Service Availability™ Forum
Service Availability Interface

C Programming Model
SAI-AIS-CPROG-B.05.02

.

SERVICE AVAILABILITY™ FORUM SPECIFICATION LICENSE AGREEMENT

The Service Availability™ Forum Application Interface Specification (the "Package") found at the URL http://www.saforum.org is generally made available by the Service Availability Forum (the "Copyright Holder") for use in developing products that are compatible with the standards provided in the Specification. The terms and conditions which govern the use of the Package are covered by the Artistic License 2.0 of the Perl Foundation, which is reproduced here.

The Artistic License 2.0

Copyright (c) 2000-2006, The Perl Foundation.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

This license establishes the terms under which a given free software Package may be copied, modified, distributed, and/or redistributed.

The intent is that the Copyright Holder maintains some artistic control over the development of that Package while still keeping the Package available as open source and free software.

You are always permitted to make arrangements wholly outside of this license directly with the Copyright Holder of a given Package. If the terms of this license do not permit the full use that you propose to make of the Package, you should contact the Copyright Holder and seek a different licensing arrangement.

Definitions

"Copyright Holder" means the individual(s) or organization(s) named in the copyright notice for the entire Package.

"Contributor" means any party that has contributed code or other material to the Package, in accordance with the Copyright Holder's procedures.

"You" and "your" means any person who would like to copy, distribute, or modify the Package.

"Package" means the collection of files distributed by the Copyright Holder, and derivatives of that collection and/or of those files. A given Package may consist of either the Standard Version, or a Modified Version.

"Distribute" means providing a copy of the Package or making it accessible to anyone else, or in the case of a company or organization, to others outside of your company or organization.

"Distributor Fee" means any fee that you charge for Distributing this Package or providing support for this Package to another party. It does not mean licensing fees.

"Standard Version" refers to the Package if it has not been modified, or has been modified only in ways explicitly requested by the Copyright Holder.

"Modified Version" means the Package, if it has been changed, and such changes were not explicitly requested by the Copyright Holder.

"Original License" means this Artistic License as Distributed with the Standard Version of the Package, in its current version or as it may be modified by The Perl Foundation in the future.

"Source" form means the source code, documentation source, and configuration files for the Package.

"Compiled" form means the compiled bytecode, object code, binary, or any other form resulting from mechanical transformation or translation of the Source form.

Permission for Use and Modification Without Distribution

(1) You are permitted to use the Standard Version and create and use Modified Versions for any purpose without restriction, provided that you do not Distribute the Modified Version.

Permissions for Redistribution of the Standard Version

(2) You may Distribute verbatim copies of the Source form of the Standard Version of this Package in any medium without restriction, either gratis or for a Distributor Fee, provided that you duplicate all of the original copyright notices and associated disclaimers. At your discretion, such verbatim copies may or may not include a Compiled form of the Package.

(3) You may apply any bug fixes, portability changes, and other modifications made available from the Copyright Holder. The resulting Package will still be considered the Standard Version, and as such will be subject to the Original License.

Distribution of Modified Versions of the Package as Source

(4) You may Distribute your Modified Version as Source (either gratis or for a Distributor Fee, and with or without a Compiled form of the Modified Version) provided that you clearly document how it differs from the Standard Version, including, but not limited to, documenting any non-standard features, executables, or modules, and provided that you do at least ONE of the following:

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

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

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

(i) the Original License or

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

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

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

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

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

Aggregating or Linking the Package

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

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

Items That are Not Considered Part of a Modified Version

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

General Provisions

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

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

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

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

(14) Disclaimer of Warranty:

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

Table of Contents C Programming Model

List of Figures

1 Document Introduction

1.1 Document Purpose

This document (**SAI-AIS-CPROG-B.05.02**) describes the programming model of the Service Availability^{TM} Forum (SA Forum) AIS specifications that define APIs in the C language. It also provides all type definitions that are common to these specifications.

Note: For the convenience of the user, the SA Forum also provides C header files for all AIS specifications. The corresponding archive can be downloaded from http://www.saforum.org.

1.2 History

In releases of the SA Forum documents prior to Release 6, the contents of this document were part of the Overview document. With the introduction of the Java mapping specifications, all sections of the Overview document that are specific to the C language have been moved to this new document.

The only previous release of this document in this new form was:

SAI-AIS-CPROG-B.05.01

1.2.1 Changes from SAI-AIS-CPROG-B.05.01 to SAI-AIS-CPROG-B.05.02

1.2.1.1 Clarifications

- A clarification has been added to Section 2.3.9 on the `SaStringT` type.
- Section 2.3.9 on `SaNameT` and its subsections have been thoroughly revised. The usage and formats of DNs and RDNs have been clarified.

1.2.2 Changes from SAI-AIS-Overview-B.04.02 to SAI-AIS-CPROG-B.05.01

Only the changes in sections of SAI-AIS-Overview-B.04.02 that have been integrated into SAI-AIS-CPROG-B.05.01 are considered in this section.

1.2.2.1 New Topics

- The enhanced track API has been introduced in Section 2.1.5.6 to support the track API of the Platform Management Service ([11]) and the enhanced track API of the Cluster Membership Service ([5]). New track flags and additional usage notes are provided in Section 2.3.12. This enhancement induced also modifications to Section 2.1.5 and to its subsections, including the introduction of the new Section 2.1.5.3. In Section 2.1.5.1, it is explained how a tracking that is in effect is affected by another call of the function to start tracking.

- Section 2.2 presents a new application interface area tag for the Platform Management Service.

1.2.2.2 Clarifications

Section 2.3.3 defines the floating point types based on the IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754, see [17]).

1.2.2.3 Other Changes

- Section 2.3.9.2.1 introduces values for the `safApp` RDN of the Hardware Platform Interface ([12]) and of the Platform Management Service ([11]).
- Section 2.3.10 on `SaServicesT` introduces a new value for the Platform Management Service ([11]).
- Section 2.3.17 contains the new error codes `SA_AIS_ERR_NOT_READY` for the Availability Management Framework ([4]) and `SA_AIS_ERR_DEPLOYMENT`.

1.2.2.4 Removed Topics

The section on the naming convention for global variables in Chapter 2 of the preceding version of the Overview document has been removed, as the SA Forum does not define any global variables.

1.3 References

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

[1] Service Availability^{TM} Forum, Service Availability Interface, Overview, SAI-Overview-B.05.02

[2] Service Availability^{TM} Forum, Application Interface Specification, Notification Service, SAI-AIS-NTF-A.03.01

[3] Service Availability^{TM} Forum, Application Interface Specification, Information Model Management Service, SAI-AIS-IMM-A.03.01

[4] Service Availability^{TM} Forum, Application Interface Specification, Availability Management Framework, SAI-AIS-AMF-B.04.01

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

[6] Service Availability^{TM} Forum, Application Interface Specification, Checkpoint Service, SAI-AIS-CKPT-B.02.02

[7] Service Availability^{TM} Forum, Application Interface Specification, Message Service, SAI-AIS-MSG-B.03.01

[8] Service Availability™ Forum, Application Interface Specification, Naming Service, SAI-AIS-NAM-A.01.01

[9] Service Availability™ Forum, Application Interface Specification, Software Management Framework, SAI-AIS-SMF-A.01.02

[10] Service Availability™ Forum, Application Interface Specification, Security Service, SAI-AIS-SEC-A.01.01

[11] Service Availability™ Forum, Application Interface Specification, Platform Management Service, SAI-AIS-PLM-A.01.02

[12] Service Availability™ Forum, Hardware Platform Interface Specification, SAI-HPI-B.03.02

[13] Service Availability™ Forum, HPI C Header Files for Release 6, SAI-HPI-CH-B.03.02.zip

[14] Service Availability™ Forum, AIS C Header Files for Release 6, SAI-AIS-R6-CH-A.01.02

[15] IETF RFC 2253 (http://www.ietf.org/rfc/rfc2253.txt)

[16] Unicode Standard (http://www.unicode.org)

[17] ANSI/IEEE Standard 754-1985, Standard for Binary Floating Point Arithmetic

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

1.4 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.

1.5 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.

1.6 Additional Information

1.6.1 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).

1.6.2 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 Programming Model and Naming Conventions

This chapter describes the programming model and naming conventions used by the SA Forum Application Interface Specification (AIS) to define APIs in the C language. This chapter contains the following features:

- Discussion of asynchronous and synchronous APIs (see Section 2.1.1).
- Discussion of APIs for using a library of the Application Interface Specification (Library Life Cycle, see Section 2.1.2). Section 2.1.2.4 explains the usage of hidden threads in AIS Service libraries.
- Interaction between POSIX and AIS APIs (see Section 2.1.3).
- Memory management rules (see Section 2.1.4).
- Usage of track APIs (see Section 2.1.5).
- Description of interfaces for retrieving values of service-specific limits of an implementation (see Section 2.1.6).
- Discussion of the various conditions that cause an AIS Service to be unavailable within the scope of a node along with the behavior of the AIS Service API functions under these conditions (see Section 2.1.7).
- Rules for forming names of types, functions, and macro declarations (see Section 2.2).
- Definitions of the predefined types and constants (see Section 2.3), which support application portability between platforms and implementations. Section 2.4 explains how the SA Forum handles backward compatibility.

2.1 Programming Model and Usage Overview

This section provides an overview of the SA Forum Application Interface programming model and the generally intended usage of the SA Forum Application Interfaces. The descriptions contained herein are not intended to constrain implementations unduly.

The SA Forum Application Interface occurs between a process and a library that implements the interface. The interface is designed for use by both threaded and non-threaded application processes.

The term **process**—as used in this document—can be regarded as being equivalent to a process defined by the POSIX standard. However, the use of the term process does not mandate a POSIX process but, rather, any equivalent entity that a system provides to manage executing software.

The **area server** is an abstraction that represents the server that provides services for a specification area (Availability Management Framework, Cluster Membership Service, Checkpoint Service, and so on). Each **area** has a separate logical area server, although the implementer is free to create a separate physical module for each area server or combine one or more area servers into a single physical module.

The area implementation libraries may be implemented in one or several physical libraries; however, a process is required to initialize, register, and obtain an operating system **selection object** separately for each area's implementation library. Thus, from a programming standpoint, it is useful to view these as separate libraries.

