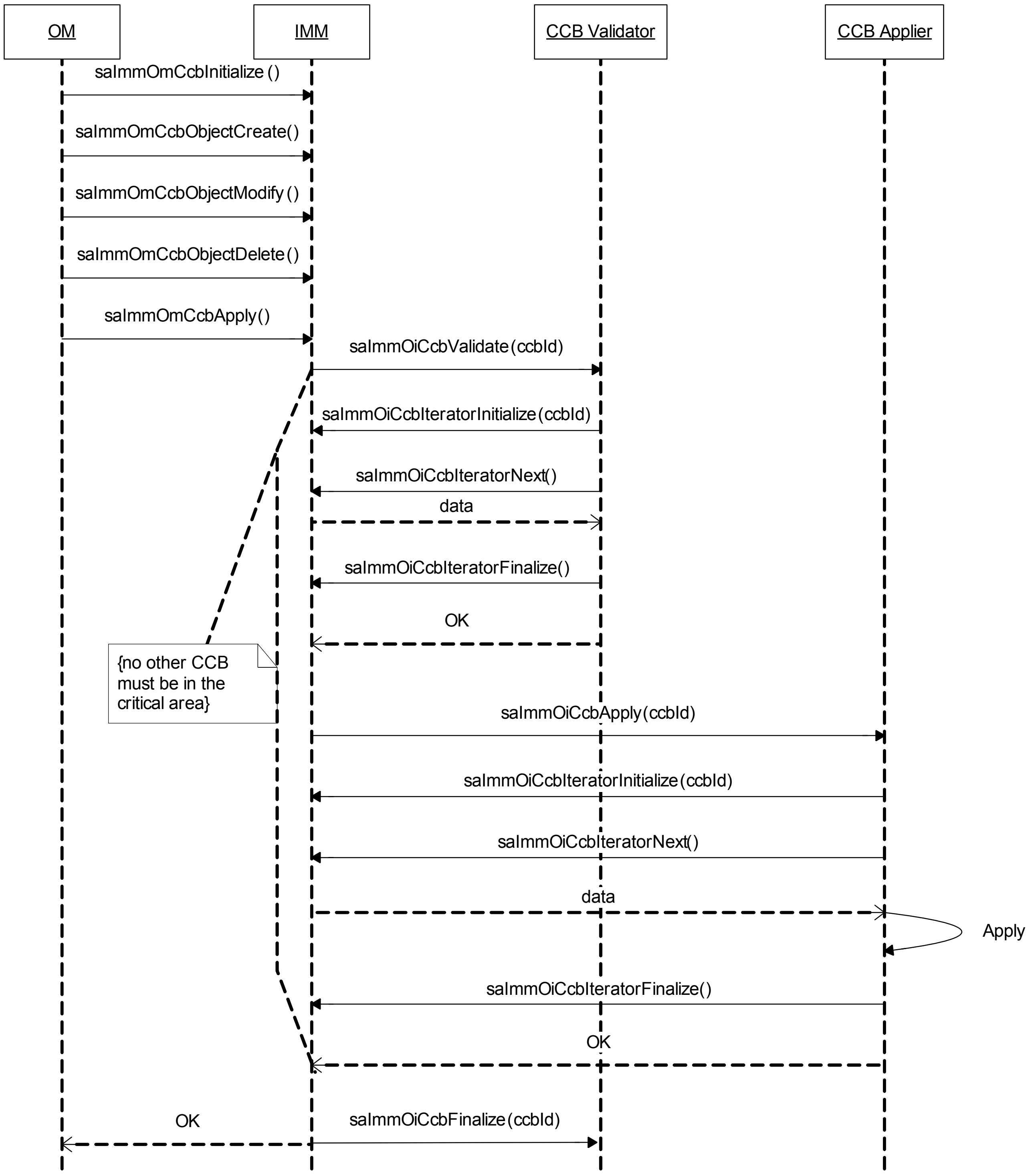

OM
IMM
CCB Validator
CCB Applier
saImmOmCcbInitialize ()
saImmOmCcbObjectCreate()
saImmOmCcbObjectModify ()
saImmOmCcbObjectDelete ()
saImmOmCcbApply()
saImmOiCcbValidate(ccbId)
saImmOiCcbIteratorInitialize(ccbId)
saImmOiCcbIteratorNext()
data
saImmOiCcbIteratorFinalize()
OK
{no other CCB must be in the critical area}
saImmOiCcbApply(ccbId)
saImmOiCcbIteratorInitialize(ccbId)
saImmOiCcbIteratorNext()
data
Apply
saImmOiCcbIteratorFinalize()
OK
OK
saImmOiCcbFinalize(ccbId)

FIGURE 6 Failed CCB, Independent CCB Appliers and Validators

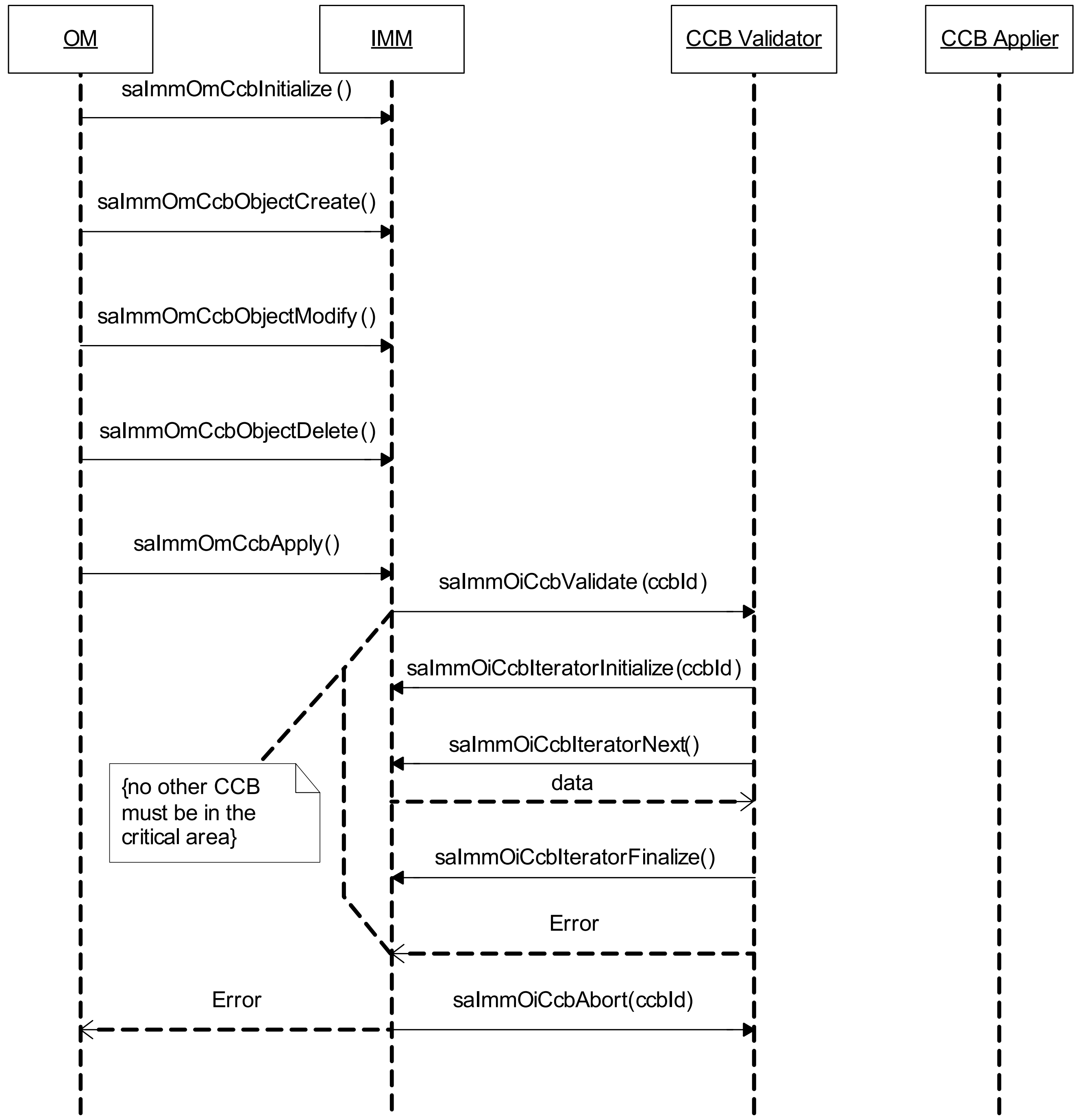

OM
IMM
CCB Validator
CCB Applier
saImmOmCcbInitialize ()
saImmOmCcbObjectCreate()
saImmOmCcbObjectModify ()
saImmOmCcbObjectDelete ()
saImmOmCcbApply()
saImmOiCcbValidate (ccbId)
saImmOiCcbIteratorInitialize (ccbId)
saImmOiCcbIteratorNext()
data
saImmOiCcbIteratorFinalize()
Error
{no other CCB must be in the critical area}
Error
saImmOiCcbAbort(ccbId)