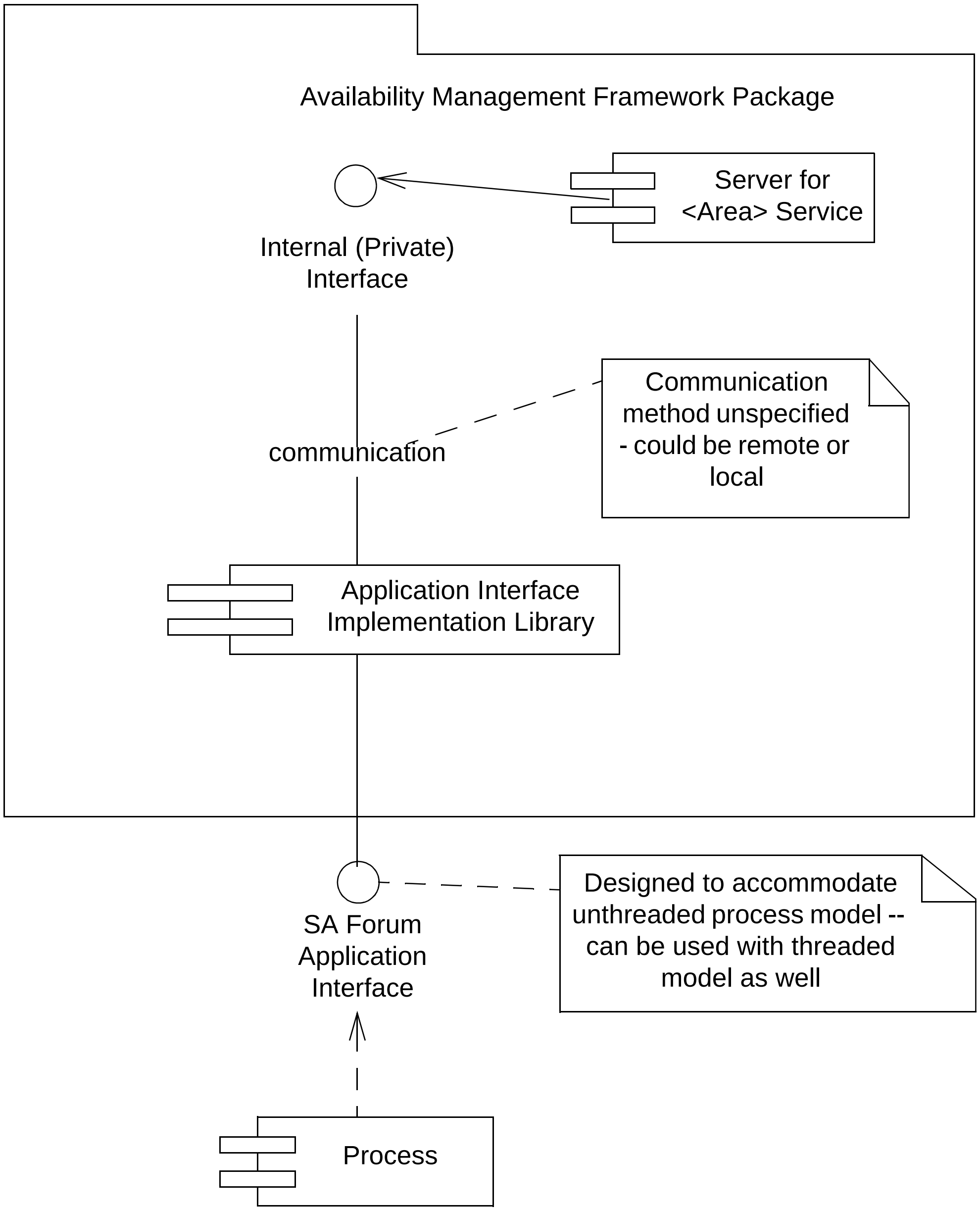
The UML diagram in FIGURE 1 shows the relationships among an area server, an area implementation library, and a process, all represented as UML components.

Although FIGURE 1 shows only one area server, one area implementation library, and one application component, nothing restricts an area server from interfacing with numerous area implementation libraries and an area implementation library from servicing multiple application components. If a component comprises multiple processes, each process must initialize the instances of the area implementation libraries that it uses.

Note: For those readers who are unfamiliar with UML, the boxes with two rectangles on the left are UML "components" (not to be confused with components in the context of the SA Forum Application Interface Specification), the box with a "tab" at the top is a package, and the two circles are interfaces. The dashed lines to the interfaces are dependency or "consumes" relationships, and the solid lines to the interfaces are "realizes" or "provides" relationships. Thus, the process connected to the interface by the dashed line is an interface consumer, whereas that connected by the solid line is an interface provider. As shown in FIGURE 1, the area server and the area implementation library are packaged together.

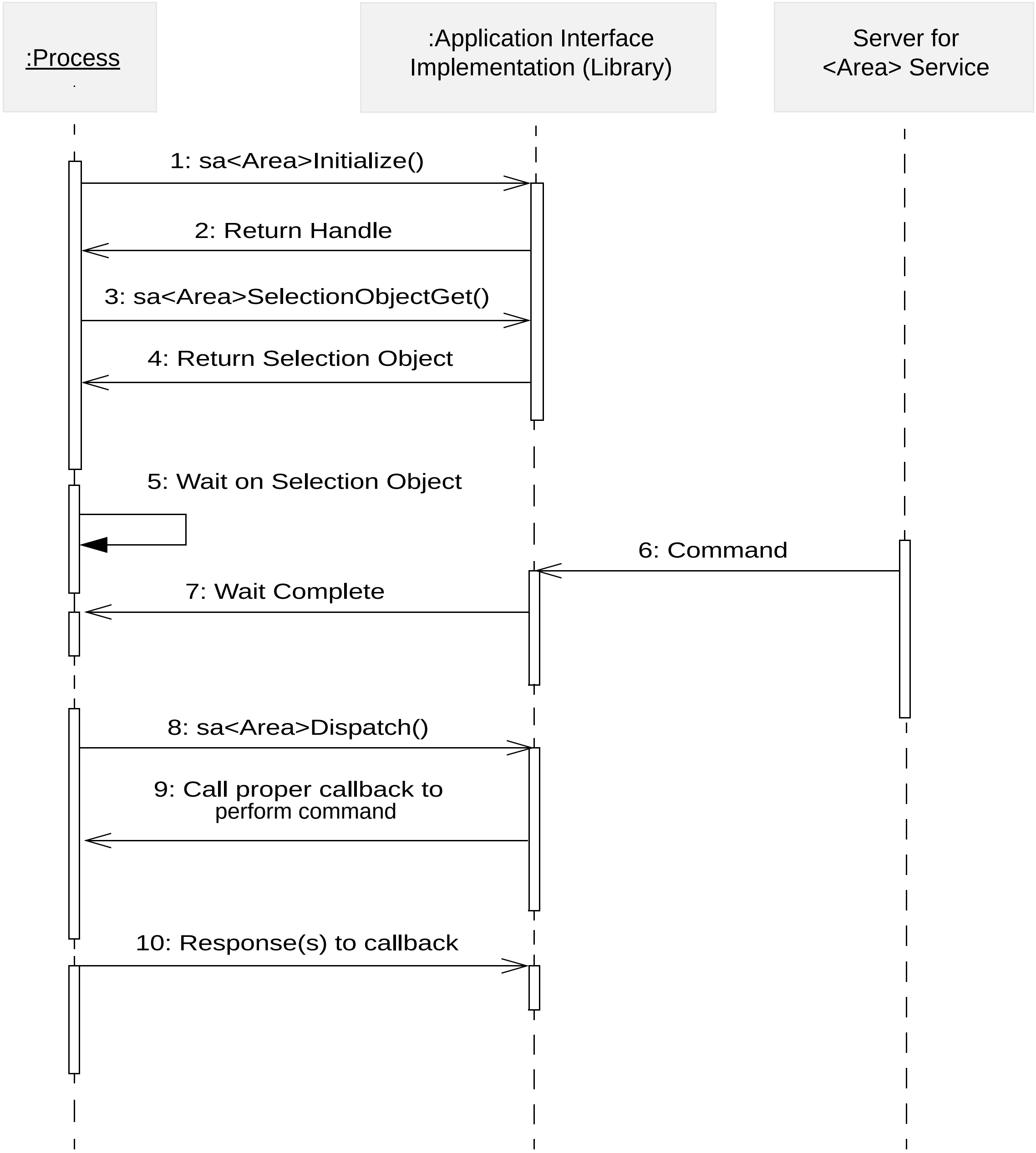
It is expected that the area server and the area implementation library be packaged together and be designed to be released as a set. However, this does not preclude providing other packaging options.
The interface between the area server and the area implementation library is proprietary and outside the scope of this specification. The area server and the area implementation library could reside on the same or separate computers, and perhaps even within the same software module.

FIGURE 1 Interface Relationships


Availability Management Framework Package
Server for <Area> Service
Internal (Private) Interface
communication
Communication method unspecified - could be remote or local
Application Interface Implementation Library
SA Forum Application Interface
Designed to accommodate unthreaded process model -- can be used with threaded model as well
Process

The SA Forum Application Interface Specification programming/usage model views the area server as a server for the component and the component as a client of the area server. In this sense, the usage model is typical of an event-driven architecture, in which the application performs a setup and then receives callbacks as events occur.

FIGURE 2 **Programming/Usage Model**


:Process
:Application Interface Implementation (Library)
Server for <Area> Service
1: sa<Area>Initialize()
2: Return Handle
3: sa<Area>SelectionObjectGet()
4: Return Selection Object
5: Wait on Selection Object
6: Command
7: Wait Complete
8: sa<Area>Dispatch()
9: Call proper callback to perform command
10: Response(s) to callback

The programming/usage model is shown in FIGURE 2. Again, this model is intended to show the usage for which the interfaces were generally intended, rather than unduly constraining implementations. For example, it is possible that actions 6 and 7 of the model might be combined, or the library might obtain the command from the area server between actions 8 and 9.

The following example of APIs shows the callback mechanism.

1. The process within the component invokes the `sa<Area>Initialize()` function to initialize the AIS `<Area>` Service library and to provide a set of callbacks for use by the library in calling back the process.
2. The `sa<Area>Initialize()` function returns an interface handle to the invoking process.
3. The process invokes the `sa<Area>SelectionObjectGet()` function to obtain a selection object, which is an operating-system-dependent object (for instance, a file descriptor suitable for use in `select()` for Unix/Linux).
4. The interface returns a selection object to the process. This operating system-dependent object allows the process to wait until an invocation to a callback function is pending for it.
5. The process waits on the selection object.
6. The area server sends a command over its private interface to the library.
7. The library "awakes" the selection object, thereby awaking the process.
8. The process invokes the `sa<Area>Dispatch()` function.
9. The library invokes the appropriate callback function of the process corresponding to the command received from the area server. The callback function parameters inform the process of the specific details of the command issued by the area server or the information provided by the area server.
10. Once the process completes processing the callback, it responds by invoking a function of the area interface. In some cases, more than one response invocation (or no response) may be necessary.

In addition to the callback mechanism, certain functions that the component may invoke are asynchronous, for example, functions for obtaining information from the area server by using the library or for reporting errors.

2.1.1 Synchronous and Asynchronous Programming Models

The Application Interface Specification employs both the synchronous and asynchronous programming models. The **synchronous programming model** is generally easier for programmers to understand and use. However, a large number of simultaneous outstanding requests may preclude having an independent thread of execution for each request. Some applications also require direct control of scheduling within a process. To support such applications, asynchronous APIs are used in the core of the service availability components.
AIS defines synchronous and asynchronous variants of open calls (`sa<Area>XxxOpen()` and `sa<Area>XxxOpenAsync()`), as it is expected that these operations are cluster-wide operations needing some time to complete. In contrast, only synchronous close calls (`sa<Area>XxxClose()`) are specified, as it is expected that these calls return as soon as possible to the caller and that the remaining processing is done asynchronously.

Synchronous APIs are generally used for library and association housekeeping interfaces.

Note: Some of the examples in this section contain POSIX operating system-specific constructs. The examples are given for illustrative purposes only and do not imply that POSIX-specific constructs are necessary to use a particular programming model.

2.1.1.1 Asynchronous APIs

Functions that are called by an application process and that solicit an asynchronous response from the area server, for instance, those with an `Async` suffix, generally have as the first two parameters `<area>Handle` and `invocation`. The `<area>Handle` is the **handle** that was provided by the `sa<Area>` library when the process invoked the `sa<Area>Initialize()` function. This allows the `sa<Area>` library to invoke the response callback function by using the correct selection object in a multithreaded process.

The process allocates and sets `invocation` for the call and uses `invocation` subsequently to distinguish the corresponding response invocation. Typically, response invocations have `invocation` as the first parameter.

If the API implementation does not invoke the callback function, for whatever reason, the process receives no other indication of the completion or success of the asynchronous function that it invoked.

Typically, the choice is left to the implementation whether errors are detected in the library and returned by the asynchronous API, or whether errors are detected by the area server and returned subsequently by the callback. In order to allow this flexibility, some error codes are listed as returned values of the asynchronous API as well as errors returned by the callback.

If an error is detected directly by the asynchronous API (which typically means that the return value from the API is different from SA_AIS_OK), the request for the corresponding asynchronous operation is implicitly canceled and no callback is invoked subsequently for this operation.

Example

An asynchronous function declaration:

```
SaAisErrorT saClmClusterNodeGetAsync(
     SaClmHandleT clmHandle,
     SaInvocationT invocation,
     SaClmClusterNodeIdT nodeId,
     SaClmClusterNodeT *clusterNode
);
```

The corresponding response declaration:

```
typedef void (*SaClmClusterNodeGetCallbackT)(
     SaInvocationT invocation,
     const SaClusterNodeInfoT *clusterNode,
     SaAisErrorT error
);
```

2.1.1.2 Synchronous APIs

Two types of synchronous APIs do not need any particular consideration:

1. A synchronous API that does not require a context switch, that is, it can be completed by local processing within the library.

2. A synchronous API that will not, or may not, be called from a function with bounded time constraints.

Other APIs and, in particular, the synchronous counterparts of asynchronous APIs provide a timeout parameter to control the blocking behavior of the call.

Example

```
SaAisErrorT error;
SaClmClusterNodeT clusterNode;
SaClmNodeIdT nodeId;
SaTimeT timeout; /* timeout value for synchronous invocations */
...
timeout = 100 * SA_TIME_ONE_MILLISECOND; /* 100 milliseconds */
nodeId = 10;
error = saClmClusterNodeGet(clmHandle, nodeId, timeout,
     &clusterNode);
if (error == SA_AIS_ERR_TIMEOUT) { /* process the error*/ }
```

2.1.2 Library Life Cycle

2.1.2.1 Initialization

The use of a Service Availability library starts with a call to initialize the library. This **area initialization** call potentially loads any dynamic code and binds the asynchronous calls implemented by the process.

Prototype

```
SaAisErrorT sa<Area>Initialize(
     Sa<Area>HandleT *<area>Handle,
     const Sa<Area>CallbacksT *<area>Callbacks,
     const SaVersionT *version
);
```

The <area>Handle parameter points to a handle that represents the association of the library initialization. This handle is returned by the library and used in subsequent calls. AIS libraries must support several invocations of sa<Area>Initialize() issued from the same binary program (for instance, process in the POSIX.1 world). Each call to sa<Area>Initialize() returns a different handle. The process can obtain a separate selectionObject for each handle, thereby allowing support for multithreaded dispatching of <area> callbacks.

When a process invokes an asynchronous function of the <area> library, the <area>Handle, cited as a parameter of that function, can determine the selection object that the library uses for the asynchronous response callback.

The <area>Callbacks parameter points to a structure of pointers to the functions implemented by the process and that the <area> library can invoke. If the process does not implement any callback functions, it must invoke sa<Area>Initialize() and specify a NULL <area>Callbacks parameter. The process must also set individual members of Sa<Area>CallbacksT to NULL if these particular callbacks are not to be used by the particular initialization and must not be invoked by the <area> library.

Prototype

```
typedef void (*SaComponent<Object><Action>T)(...);
```

Example

```
typedef void (*SaClmClusterNodeGetCallbackT)(
     SaInvocationT invocation,
     SaClmClusterNodeT *clusterNode,
     SaAisErrorT error,
);
```

Prototype of the structure containing callback functions provided by the process:

```
typedef struct {
Sa<Area><Object><Action-1>CallbackT
     sa<Area><Object><Action-1>Callback;
Sa<Area><Object><Action-2>CallbackT
     sa<Area><Object><Action-2>Callback;
...
Sa<Area><Object><Action-N>CallbackT
     sa<Area><Object><Action-N>Callback;
} Sa<Area>CallbacksT;
```

Any API calls, including the `sa<Area>Dispatch()` call (refer to Section 2.1.2.3 on page 21), can be called from any callback function.

If the invoking process exits after having successfully returned from the `sa<Area>Initialize()` function, and before it invokes `sa<Area>Finalize()` to finalize the handle `<area>Handle` (see Section 2.1.2.2 on page 21), the `<Area>` Service automatically finalizes this handle and any other handles obtained with the handle `<area>Handle` when the death of the process is detected.

As an input parameter of the `sa<Area>Initialize()` function, the structure pointed to by `version` indicates the version of the particular AIS Service that the process requires. This parameter can be used by library implementers to provide support for different API versions in a single library. As an output parameter, the version actually supported by the particular AIS Service is delivered. For more details on versioning, refer to the description of the `SaVersionT` type in Section 2.3.11 on page 53.

2.1.2.2 Finalization

When the process no longer requires the use of the area functions, it calls the **area finalization** function. The exact semantics of finalization is area-dependent concerning termination of outstanding requests; however, the intention is to disassociate the process from the interface area implementation instance and recover any associated resources. If a process has invoked `sa<Area>Initialize()` multiple times to obtain multiple `<area>Handles`, it must invoke `sa<Area>Finalize()` separately for each such handle.

Prototype

```
SaAisErrorT sa<Area>Finalize(Sa<Area>HandleT <area>Handle);
```

where the value of the `<area>Handles` parameter is the value of the handle returned by the corresponding prior invocation of the initialization function.

2.1.2.3 Dispatching

In the synchronous model, the dispatching of Service Availability interface area library calls is done when the process invokes an API function of the area. This interaction may depend on some IPC or synchronization primitives that might be blocking. If synchronous versions of the APIs are used in a non-threaded environment, polling by repeatedly invoking the call with a small timeout value might be required to service multiple requests simultaneously.

Dispatching in the asynchronous model is supported by obtaining an operating system handle that allows the process to ascertain whether any calls are pending. The generic call to obtain the operating system handle is as follows:

```
SaAisErrorT sa<Area>SelectionObjectGet(
    Sa<Area>HandleT <area>Handle,
    SaSelectionObjectT *selectionObject
);
```

In the POSIX.1 world, the selection object is simply a file descriptor provided by the operating system, and `selectionObject` is a pointer to the file descriptor. The `selectionObject` returned by `sa<Area>SelectionObjectGet()` is valid until `sa<Area>Finalize()` is invoked on `<area>Handle`.

The following code fragment illustrates how to detect pending area invocations for various library associations referred to by the handle parameter of the corresponding `sa<Area>SelectionObjectGet()` calls. Note that multiple active handles for the same area can exist at a point in time.

Example

```
#define MAX_AREA 5
SaSelectionObjectT fd[MAX_AREA];
void (*dispatch[MAX_AREA])();
SaUint32T *handle[MAX_AREA];
SaUint32T handle0;
SaUint32T handle1;
...
int i;
fd_set rfds;
int nfds = 0;
int numArea = 0;
struct timeval timeout;
sa<Area0>SelectionObjectGet(handle0, &fd[numArea]);
dispatch[numArea] = (void *) sa<Area0>Dispatch;
handle[numArea] = &handle0;
numArea++;
sa<Area1>SelectionObjectGet(handle1, &fd[numArea]);
dispatch[numArea] = (void *) sa<Area1>Dispatch;
handle[numArea] = &handle1;
numArea++;
...
FD_ZERO(&rfds);
for (i=0; i<numArea; i++) {
     if (nfds < fd[i]) nfds = fd[i]; /* find max fd */
     FD_SET(fd[i], &rfds);
}
select(nfds+1, &rfds, NULL, NULL, &timeout);
for (i=0; i<numArea; i++) {
     if (FD_ISSET(fd[i], &rfds)) (*dispatch[i])(*handle[i],
          SA_DISPATCH_ONE);
}
```

When the process detects that invocations are pending for a library association and is ready to process them, it calls the relevant sa<Area>Dispatch() function. This invocation may be made in the main thread or in a dedicated thread. Dispatching with different priorities can be achieved by initializing multiple associations, each with a dedicated thread running at the appropriate operating system priority.

Prototype

```
SaAisErrorT sa<Area>Dispatch(
     Sa<Area>HandleT <area>Handle,
     SaDispatchFlagsT dispatchFlags
);
```

The <area>Handle is obtained from the sa<Area>Initialize() function, and the dispatchFlags specify the callback execution behavior of the sa<Area>Dispatch() function. In the context of the calling thread, the sa<Area>Dispatch() function invokes pending callbacks for the handle designated by <area>Handle in the way specified by the dispatchFlags parameter.

If no callbacks are pending, and sa<Area>Dispatch() is invoked with either the SA_DISPATCH_ONE or the SA_DISPATCH_ALL flags, it returns immediately with an SA_AIS_OK value. For the meaning of the SA_DISPATCH_ONE and SA_DISPATCH_ALL flags, refer to Section 2.3.13 on page 56.

Different threads of a process can invoke sa<Area>Dispatch() on the same handle. As a consequence, several pending callbacks may be invoked concurrently. It is up to the application to provide concurrency control (for instance, locking), if needed.

2.1.2.4 Hidden Threads

The SA Forum APIs are designed to avoid imposing a particular thread programming model on applications and allows both singlethreaded and multithreaded processes to use SA Forum APIs. This means, in particular, that the APIs are designed in a way that does not force the library implementer to hide threads inside the library (as this would lead to singlethreaded application code to execute in a multithreaded environment).

Section 2.1.2.3 on page 21 shows an example of such a design choice: various sa<Area>Dispatch() API calls are specified, which allows the application process to provide the threads that will execute the callback functions.

2.1.3 Interaction Between AIS and POSIX APIs

In a POSIX environment, the AIS functions can be invoked concurrently by different threads of a process. Hence, the AIS functions must be thread-safe. However, this specification does not require that the AIS functions can be safely invoked from a signal handler.