FIGURE 7 Successful CCB, OI is both CCB Validator and CCB Applier

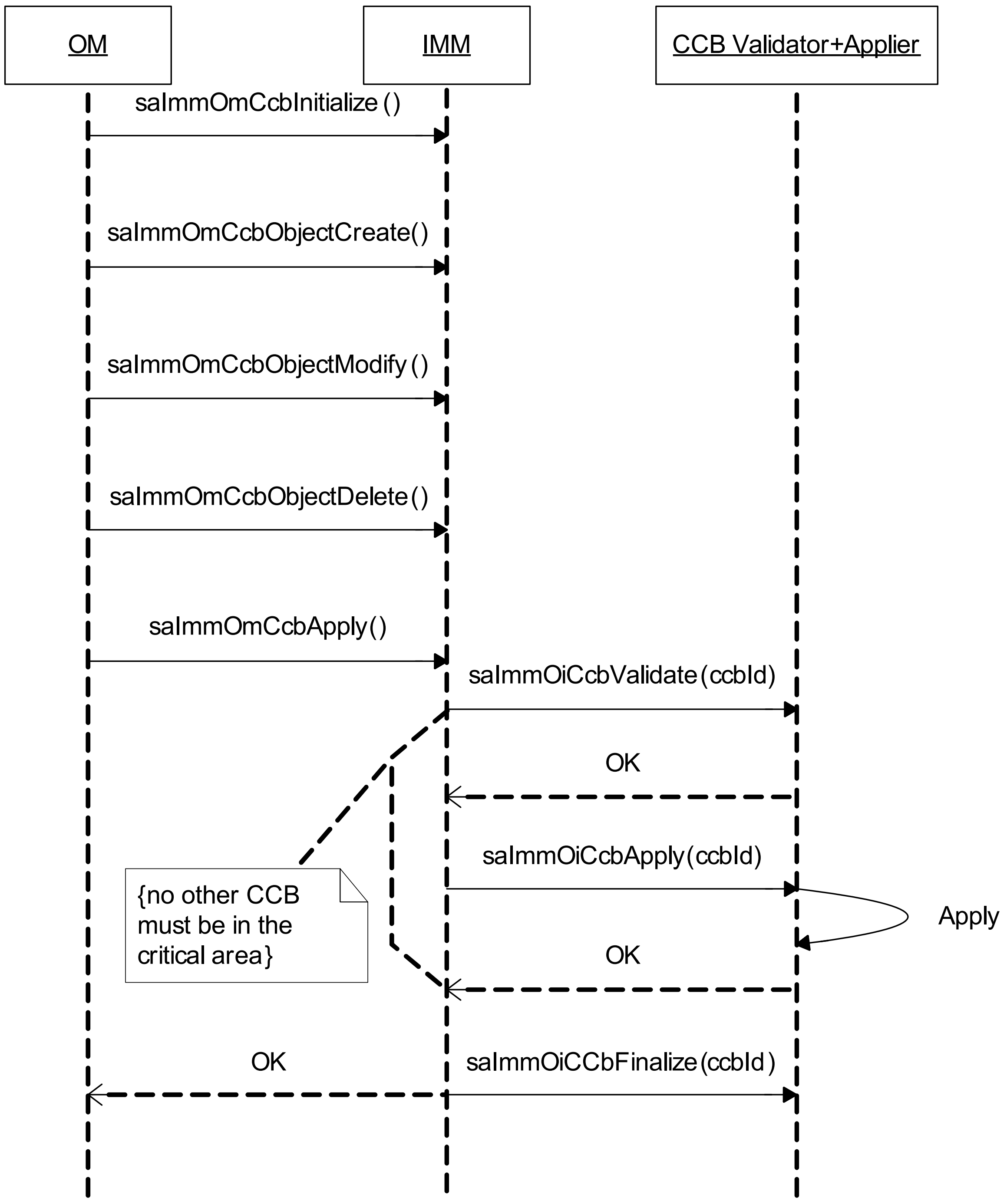
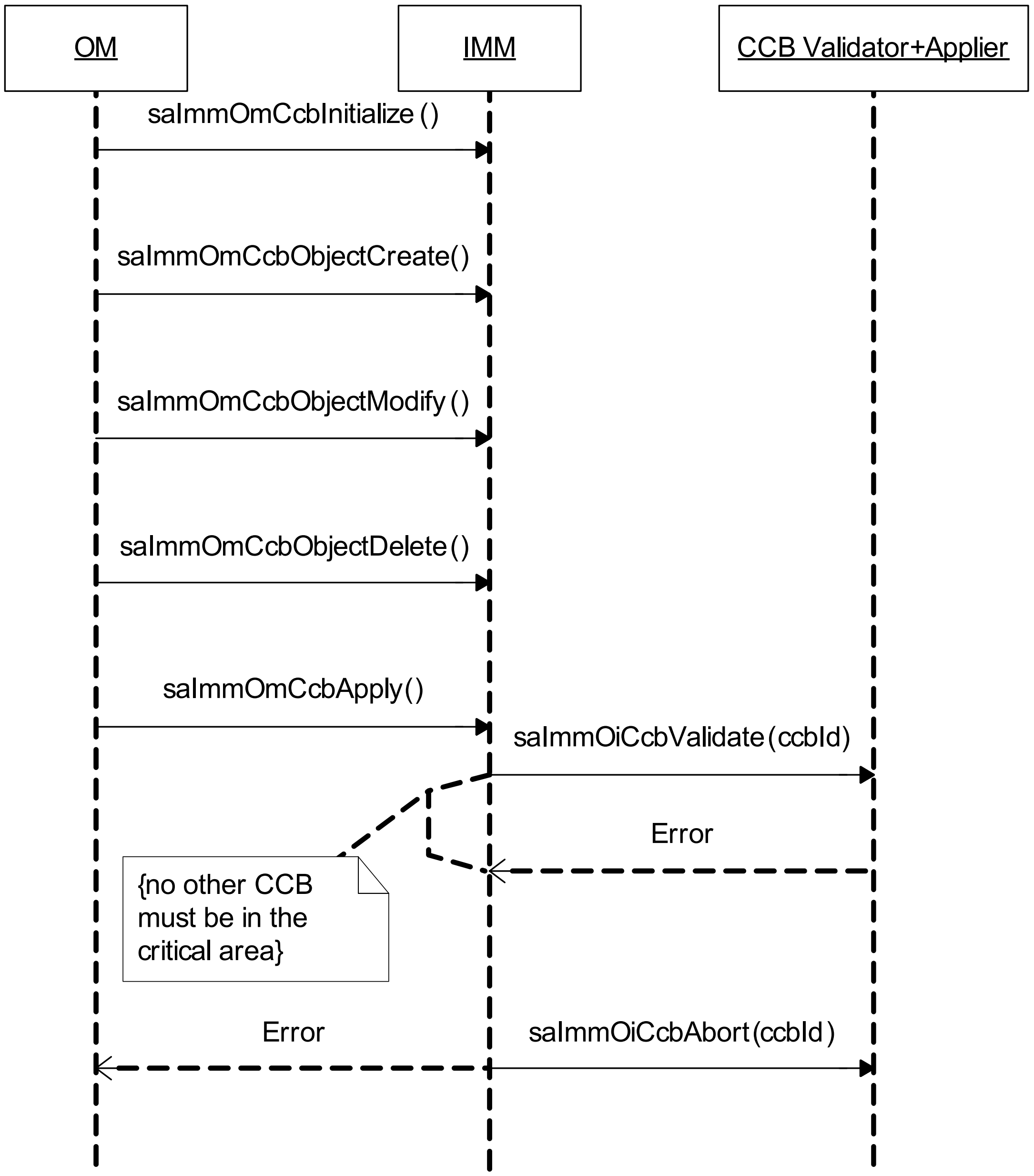