When developed in a POSIX environment, greater portability of applications from one AIS implementation to another can be attained by observing the following rules during application development:

- Avoid using any SA Forum API from a signal handler.
- Do not assume that SA Forum APIs are interruptible by signals.
- Do not assume that SA Forum APIs are thread cancellation points.
- Do not assume that the AIS functions are fork-safe. Therefore, if a process using AIS functions forks a child process in which AIS functions will be called, the child process should `exec()` a new program immediately after being forked. This new program can then use AIS functions.

2.1.4 Memory Management

2.1.4.1 Usage of [in], [out], and [in/out] in Parameters

AIS Services use the acronyms [`in`], [`out`], and [`in/out`] in the description of parameters. These acronyms have the following meaning:

- [**in**] is used when a parameter passes information to the invoked function, and the invoked function shall not modify that information. These parameters are also said to be 'passed by value'.
- [**out**] is used when the caller passes a memory area by a pointer, and no additional information for the invoked function is passed in this memory area. The invoked function supplies the requested information into the provided memory area. These parameters are also said to be 'passed by reference'.
- [**in/out**] is used when a parameter passes information to the invoked function and receives information from the invoked function. These parameters are also said to be 'passed by reference'.

2.1.4.2 Memory Allocation and Deallocation

Rule 1

Memory dynamically allocated by one entity (user process or service area library) is deallocated by the same entity that allocated it. This rule has only one exception, described in rule 2.

Rule 2

In the following cases, it is simpler to have the area service library allocate the buffer and have the service user deallocate the memory:

- It is not easy to provide a buffer of the appropriate size by the invoking process, as it is hard to predict in advance how much memory is actually required.
- Avoid excessive copying for performance reasons.

This type of use must be clearly documented, because it is a potential source of memory leaks.

Each area service providing a function that dynamically allocates memory for a user process must provide a function to be called by the user to deallocate the allocated memory.

The following prototype definitions and a code sample illustrate the use of rule 2.

Prototype

```
typedef struct{
      char *buf;
      SaInt32T len;
} SaXxxBufferT;

SaAisErrorT saXxxReceive(SaXxxHandleT handle, SaXxxBufferT *buffer);

SaAisErrorT saXxxReceiveDataFree(SaXxxHandleT handle, char *buf);
```

Example

```
SaXxxxBufferT msg;
SaInt32T myLen;
msg.buf = NULL;
error = saXxxReceive(handle, *msg);
if (error != SA_AIS_OK) { /* handle error */ }
if (msg.buf != NULL) {
     /* process message */
     myLen = msg.len; /* area service sets length */
     process_message(msg.buf, myLen);
     saXxxReceiveDataFree(handle, msg.buf);
     msg.buf = NULL;
};
```

2.1.4.3 Handling Pointers in a Process and in an Area Service

The following notes explain how a service user process and the area service should handle pointers passed as parameters:

- When the area service library invokes a callback function provided by the process, and that callback function has a parameter that is a pointer, the process must not use that pointer after the callback function has returned. Rather, if the process needs to retain the information passed by the pointer, it must copy the information into memory that it has allocated.
- When the process invokes a synchronous function provided by the area service, the area service must not retain any pointer passed to it as a parameter of that function after the function has returned.
- When the process invokes an asynchronous function provided by the area service, the area service must not retain any pointer passed to it as a parameter of that function after the area service has invoked the corresponding asynchronous callback function.

2.1.5 Track APIs

Some AIS Services provide the capability to track changes in entities that they define. The **track APIs** can track a single entity or a group of entities, depending on the AIS Service. In the remainder of this description, the term “entity” is used to represent a single entity or a group of entities. Client processes of an AIS Service initiate the tracking and interact with it by invoking the track APIs of the AIS Service. For example, the Message Service allows tracking the membership of message queues within message queue groups.

The track APIs of AIS Services providing them are not identical, but similar. Their main characteristics are described in this section. A track API typically consist of three or four functions:

- Track an entity
- Stop tracking an entity
- Callback to notify changes or pending changes of an entity
- Respond to a notification callback (optional)

The format of a function name is:

```
sa<Area><Xxx>Track[<Func>]()
```

where `<Area>` and `<Func>` denote the area service and one of the track functions respectively. `<Xxx>` identifies the kind of changes being tracked by the set of API functions.

How the entity to be tracked is identified is specific to the service and typically includes one handle. For example:

- A tracked queue group is identified by the Message Service handle and the queue group name.
- A tracked cluster membership is identified by the Cluster Membership Service handle.

A client process can specify different kinds of tracking behavior by using the track flags, which are described in detail in Section 2.3.12.

Some AIS Services offer enhanced track functionality. The client processes can

- request to be notified before the tracked entity is changed; they can also
- validate (that is, accept or reject) a request to change the status of a tracked entity and can be notified to perform some actions prior to a change in a tracked entity taking effect.

For details on the enhanced tracking functions, see Section 2.1.5.6.

2.1.5.1 Track an Entity

A call to the function

```
SaAisErrorT sa<Area><Xxx>Track(
     /* service-specific parameters that identify the
      entity to be tracked */
     SaUin8T trackFlags,
     Sa<Area><Xxx>NotificationBufferT *notificationBuffer
);
```

tracks an entity as specified by the trackFlags parameter (see Section 2.3.12 on page 55).

If the flag SA_TRACK_CURRENT is set in the trackFlags parameter, this function retrieves status information of the tracked entity at the time of the call. If the notificationBuffer parameter is not NULL, the status information is passed in the structure to which notificationBuffer points; otherwise, it is passed asynchronously in the callback notification API.

The structure pointed to by notificationBuffer is of the following type:

```
typedef struct{
     /* Optional fields specific to the service */
     SaUint32T numberOfItems;
     Sa<Area><Xxx>NotificationT *items;
} Sa<Area><Xxx>NotificationBufferT;
```

If items is NULL, the area service will allocate an array for the required information, and items will be set to point to this array. The required information will be placed by the area service library into the allocated array when the sa<Area><Xxx>Track() call returns. It is the responsibility of the calling process to invoke the corresponding free function of the area service library to deallocate the allocated memory for the array (see Section 2.1.5.5 on page 30).

Status changes in a tracked entity are always passed asynchronously by an invocation of the callback notification API; however, if the trackFlags parameter contains the SA_TRACK_CURRENT flag and none of the flags SA_TRACK_CHANGES and SA_TRACK_CHANGES_ONLY, a one-time status request is made. No subsequent status changes are notified, unless they have been requested in a preceding sa<Area><Xxx>Track() call.

A client process may call sa<Area><Xxx>Track() repeatedly for the entity, regardless of whether the call initiates a one-time status request or a series of callback notifications.

If a process had called sa<Area><Xxx>Track() with SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY and calls sa<Area><Xxx>Track() again for the same entity, the following applies, depending on the flags in the second call:

- In case the second call has SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY set, the new combination of flags is used to change the settings for the tracking.
- In case the second call did not set SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY, but only SA_TRACK_CURRENT, the former tracking will proceed unchanged, and the user will additionally receive the current status information.

2.1.5.2 Callback Notification

If a client process called sa<Area><Xxx>Track() such that asynchronous notifications will take place, these notifications are passed to the process by the following callback.

```
typedef void (*Sa<Area><Xxx>TrackCallbackT)(
      /* service-specific parameters that identify the
       tracked entity and provide additional information on
       the cause of the callback invocation */
      SaInvocationT invocation,
      Sa<Area><Xxx>NotificationBufferT *notificationBuffer,
      SaErrorT error
);
```

The invocation parameter is only present in area services providing enhanced tracking, as these area services require in some cases a response from the client process to the notification. The notificationBuffer parameter points to the information on the tracked entity according to the trackFlags parameter in the previous corresponding sa<Area><Xxx>Track() call. Memory required for this information is always allocated by the area service, and it cannot be accessed outside the callback routine.

2.1.5.3 Responding to a Track Notification Callback

Some area services providing enhanced tracking expect a response from the client process to the notification of a change or a pending change in a tracked entity. A call to the function

```
SaAisErrorT sa<Area><Xxx>TrackResponse(
      /* parameters specific to the service */
      SaInvocationT invocation,
      /* parameters specific to the service */
);
```

provides a response to the track callback notification identified by the `invocation` parameter.

2.1.5.4 Stop Tracking an Entity

A call to the function

```
SaAisErrorT sa<Area><Xxx>TrackStop(
      /* service-specific parameters that identify the
       tracked entity */
);
```

stops tracking the entity. No more callback notifications about entity status changes will be sent to the process.

This call is only needed if `sa<Area><Xxx>Track()` was previously invoked, and this invocation was not a one-time status request for the entity.

2.1.5.5 Deallocating Memory Allocated for Tracking an Entity

A call to the function

```
SaAisErrorT sa<Area><Xxx>NotificationFree(
      /* service-specific parameters that identify the
       tracked entity */
      Sa<Area><Xxx>NotificationT *items
);
```

deallocates the memory pointed to by the `items` parameter. This memory was allocated by the area service library in a previous call to the `sa<Area><Xxx>Track()` function.
For details when this memory is allocated, refer to the description of the `items` field

in the Sa<Area><Xxx>NotificationBufferT structure (see Section 2.1.5.1 on page 28).

2.1.5.6 Enhanced Tracking

In the normal case, client processes are notified of a change in a tracked entity after the change has already occurred. However, some use-cases require that client processes be able to accept or reject a request to change the status of a tracked entity or to be able to perform actions prior to the change taking effect.

The track interface provides four options for enhanced tracking:

- Validate: enables subscribed processes to receive a request to validate, that is, to accept or reject the operation that is the cause of the change.
- Start: enables subscribed processes to receive a notification of an imminent change in a tracked entity.
- Completed: enables subscribed processes to be notified that the change has been effected.
- Aborted: enables subscribed processes to be notified when a request to validate a change operation was rejected.

Subscribers must use the track flags (see Section 2.3.12) to indicate whether they request to be notified in the start or validate step.

The sequence of steps in enhanced tracking is:

(1) SA_<area>_CHANGE_VALIDATE
The track callbacks are invoked requesting the subscribed processes to validate the pending action and to prepare themselves for the action. The invoked processes must provide a response to the area service by invoking the sa<area>Response() function.

(2) SA_<area>_CHANGE_START or SA_<area>_CHANGE_ABORTED
If at least one subscribed process whose track callback function was invoked during the SA_<area>_CHANGE_VALIDATE step rejects the operation, the AIS Service invokes track callbacks indicating that the pending action has been aborted (SA_<area>_CHANGE_ABORTED step); otherwise, when all subscribed processes invoked during the SA_<area>_CHANGE_VALIDATE step have provided a response stating that they agreed with the change, the area service invokes the track callbacks again requesting the processes to now perform any required action before the change (SA_<area>_CHANGE_START step). Note that if a subscribed process has not provided a response because the handle with which the track was started has become invalid in the meantime, the area server interprets this condition, as if this subscribed process had accepted the

operation. Processes must respond to the AIS Service when the operation is completed, or if they fail to complete the operation.
When subscribed processes are not allowed to reject the pending change, they may be directly notified with an SA_<area>_CHANGE_START step without any prior tracking notification with an SA_<area>_CHANGE_VALIDATE step.

(3) SA_<area>_CHANGE_COMPLETED
When all subscribed processes involved in the SA_<area>_CHANGE_START step reported that they have completed their actions, the AIS Service performs actions required to complete the change. When the change is completed, the subscribed processes are notified by the AIS Service (SA_<area>_CHANGE_COMPLETED step).

2.1.6 Retrieving Implementation-Specific Limits of AIS Services

A service implementation usually has limits such as the maximum number of supported entities of a certain type, the maximum size of certain objects, or similar limits. These limits are used to bound resource consumption.

Some AIS Services define a set of limits and interfaces, so that an invoking process can retrieve the values for these limits for a particular AIS Service implementation.

Note that no managed object's attributes are defined by the AIS Service specifications to configure these values in the IMM Service, as these values are usually predefined by the implementation.

Each AIS Service defining implementation-specific limits provides an `Sa<Area>LimitIdT` enumeration containing the set of values that identify these limits. Typically, these limits refer to the maximum number or size of entities of a certain type that the implementation can support; however, other limits such as thresholds can be defined.
A process can retrieve at runtime the current value of a particular limit by specifying the corresponding identifier of the limit when invoking the `sa<Area>LimitGet()` function. The limit value is returned in a parameter of a generic type (`SaLimitValueT` type, see Section 2.3.16 on page 57). For further access to the limit returned by the `sa<Area>LimitGet()` function, the programmer should use the field of the `SaLimitValueT` type that corresponds to the type of the particular limit.

Prototype

```
SaAisErrorT sa<Area>LimitGet(
     Sa<Area>HandleT <area>Handle,
     Sa<Area>LimitIdT limitId,
     SaLimitValueT *limitValue
);
```

The invoking process provides values for `<area>Handle` and `limitd`, and the memory to which `limitValue` points. The handle `<area>Handle` is the handle which was obtained by a previous invocation of the `sa<Area>Initialize()` function and which identifies this particular initialization of the <Area> Service. `limitId` identifies the limit whose implementation-specific value is to be retrieved. The <Area> Service returns in the memory area to which `limitValue` points the current value of the limit specified in `limitId`.

```
typedef enum {
      SA_<AREA>_<NAME1>_ID = 1,
      ...
      SA_<AREA>_<NAMEn>_ID = n
} Sa<Area>LimitIdT;
```

The name of each value in the enumeration is constructed according to the rule described in Section 2.2.5 on page 40 and must end with `_ID`. `<NAMEi>` consists of underscore-separated names. Example: `SA_EVT_MAX_NUM_CHANNELS_ID`.

Some implementations may not define a fixed value for a specific limit; instead, the specific limit will be reached when some other resource (such as memory) has reached its limit. In these cases, an implementation may return the maximum value for the type of the limit.

Example: 0x7FFFFFFFFFFFFFFF can be used for the limit of the Event Service that is identified by `SA_EVT_MAX_NUM_CHANNELS_ID`.

2.1.7 Unavailability of the Area Service API on a Non-Member Node

This section describes the behavior of area services under various conditions that cause the area service to be unavailable on a node. Section 2.1.7.1 on page 35 contains guidelines to area service implementers for dealing with a temporary unavailability of providing service.

First, some definitions are provided. For the terms cluster node and member node refer to [5].

The attribute **cluster-wide** is used to indicate logical entities that span one or more cluster nodes and that are designated by names unique in the entire cluster. Checkpoints and event channels are examples of cluster-wide entities.

The attribute **node-local** is used to indicate logical entities that are defined on a cluster node only. These entities are accessible by processes on this node only and are designated by names unique within the node. Timers of the Timer Service and node-local naming contexts are examples of node-local entities.

In general, operations of an area service that target cluster-wide entities are not allowed for processes running on cluster nodes that are not in the cluster membership, except for operations that enable or detect the formation of the cluster membership.
In general, operations of an area service that target node-local entities (such as node-local contexts of the Naming Service), are supported for processes running on the same cluster node where the entity resides, even if the cluster node is not in the cluster membership.
The Timer Service defines only node-local entities, and it does provide service to processes on cluster nodes that are not in the cluster membership.

The specification of an area service describes the exact behavior of the respective service API functions under various conditions that cause the service to be unavailable within the scope of a node.

2.1.7.1 Guidelines for Service Implementers

The implementation of an area service must leverage the SA Forum Cluster Membership Service to determine the membership status of a node for the cases described in the preceding section before returning `SA_AIS_ERR_UNAVAILABLE`. If the Cluster Membership Service considers a node as a member of the cluster, but the area 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`.

2.2 Naming Conventions

The naming conventions for constants, types, variables, and functions defined in the SA Forum Application Interface Specification are covered in this section. The Application Interface Specification is broken down into interface areas. An interface area consists of a set of self-contained APIs that can be provided as a single library with its associated header file(s). Each interface area is assigned an interface area tag (or simply area tag, if the context makes it clear) that identifies the functions pertaining to a specific area.

Application interface area tags:

- `Hpi` ::= Hardware Platform Interface
- `Amf` ::= Availability Management Framework
- `Ckpt` ::= Checkpoint Service
- `Clm` ::= Cluster Membership Service
- `Evt` ::= Event Service
- `Imm` ::= Information Model Management Service
- `Lck` ::= Lock Service
- `Log` ::= Log Service
- `Msg` ::= Message Service
- `Nam` ::= Naming Service
- `Ntf` ::= Notification Service
- `Plm` ::= Platform Management Service
- `Sec` ::= Security Service
- `Smf` ::= Software Management Framework
- `Tmr` ::= Timer Service

<`Area`> used in names (see the following subsections) consists of the interface area tag followed by an optional sub-area tag:

```
<Area> = <Area tag> [<Sub-area tag>]
```

The <`Sub-area tag`> is currently only defined for the Information Model Management Service. Two values are defined for the <`Sub-area tag`> of this service:

- `Om` for Object Management
- `Oi` for Object Implementer

2.2.1 Case Sensitivity

All usage of strings in the AIS documents is assumed to be case sensitive, and an AIS Service implementation must not make any assumptions regarding the strings being case insensitive, especially for processing and comparison purposes.

2.2.2 Global Function Declarations

The function name of a global declaration (that is, one that is visible to an application component) has a prefix that starts with the letters `sa` (in lowercase) for “service availability”, followed by `<Area>`, which identifies the area of the specification. The remaining part of the function name is formed from capitalized words that are descriptive of the object, action, and tag of the function.

Prototype

```
type sa<Area><Object><Action><Tag>(<arguments>);
```

where `sa` = prefix for “service availability”

- `<Area>` = interface area
- `<Object>` = name or abbreviation of object or service
- `<Action>` = name or abbreviation of action
- `<Tag>` = tag for the function such as Async or Callback

Example without <Sub-area Tag>

```
SaAisErrorT saEvtChannelOpen(
     const SaEvtHandleT evtHandle,
     const SaNameT *channelName,
     SaEvtChannelOpenFlagsT channelOpenFlags,
     SaTimeT timeout,
     SaEvtChannelHandleT *channelHandle
);
```

`<Area>` = `Evt` for Event Service, `<Object>` = `Channel`, and `<Action>` = `Open`.

Example with <Sub-area Tag>

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

<Area> = ImmOm for the Object Management sub-area of the Information Model Management Service, <Object> = Object, and <Action> = Delete.

Some other common <Action> suffixes are:

- Request
- Response
- Set
- Get

For functions that solicit an asynchronous invocation from the area service, the prototype has an Async suffix, unless it is obvious from the <Action> suffix. The corresponding callback invocation function prototype has a Callback suffix.

Examples

```
SaAisErrorT saClmClusterNodeGetAsync(
     SaClmHandleT clmHandle,
     SaInvocationT invocation,
     SaClmNodeIdT nodeId,
     SaClmClusterNodeT *clusterNode
);
```

<Area> = Clm for Cluster Membership Service, <Object> = ClusterNode, <Action> = Get, and <Tag> = Async.

```
typedef void (*SaClmClusterNodeGetCallbackT)(
     SaInvocationT invocation,
     const SaClmClusterNodeT *clusterNode,
     SaAisErrorT error
);
```

<Area> = Clm for Cluster Membership Service, <Object> = ClusterNode, <Action> = Get, and <Tag> = Callback.

2.2.3 Type Declarations

The names of types that are visible to an application component have a prefix that starts with the letters Sa, followed by <Area>, which identifies the area of the specification. The remaining part of the name is formed from capitalized words that describe the type.

Prototype

```
typedef <...> Sa<Area><TypeName>T;
```

Example

```
typedef SaUint32T SaCkptHandleT;

typedef SaUint32T SaEvtChannelOpenFlagsT;
```

2.2.4 Macro Declarations

The names of macros that are visible to an application component use only uppercase letters and the digits 0-9. Underscores are used to separate words and improve readability. Macro names start with the letters “SA”, followed by an underscore, followed by `<Area>`, followed by an underscore, and followed by underscore-separated words.

Prototype

```
#define SA_<AREA>_<MACRO NAME> <macro definition>
```

Example

```
#define SA_EVT_HIGHEST_PRIORITY 0
```

2.2.5 Enumeration Type Declarations

The names of enumeration elements use only uppercase letters and the digits 0-9. Underscores are used to separate words and improve readability. Element names start with the letters SA, followed by an underscore, followed by `<Area>`, followed by an underscore, and underscore-separated words.

Prototype

```
typedef enum {
     SA_<AREA>_<ENUMERATION_NAME1> [= <value>],
     SA_<AREA>_<ENUMERATION_NAME2> [= <value>],
     ....
     SA_<AREA>_<ENUMERATION_NAMEn> [= <value>]
} <enumeration type name>;
```

Example

```
typedef enum {
     SA_CKPT_SECTION_VALID           = 1,
     SA_CKPT_SECTION_CORRUPTED       = 2
} SaCkptSectionStateT;
```

2.3 Standard Predefined Types and Constants

2.3.1 Boolean Type

The `SaBoolT` type defines the standard boolean type.

Definition

```
typedef enum {
     SA_FALSE   = 0,
     SA_TRUE    = 1
} SaBoolT;
```

2.3.2 Signed and Unsigned Integer Types

The set of fixed bit-width integer types expected to be supported on all platforms are defined in the following subsections.