OM
IMM
CCB Validator+Applier
saImmOmCcbInitialize ()
saImmOmCcbObjectCreate()
saImmOmCcbObjectModify ()
saImmOmCcbObjectDelete ()
saImmOmCcbApply()
saImmOiCcbValidate (ccbId)
OK
saImmOiCcbApply(ccbId)
Apply
OK
{no other CCB must be in the critical area}
OK
saImmOiCCbFinalize (ccbId)

FIGURE 8 Failed CCB, OI is both CCB Validator and CCB Applier



Appendix C Compatibility Issues

An IMM Service implementation that is capable of supporting multiple versions of the IMM Service API specifications concurrently must handle the case that a CCB will affect object implementers that were written against different versions of the APIs. This section describes the considerations that an object manager needs to take into account in such a case and also presents an example to illustrate the collaboration between these entities of different versions.

Up to the A.02.01 version of the IMM Service specification, each change request issued by an object manager was immediately propagated by the IMM Service to the single object implementer of the targeted object. Thus, change requests could be verified at this early stage and rejected by an object implementer due to any reason, including the ordering of the change requests. The current specification removes the requirement of propagating each change requests to the object implementers; instead, it asks for validation only when the object manager has completed the CCB and invokes the `saImmOmApply_3()` function. As a consequence, the object implementers receive for verification only the proposed final state of the SA Forum Information Model, and they are unaware how this state was reached; however, an IMM Service implementation supporting earlier versions and the current versions of the API, will act appropriately toward each object implementer.

C.1 Object Manager Considerations

Object manager applications implemented against an earlier version of the IMM Service API face no problem in collaborating with implementations written against the current specification. Any CCB that would succeed according to earlier versions shall succeed according to the new version.

The setting of the `SA_IMM_CCB_REGISTERED_OI` flag is interpreted by the IMM Service in the following way:

- If this flag is set by the object implementer initializing the CCB, it is interpreted as if none of the `SA_IMM_CCB_ALLOW_ABSENT_VALIDATORS` and `SA_IMM_CCB_ALLOW_ABSENT_APPLIERS` flags were set.
- If the `SA_IMM_CCB_REGISTERED_OI` flag is not set, it is mapped as if both the `SA_IMM_CCB_ALLOW_ABSENT_VALIDATORS` and `SA_IMM_CCB_ALLOW_ABSENT_APPLIERS` flags were set.

Object manager applications implemented against the current specification need to consider object implementers that were written against an earlier version of the specification, as these object implementers will receive and therefore verify the change requests in the order the object manager issues them. This may have the undesirable effect that these object implementers refuse change requests even before the `saImmOmApply_3()` function is invoked.

Thus, to successfully apply the CCB in this case, the object manager needs to issue the change requests pertaining to the CCB in a way that is compliant to earlier versions of the specification.

The setting of the `SA_IMM_CCB_ALLOW_ABSENT_VALIDATORS` and `SA_IMM_CCB_ALLOW_ABSENT_APPLIERS` flags are propagated by the IMM Service to the object implementers of earlier versions as follows:

- If one or both of these flags are set, this setting is propagated as if `SA_IMM_CCB_REGISTERED_OI` flag were not set;
- If neither of these flags are set then, this setting is propagated as if the `SA_IMM_CCB_REGISTERED_OI` flag were set.