2.3.2.1 Signed Types

• Signed 8 bit integer quantity:	`SaInt8T`
• Signed 16 bit integer quantity:	`SaInt16T`
• Signed 32 bit integer quantity:	`SaInt32T`
• Signed 64 bit integer quantity:	`SaInt64T`

A typical declaration of these types on a 32-bit platform is as follows:

```
typedef char SaInt8T;
typedef short SaInt16T;
typedef int SaInt32T;
typedef long long SaInt64T;
```

2.3.2.2 Unsigned Types

• Unsigned 8-bit integer quantity:	`SaUint8T`
• Unsigned 16 bit integer quantity:	`SaUint16T`
• Unsigned 32 bit integer quantity:	`SaUint32T`
• Unsigned 64 bit integer quantity:	`SaUint64T`

A typical declaration of these types on a 32-bit platform is as follows:

```
typedef unsigned char SaUint8T;
typedef unsigned short SaUint16T;
typedef unsigned int SaUint32T;
typedef unsigned long long SaUint64T;
```

2.3.3 Floating Point Types

Two floating point types are defined to store numbers in formats specified by the IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754, see [17]):

- `SaFloatT` is used to store numbers in the IEEE 754 32-bit single-precision format.
- `SaDoubleT` is used to store numbers in the IEEE 754 64-bit double-precision format.

On most processor architectures these floating point types are typically defined as:

```
typedef float SaFloatT;
typedef double SaDoubleT;
```

2.3.4 String Type

The `SaStringT` type is used to specify an array of characters ending with the null character ('\0').

Definition

```
typedef char * SaStringT;
```

Note that in cases when a pointer to the `SaStringT` is used, it is interpreted as a pointer to a pointer.

2.3.5 Size Type

The `SaSizeT` type is used to specify sizes of objects.

Definition

```
typedef SaUint64T SaSizeT;
```

2.3.6 Offset Type

The `SaOffsetT` type is used to specify offsets in data areas.

Definition

```
typedef SaUint64T SaOffsetT;
```

2.3.7 Time Type

The `SaTimeT` type is used to specify time values. A time value is always expressed as a positive number of nanoseconds, except for the `SA_TIME_UNKNOWN` constant, which is defined later in this section.

The `SaTimeT` type can be interpreted as either an absolute timestamp or a time duration.

An interface specification containing a parameter of `SaTimeT` type should state how the time value is interpreted. If no such statement is present, a duration value is assumed.

Definition

```
typedef SaInt64T SaTimeT;
```

Granularity

Nanoseconds = 10^{-9} seconds

Range

Approximately 292 years

In some cases, it is necessary to represent an unknown time value. A special value is reserved for this:

Definition

```
#define SA_TIME_UNKNOWN 0x8000000000000000LL
```

This hexadecimal constant corresponds to a time of -2^{63} nanoseconds.

2.3.7.1 Timestamps

A timestamp is represented in an SaTimeT data item as the number of positive nanoseconds elapsed since 00:00:00 UTC, 1 Jan 1970.

It is common to use the term “absolute time” for a timestamp. These two terms are often used interchangeably.

Definition

```
#define SA_TIME_END 0x7FFFFFFFFFFFFFFFLL
```

SA_TIME_END represents the largest timestamp value: Fri Apr 11 23:47:16.854775807 UTC 2262.

Definition

```
#define SA_TIME_BEGIN 0x0LL
```

SA_TIME_BEGIN represents the smallest timestamp value: Thu 1 Jan 00:00:00 UTC 1970.

2.3.7.2 Time Durations

A time duration is represented in an SaTimeT data item as the number of positive nanoseconds counted from a specific reference time, for instance, the time of an API call.

For the convenience of the user, the following values are defined:

```
#define SA_TIME_ONE_MICROSECOND                      1000LL
#define SA_TIME_ONE_MILLISECOND                   1000000LL
#define SA_TIME_ONE_SECOND                     1000000000LL
#define SA_TIME_ONE_MINUTE                    60000000000LL
#define SA_TIME_ONE_HOUR                    3600000000000LL
#define SA_TIME_ONE_DAY                    86400000000000LL
#define SA_TIME_MAX                               SA_TIME_END
```

A duration of SA_TIME_MAX is interpreted as an infinite duration. If a timeout parameter is set to SA_TIME_MAX when invoking an AIS API function, no time limit will be associated with this request. This value should be viewed as a convenience value for programmers who do not care about timeouts associated with various APIs. Typically, it is not advisable to use SA_TIME_MAX in timeout parameters; the other predefined constants should generally suffice.

2.3.8 Sequence of Octets Type

The SaAnyT type is used to define a set of arbitrary octets.

Definition

```
typedef struct {
     SaSizeT bufferSize;
     SaUint8T *bufferAddr;
} SaAnyT;
```

2.3.9 Name Type

The SaNameT type is intended to be used to represent logical entity names that are passed in SA Forum APIs. It allows for both human-readable and other representations. Human-readable representations include ASCII and multibyte character locales. The length field in the SaNameT structure refers to the number of octets (bytes) used by the representation of the name in the value field. If the C character string representation is used, the value field contains the characters in the string without the terminating null character, and the length field contains the number of these characters.

Definition

```
#define SA_MAX_NAME_LENGTH 256

typedef struct {
     SaUint16T length;
     SaUint8T value[SA_MAX_NAME_LENGTH];
} SaNameT;
```

Example

```
SaNameT myName;
...
myName.length = strlen("fred");
memcpy(myName.value, "fred", myName.length);
error = saXxxCreateObject(myName,yyy,zzz);
```

2.3.9.1 Note on AIS Object Names

AIS Services use LDAP distinguished names (DNs) to name objects in the SA Forum Information Model. These DNs are used in runtime APIs, administrative APIs, and alarms and notifications to refer, as appropriate, to a managed object or to the logical entity that the object represents.
These LDAP DNs follow UTF-8 encoding conventions described in [15].

The scope of these names is limited to a single instance of the SA Forum Information Model that is maintained by an single instance of the Information Model Management Service. Hence, the names do not include any relative distinguished name (RDN) to identify the SA Forum Information Model instance.

LDAP names are encoded in `SaNameT` by using their UTF-8 representation without a terminating null character. The backslash ('\', ASCII 92) is used as escape character, as described in [15]; however backward compatibility support toward earlier standards (RFC 1779) is not required. Only printable Unicode characters must be used in LDAP names. This simplifies printing or displaying these names (see [16]).

The supported formats of DN and RDN types of the various names used in the AIS Services are described in the corresponding specification documents. See also [1]. The SA Forum does not define the object identifiers (OIDs) corresponding to RDN types [15].

2.3.9.1.1 Recommendations on RDN Values

The Information Model Management Service recognizes and is capable of handling RDN values of types `SaStringT` and `SaNameT`. If there is a need to use any other type as an RDN value, a mapping between the values of the given type and one of the allowed by IMM types need to be defined. Multivalued RDNs are not supported.

An SA Forum-defined RDN type has the "`saf`" prefix followed by a string describing the entity or entity type the object represents. For example the "`safSu`" is the RDN type for objects representing service units of the Availability Management Framework. The same string is used in the UML object class to name the attribute containing the RDN value. Appropriately, "`safSu`" is the name of the RDN attribute of the "`SaAmfSu`" object class. A value of this attribute may be "`mySu`". The complete RDN is constructed by concatenating the RDN attribute name string (for instance, "`safSu`") with the RDN attribute value (for instance, "`mySu`") through the '=' character, resulting in the RDN “`safSu=mySu`”.

The values of the RDN of CLM nodes, AMF nodes, and PLM execution environments should be identical to or derived from the operating system node name, provided that this notion is supported by the operating system configured to run as the execution environment.

In cases of potential name conflicts, RDN values for different objects should include a specific prefix. The stock symbol of the company providing the application is an example for such prefix.

RDNs are concatenated to form the DN of an object of the SA Forum Information Model. When exposed by the AIS interfaces, these DNs are encapsulated in an `SaNameT` data structure and normalized as follows:

- All tabs or white spaces before or after '=' separating the RDN type from the RDN value, and before or after the ',' character separating the RDNs, are removed.
- Only ',' is used to separate RDNs.
- Because `SaNameT` has a size of 256 characters,
  - the size of the RDN values represented as UTF-8 strings is limited to 64 characters;
  - the RDN attribute name (that is, the RDN type) shall be kept to a minimum.

2.3.9.1.2 Notation Used to Specify DN Formats

This section describes the notation used to specify DN formats.

In the subsequent discussion <`rdnType`> denotes an arbitrary RDN type and '...' denotes an arbitrary RDN value. For example, the DN of objects located at the root of the SA Forum Information Model is of format of “<`rdnType`>=...".

The square brackets '[' and ']' are used to indicate an optional portion of the RDN. For example, “[,<`rdnType`>=...]” indicates that the RDN of type <`rdnType`> is optional.

The star '*' and the plus sign '+' characters are used to indicate that an element may be repeated any number of times. In case of '*', the element is optional, in case of '+', it has to be present at least once. For example, "<`rdnType`>=...,*" indicates that the RDN of type <`rdnType`> may be followed by any number of RDNs of any type. The '*' becomes effectively a wildcard. The “[,<`rdnType`>=...]+” indicates that the RDN of type <`rdnType`> must be present at least once and can be repeated any number of times.

Finally, alternatives are expressed using the '|' character. For example, the expression "`<rdnType>`=A|B|C" indicates that the RDN of type `<rdnType>` may take only values A, B, or C.

2.3.9.1.3 DN Conventions

As objects in the SA Forum Information Model are arranged into a tree based on their DNs, the DN formats indirectly determine the organization of SA Forum Information Model.

The SA Forum does not mandate an exact naming schema and, therefore, an exact organization of the Information Mode. However, some conventions are respected when the DNs are defined for the different AIS Services. These conventions are driven by considerations of the scope and the life-cycle of the relevant objects and need to be observed when new DNs are defined. New DNs can be defined, for example, for an application that uses the Information Model Management Service to store its application objects.

Objects of global relevance are placed immediately at the root of the SA Forum Information Model; therefore, their DN consists of a single RDN and has the format of:

"`<rdnType>`=..."

Example: "`safCluster=myClmCluster`".

It is desirable that the number of such global objects is limited; therefore, objects relevant only to a particular application or to an AIS Service are placed under the subtree of the related application or service. This is reflected in the DN format as follows (see Section 2.3.9.2 for the `safApp` values for AIS Services):

"`<rdnType>`=...,`safApp`=..."

Example:
"`safStaticFilter=myStaticFilter`,`safApp=safNtfService`".

A particular case is the configuration object used to configure an AIS Service; this object has the DN format of:

"`safRdn`=...,`safApp`=..."

Example: "`safRdn=immManagement`,`safApp=safImmService`"

These placements reflect the dependency between an object and the relevant application or service in the sense that if the service or application is removed from the system, and, therefore, the representing object is removed from the SA Forum Information Model, all the child objects of the service or application are also removed.

Logical objects that are associated primarily with their location within the cluster are placed under the CLM cluster or the appropriate CLM node. Accordingly, their DN format is respectively:

"`<rdnType>=...,safCluster=...`"

or

"`<rdnType>=...,safNode=...,safCluster=...`".

Examples:

"`safNamContext=saNamContextClusterDefault`,
`safCluster=myClmCluster`"

"`safNamContext=saNamContextNodeDefault`,`safNode=myClmNode1`,
`safCluster=myClmCluster`"

The latter DN format ensures that if a node is removed from the cluster, the Naming Service default context associated with the node is also removed from the cluster.

In many cases, the SA Forum does not mandate a particular structure of the SA Forum Information Model, or it specifies the structure only partially. In these cases, the DN format (at the point of the wildcard) allows for many different arrangements:

"`<rndType>=...,*`"

or

"`<rndType>=...,*,safApp=...`"

The first format is satisfied by any of the following DNs:

"`safMq=myMsgQueue`",

"`safMq=myMsgQueue`,`safApp=myApplication`",

or

"`safMq=myMsgQueue`,`safCsi=myCsi`,`safSi=mySi`, `safApp=myApplication`".

Examples for the second format are:

"`safSwBundle=myBundle`,`safApp=safSmfService`", or

"safSwBundle=myBundle,safRdn=myRepository,
safApp=safSmfService".

In these cases, it is left up to the application or site designer to come up with a concrete naming schema that guarantees the consistency of the Information Model throughout the system lifetime. In any case, objects of classes of weaker persistency must not be parents of objects of classes of stronger persistency. For example, using the concepts of [3], a runtime object must not be defined as a parent of a configuration object.

Within the SA Forum Information Model, an attribute of type SaNameT is always interpreted as a DN and, therefore, as a reference to an object within the Information Model. Accordingly, the UML association class relationship is reflected through a particular use of the DNs and RDNs of the objects participating in the association. If the object *x* is an object of the association class between two associated objects *y (*the object of the first associated class) and *z* (the object of the other associated class), then *x* is defined as a child of *y* and the RDN attribute of *x* is set to point to *z*. This is achieved by setting the RDN value of *x* to the DN of *z*. The direction of the association between the associated classes determines the objects of the associated object classes that take the parent role. That is, the association is navigable from the class taking the parent role to the object of the class to which the RDN attribute points. For example, considering FIGURE 3, an object of the SaAmfCSIAssignment object class is a child of an object of the SaAmfCSI object class and has, accordingly, the DN format:

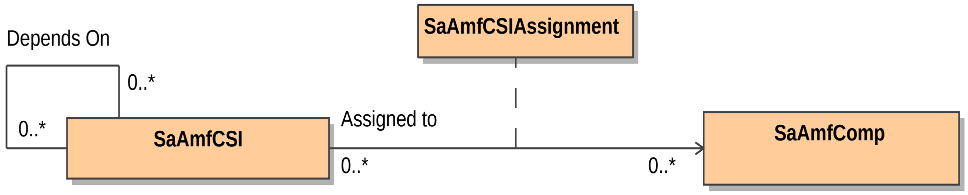
"safCSIComp=...,safCsi=...,safSi=...,safApp=...".

The RDN value of the safCSIComp RDN type is set to the DN of the associated object of the SaAmfComp class. This means that the DN of the object of the SaAmfCSIAssignment class can be unfolded as:

"safCSIComp=safComp=...\,safSu=...\,safSg=...\,safApp=...,safCsi=...,
safSi=...,safApp=...".

Note that the '\' escape character is used within the DN, which is used as an RDN value.

FIGURE 3 Example of a UML Association Class Relation



2.3.9.2 Well-known DNs for AIS Services

The SA Forum defines some well-known DNs for the AIS Services that it specifies. This is explained in the following subsections.

2.3.9.2.1 Values for the safApp Application RDN of AIS Services

An object representing an application in the SA Forum Information Model has the RDN type `safApp`, as defined in [4]. This RDN type is also used to define standard RDNs for AIS Services, regardless of whether the actual service implementation is managed by the Availability Management Framework. The RDN values use a common format of `safApp=saf<Area>Service`[:`<varAppName>`], where the `saf<Area>Service` part has constant well-known values (as defined below), and the `<varAppName>` is an arbitrary string (according to the rules defined in Section 2.3.9.1.1 and Section 2.3.9.1.2).

Availability Management Framework	`"safApp=safAmfService"`
Checkpoint Service	`"safApp=safCkptService"`
Cluster Membership Service	`"safApp=safClmService"`
Event Service	`“safApp=safEvtService"`
Hardware Platform Interface	`“safApp=safHpiService"`
Information Model Management Service	`“safApp=safImmService"`
Lock Service	`"safApp=safLckService"`
Log Service	`"safApp=safLogService"`
Message Service	`"safApp=safMsgService"`
Naming Service	`"safApp=safNamService"`
Notification Service	`"safApp=safNtfService"`
Platform Management Service	`"safApp=safPlmService"`
Security Service	`"safApp=safSecService"`
Software Management Framework	`"safApp=safSmfService"`
Timer Service	`"safApp=safTmrService"`

The <varAppName> part of the RDN value can be used to distinguish between multiple implementations of the same AIS Service.

2.3.9.2.2 Values for the safAppType and safVersion RDNs for AIS Services

Each object representing a software application deployed in the SA Forum cluster (with the RDN type safApp) refers in its saAmfAppType attribute to the particular version of the application software it is using. The DN of the object representing a version of an application software has the format of "safVersion=...,safAppType=...".

This section specifies the DNs used to represent the software implementing the AIS Services and, in particular, the way how the DNs should refer to the specification (including its release) that the software implements and to which it is compliant.

The common format is

"safVersion= <specRel>[:<vendVersion>],
safAppType=saf<Area>Service[:<vendImplRef>]".

Explanation:

- The <specRel> part indicates the exact version of the latest specification to which the implementation is compliant. This version is shown on the front page of the specification, for instance, "B.06.01 ".
- The <vendVersion> part can be used for vendor-specific versioning of the implementation.
- The saf<Area>Service part has the constant well-known value, as defined in Section 2.3.9.2.1.
- The <vendImplRef> part is used to indicate the software implementation of a particular vendor.

Together, the saf<Area>Service and the <specRel> parts identify the version of the <Area> portion of the instance of the SA Forum Information Model maintained by the Information Model Management Service.

For example, if the object with the DN "safApp=safAmfService" representing the AMF implementation of a system has its saAmfAppType attribute set to "safVersion=B.05.01:myVersion,safAppType=safAmfService:mySAF", that means that this implementation is compliant to the B.05.01 release of the Availability Management Framework specification. It also indicates that the vendor implementation is mySAF of version myVersion.

2.3.10 SaServicesT

The following type enumerates the SA Services specified by the SA Forum.

```
typedef enum {
      SA_SVC_HPI       = 1,
      SA_SVC_AMF       = 2,
      SA_SVC_CLM       = 3,
      SA_SVC_CKPT      = 4,
      SA_SVC_EVT       = 5,
      SA_SVC_MSG       = 6,
      SA_SVC_LCK       = 7,
      SA_SVC_IMMS      = 8,
      SA_SVC_LOG       = 9,
      SA_SVC_NTF       = 10,
      SA_SVC_NAM       = 11,
      SA_SVC_TMR       = 12,
      SA_SVC_SMF       = 13,
      SA_SVC_SEC       = 14,
      SA_SVC_PLM       = 15
} SaServicesT;
```

2.3.11 Version Type

The SaVersionT type is used to represent software **versions** of area implementations. Application components can use instances of this type to request compatibility with a particular version of an SA Forum Application Interface area specification. The area referred to is implicit in this API. See also Section 2.4 on page 62 for a discussion on backward compatibility rules.

Definition

```
typedef struct {
      SaUint8T releaseCode;
      SaUint8T majorVersion;
      SaUint8T minorVersion;
} SaVersionT;
```

`releaseCode`: The **release code** is a single ASCII capital letter [A-Z]. All specifications and implementations with the same release code are backward compatible. For details on how the SA Forum will handle backward compatibility, refer to Section 2.4 on page 62. It is expected that the release code will change very infrequently. Release codes are assigned exclusively by the SA Forum.

`majorVersion`: The **major version** is a number in the range [01..255]. An area implementation with a particular major version number implies compliance to the interface specification bearing the same release code and major version number. Changes to a specification requiring a revision of the major version number are expected to occur at most once or twice a year for the first few years, becoming less frequent as time goes on. Major version numbers are assigned exclusively by the SA Forum.

`minorVersion`: The **minor version** is a number in the range [01..255]. Successive updates to an area implementation complying to an area interface specification bearing the same release code and major version number have increasing minor version number starting from 01. Increasing minor version numbers only refer to enhancements of the implementation, like better performance or bug fixes. Different values of the minor version may not affect the compatibility and are not used to check whether required and supported versions are compatible.

Successive updates to an area interface specification with the same release code and major version number will also have increasing minor version numbers starting from 01. However, such changes to a specification are limited to editorial changes that do not impose changes on any software implementations for the sake of compliance. Minor version numbers are assigned independently by the SA Forum for interface specifications and by members and licensed implementers for their implementations.

Example

```
SaVersionT myAmfVersion;
...
myAmfVersion.releaseCode         = 'B';
myAmfVersion.majorVersion        = 0x02;
myAmfVersion.minorVersion        = 0x00;
/* Version "B.02.xx" */
error = saAmfInitialize(handle, const &callbacks, *myAmfVersion);
```

2.3.12 Track Flags

The following constants are used by the sa<Area><Xxx>Track() API for all area services with track APIs to specify what is to be tracked and what information is supplied in the notification callback.

```
#define SA_TRACK_CURRENT 0x01
```

Information about all entities is returned immediately, either in a notification buffer as indicated by the caller or by a single subsequent notification callback.

```
#define SA_TRACK_CHANGES 0x02
```

The notification callback is invoked each time at least one change happens in the set of entities, or one attribute of at least one entity in the set changes. Information about all entities is passed to the notification callback (both for entities in which a change occurred and for entities in which no change occurred).

```
#define SA_TRACK_CHANGES_ONLY 0x04
```

The notification callback is invoked each time at least one change happens in the set of entities, or one attribute of at least one entity in the set changes. Only information about entities that changed is passed in the notification callback.

```
#define SA_TRACK_LOCAL 0x08
```

Some area services may support the tracking of only a particular entity of the set of all entities to be tracked. Which particular entity is meant by the SA_TRACK_LOCAL constant is specified by the area service.
If this flag is used together with SA_TRACK_CURRENT, only information about this particular entity is returned.
If this flag is used together with SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY, the notification callback is invoked only if this entity is affected by the change. The notification callback passes information only about that entity.
If an area service does not support this option, the flag will be ignored.

It is not permitted to set both SA_TRACK_CHANGES and SA_TRACK_CHANGES_ONLY in an invocation of sa<Area><Xxx>Track(). If both flags are set, the call to sa<Area><Xxx>Track() will return with SA_AIS_ERR_BAD_FLAGS, and tracking will not be started.

The call of the function is also invalid and will return SA_AIS_ERR_ BAD_FLAGS if none of the flags SA_TRACK_CHANGES, SA_TRACK_CHANGES_ONLY or SA_TRACK_CURRENT are set.