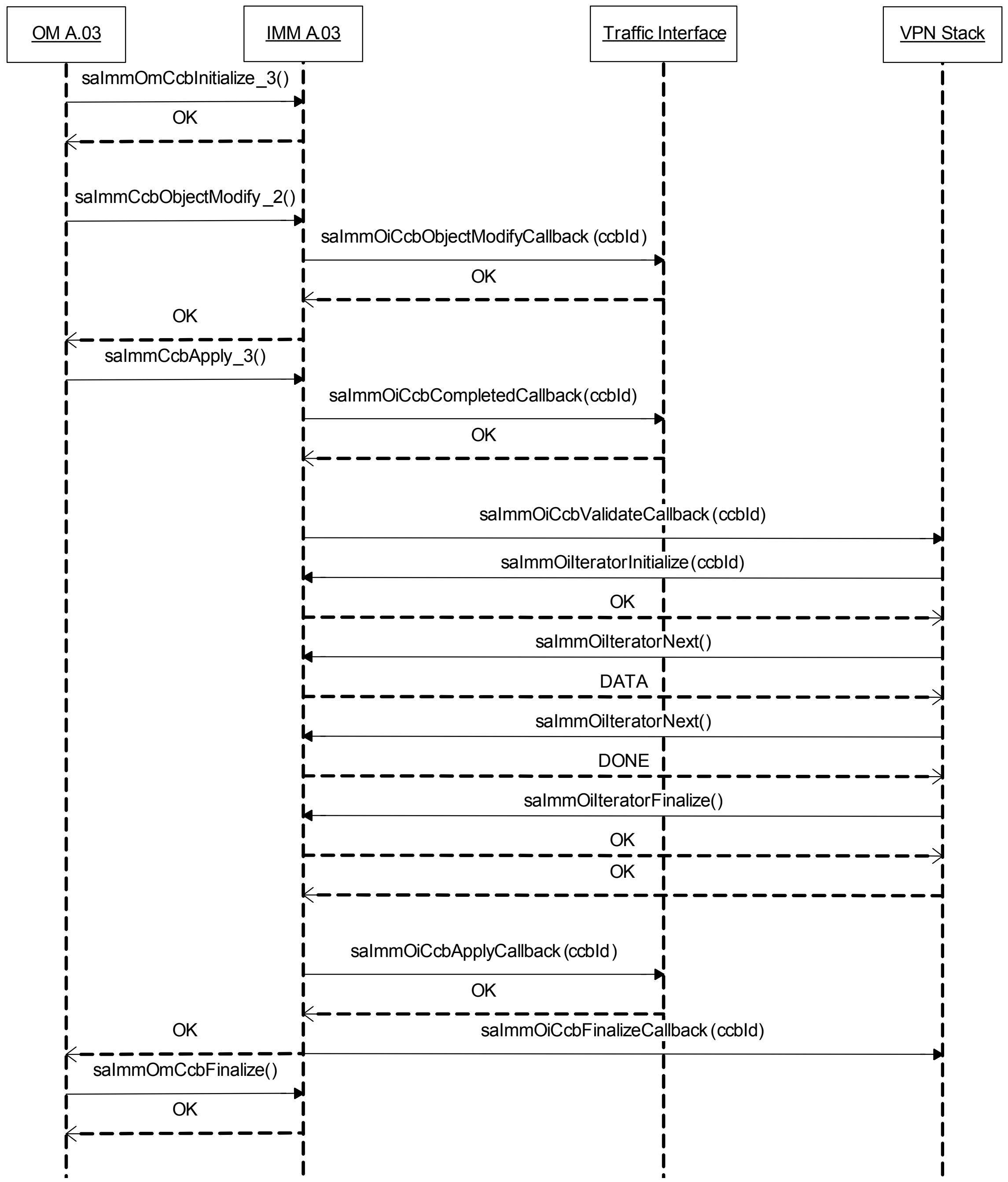
C.2 Example Using Object Implementers of Different Versions

Assume a setup similar to the one described in Appendix A and with the following additional versioning information:

- the object manager (OM) uses the A.03.01 version of the IMM API;
- the object implementer implementing the traffic interface uses the A.02.01 version of the IMM API;
- an object implementer using the A.03.01 version of the IMM API acts as both
  - runtime owner and CCB applier of the VPN stack (not used in this example) and
  - CCB validator for the traffic interface and the VPN stack.

This CCB validator needs to validate that the DNS name in the certificate used by the VPN stack must match the IP address of the traffic interface. When performing a configuration change of the traffic interface, the sequence of invocations presented in FIGURE 9 shall occur.

FIGURE 9 IMM Service Mediating Between Object Implementers of Different Versions


OM A.03
IMM A.03
Traffic Interface
VPN Stack
saImmOmCcbInitialize_3()
OK
saImmCcbObjectModify_2()
saImmOiCcbObjectModifyCallback (ccbId)
OK
OK
saImmCcbApply_3()
saImmOiCcbCompletedCallback(ccbId)
OK
saImmOiCcbValidateCallback (ccbId)
saImmOiIteratorInitialize(ccbId)
OK
saImmOiIteratorNext()
DATA
saImmOiIteratorNext()
DONE
saImmOiIteratorFinalize()
OK
OK
saImmOiCcbApplyCallback (ccbId)
OK
OK
saImmOiCcbFinalizeCallback (ccbId)
saImmOmCcbFinalize()
OK

Index of Definitions