The following constants are used by the sa<Area><Xxx>Track() API for enhanced tracking (see Section 2.1.5.6)

```
#define SA_TRACK_START_STEP 0x10
```

The client process requests that the notification callback is called in the start step. This flag is ignored if the area service does not provide the enhanced track interface or if neither SA_TRACK_CHANGES nor SA_TRACK_CHANGES_ONLY is set.

```
#define SA_TRACK_VALIDATE_STEP 0x20
```

The client process requests that the notification callback is called in the validate step. This flag is ignored if the area service does not provide the enhanced track interface or if neither SA_TRACK_CHANGES nor SA_TRACK_CHANGES_ONLY is set.

2.3.13 Dispatch Flags

The following enumeration type is used by the dispatch function for each of the different areas.

```
typedef enum {
      SA_DISPATCH_ONE              = 1,
      SA_DISPATCH_ALL              = 2,
      SA_DISPATCH_BLOCKING         = 3
} SaDispatchFlagsT;
```

The values of the SaDispatchFlagsT enumeration type have the following interpretation:

- SA_DISPATCH_ONE - Invoke a single pending callback in the context of the calling thread, and then return from the dispatch.
- SA_DISPATCH_ALL - Invoke all of the pending callbacks in the context of the calling thread if callbacks are pending before returning from dispatch.
- SA_DISPATCH_BLOCKING - One or more threads calling dispatch remain within dispatch and execute callbacks as they become pending. The thread or

threads do not return from dispatch until the corresponding finalize function is executed by one thread of the process.

2.3.14 Selection Object

The `SaSelectionObjectT` type is used for selection objects. Selection objects are operating system-dependent objects allowing a process to wait until an invocation of a callback function is pending for it.

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

Definition

```
typedef SaUint64T SaSelectionObjectT;
```

2.3.15 Invocation

The `SaInvocationT` type is used to match a callback call to the call requesting the callback. For details, including an example, refer to Section 2.1.1.1 on page 16.

Definition

```
typedef SaUint64T SaInvocationT;
```

2.3.16 SaLimitValueT

The `SaLimitValueT` type is used to retrieve the value of an implementation-specific limit. For details, refer to Section 2.1.6 on page 33.

Definition

```
typedef union {
      SaInt64T   int64Value;
      SaUint64T  uint64Value;
      SaTimeT    timeValue;
      SaFloatT   floatValue;
      SaDoubleT  doubleValue;
} SaLimitValueT;
```

2.3.17 Error Codes

To simplify the coding of error handling, error codes returned by SA Forum Application Interface Specification APIs are globally unique and are defined as follows.

```
typedef enum {
    SA_AIS_OK                                   = 1,
    SA_AIS_ERR_LIBRARY                          = 2,
    SA_AIS_ERR_VERSION                          = 3,
    SA_AIS_ERR_INIT                             = 4,
    SA_AIS_ERR_TIMEOUT                          = 5,
    SA_AIS_ERR_TRY_AGAIN                        = 6,
    SA_AIS_ERR_INVALID_PARAM                    = 7,
    SA_AIS_ERR_NO_MEMORY                        = 8,
    SA_AIS_ERR_BAD_HANDLE                       = 9,
    SA_AIS_ERR_BUSY                             = 10,
    SA_AIS_ERR_ACCESS                           = 11,
    SA_AIS_ERR_NOT_EXIST                        = 12,
    SA_AIS_ERR_NAME_TOO_LONG                    = 13,
    SA_AIS_ERR_EXIST                            = 14,
    SA_AIS_ERR_NO_SPACE                         = 15,
    SA_AIS_ERR_INTERRUPT                        = 16,
    SA_AIS_ERR_NO_RESOURCES                     = 18,
    SA_AIS_ERR_NOT_SUPPORTED                    = 19,
    SA_AIS_ERR_BAD_OPERATION                    = 20,
    SA_AIS_ERR_FAILED_OPERATION                 = 21,
    SA_AIS_ERR_MESSAGE_ERROR                    = 22,
    SA_AIS_ERR_QUEUE_FULL                       = 23,
    SA_AIS_ERR_QUEUE_NOT_AVAILABLE              = 24,
    SA_AIS_ERR_BAD_FLAGS                        = 25,
    SA_AIS_ERR_TOO_BIG                          = 26,
    SA_AIS_ERR_NO_SECTIONS                      = 27,
    SA_AIS_ERR_NO_OP                            = 28,
    SA_AIS_ERR_REPAIR_PENDING                   = 29,
    SA_AIS_ERR_NO_BINDINGS                      = 30,
        SA_AIS_ERR_UNAVAILABLE                      = 31,
        SA_AIS_ERR_CAMPAIGN_ERROR_DETECTED          = 32,
        SA_AIS_ERR_CAMPAIGN_PROC_FAILED             = 33,
        SA_AIS_ERR_CAMPAIGN_CANCELED                = 34,
        SA_AIS_ERR_CAMPAIGN_FAILED                  = 35,
        SA_AIS_ERR_CAMPAIGN_SUSPENDED               = 36,
        SA_AIS_ERR_CAMPAIGN_SUSPENDING              = 37,
        SA_AIS_ERR_ACCESS_DENIED                    = 38,
        SA_AIS_ERR_NOT_READY                        = 39,
        SA_AIS_ERR_DEPLOYMENT                       = 40
} SaAisErrorT;
```

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_VERSION - This value is returned in any of the two cases:

- The version specified in the call to initialize an instance of the service library is not compatible with the version of the implementation of the particular service.
- The invoked function is not supported in the version specified in the call to initialize the used instance of the service library.

SA_AIS_ERR_INIT - A callback function that is required for this API was not supplied in a previous call of sa<Area>Initialize().

SA_AIS_ERR_TIMEOUT - An implementation-dependent timeout or the timeout specified in the API call 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 component or process might try again later.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

SA_AIS_ERR_BAD_HANDLE - A handle is invalid.

SA_AIS_ERR_BUSY - A resource is already in use, or the AIS Service is busy with another task.

SA_AIS_ERR_ACCESS - Access is denied due to a reason other than a security violation.

SA_AIS_ERR_NOT_EXIST - An entity to which is referred does not exist.

SA_AIS_ERR_NAME_TOO_LONG - The size of a name exceeds the maximum length.

SA_AIS_ERR_EXIST - An entity to which is referred already exists.

SA_AIS_ERR_NO_SPACE - The buffer provided by the component or process is too small.

SA_AIS_ERR_INTERRUPT - The request was canceled by a timeout or other interrupt.

SA_AIS_ERR_NOT_SUPPORTED - The requested function is not supported.

SA_AIS_ERR_BAD_OPERATION - The requested operation is not allowed.

SA_AIS_ERR_FAILED_OPERATION - The requested operation failed.

SA_AIS_ERR_NO_RESOURCES - There are not enough resources to provide the service.

SA_AIS_ERR_MESSAGE_ERROR - A communication error occurred.

SA_AIS_ERR_QUEUE_FULL - For the description of this error code, refer to the Message Service specification ([7]).

SA_AIS_ERR_QUEUE_NOT_AVAILABLE - For the description of this error code, refer to the Message Service specification ([7]).

SA_AIS_ERR_BAD_FLAGS - The flags are invalid.

SA_AIS_ERR_TOO_BIG - A value is larger than the maximum value permitted.

SA_AIS_ERR_NO_SECTIONS - For the description of this error code, refer to the Checkpoint Service specification ([6]).

SA_AIS_ERR_NO_OP - The requested operation had no effect.

SA_AIS_ERR_REPAIR_PENDING - The administrative operation is only partially completed as some targeted components must be repaired.

SA_AIS_ERR_NO_BINDINGS - For the description of this error code, refer to the Naming Service specification ([8]).

SA_AIS_ERR_UNAVAILABLE - The operation requested in this call is unavailable on this cluster node as the cluster node is not a member node, and the requested operation is not permitted on a non-member node.

SA_AIS_ERR_CAMPAIGN_ERROR_DETECTED - For the description of this error code, refer to the Software Management Framework specification ([9]).

SA_AIS_ERR_CAMPAIGN_PROC_FAILED - For the description of this error code, refer to the Software Management Framework specification ([9]).

SA_AIS_ERR_CAMPAIGN_CANCELED - For the description of this error code, refer to the Software Management Framework specification ([9]).

SA_AIS_ERR_CAMPAIGN_FAILED - For the description of this error code, refer to the Software Management Framework specification ([9]).

SA_AIS_ERR_CAMPAIGN_SUSPENDED - For the description of this error code, refer to the Software Management Framework specification ([9]).

SA_AIS_ERR_CAMPAIGN_SUSPENDING - For the description of this error code, refer to the Software Management Framework specification ([9]).

SA_AIS_ERR_ACCESS_DENIED - The required access to a particular function of the AIS Service is denied due to a security violation.

SA_AIS_ERR_NOT_READY - For the description of this error code, refer to the Availability Management Framework specification ([4]).

SA_AIS_ERR_DEPLOYMENT - The requested operation was accepted and applied at the information model level. However, its complete deployment in the running system may not be guaranteed at the moment.

2.4 Notes on Backward Compatibility

To achieve backward compatibility when evolving the AIS specification in the future, the SA Forum will follow the rules below. Note, however, that this goal can only be achieved with the cooperation of AIS implementers.

- A function or type definition never changes for a specific SA Forum release.
- Changes in a function or type definition (adding a new argument to a function, adding a new field to a data structure) force the definition of a new function or type name. A new function or type name is built from the original name in the previous version with a suffix indicating the version where the function/type changed (for instance, `saAmfComponentRegister_3()`).
- As an exception to the previous rule, new enum values, flag values, or union fields can be added to an existing enum, flag, or union types without changing the type name, as long as the size of the enum, flag, or union type does not change.
- AIS implementers must ensure that they respect the version numbers provided by the application when it initializes the library and do not expose new enum values to applications using older versions.
- AIS implementers must also ensure that they respect the version numbers provided by the application when the library is initialized, with regard to new or modified error codes and do not expose error codes that only apply to functions in the most recent version of the specification to applications written to an older version of the specification.

AIS implementers must also ensure that they respect the version numbers provided by the process when the library is initialized, with regard to which function can be invoked by the process with the returned library handle. Only functions corresponding to the requested version can be invoked using the returned library handle. In case of version mismatch, the function returns the `SA_AIS_ERR_VERSION` error code.

As an example, consider a `majorVersion` `Vx` of a given service that includes a function `f()`, and assume that `f()` had to be modified in a newer `majorVersion` `Vy` (`Vy` > `Vx`) which led to the introduction of the `f_y()` variant that now replaces `f()` in `Vy`.

Considering an implementation that supports both versions `Vx` and `Vy`, a process can initialize the library specifying either `Vx` or `Vy`:

- if the process initializes a library handle with `Vx`, this handle does not provide access to functions that have been introduced in versions newer than `Vx`. In particular, this handle will not enable the process to successfully invoke `f_y()`.

- if the process initializes a library handle with Vy, this handle does not provide access to a function introduced in versions older than Vy and then replaced by a newer variant of the same function. In particular this handle will not enable the process to successfully invoke `f()`.

The specification document of an AIS Service for Vy only includes the latest variant of a function or type definition supported by Vy.

Index of Definitions