Service Availability™ Forum Application Interface Specification

Platform Management Service SAI-AIS-PLM-A.01.02

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

List of Figures

List of Tables

1 Document Introduction

1.1 Document Purpose

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

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

1.2 AIS Documents Organization

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

1.3 History

The first and only previous release of the Platform Management Service specification is:

SAI-AIS-PLM-A.01.01

This section presents the changes of the current release, SAI-AIS-PLM-A.01.02, with respect to the SAI-AIS-PLM-A.01.01 release. Editorial changes that do not change semantics or syntax of the described interfaces are not mentioned.

1.3.1 New Topics

None

1.3.2 Clarifications

⇒ In Section 3.1.1 on page 19 on the role of PLM in the overall architecture, in a paragraph on page 20, the text fragment “modeled as objects of the `SaPlmEE` object class” has replaced the fragment “modeled as an object class“. In the same paragraph, the word “directly” has been added to the second sentence. Note also a few editorial changes in the paragraph following this one.

⇒ The title of FIGURE 1 on page 21 has changed.

⇒ In the first paragraph of the description of the not-present state in Section 3.1.3.1.1 on page 26 on the HE presence state, a reference to Section 4.5.1 has been added.

⇒ The first paragraph in the description of the management-lost readiness flag of an HE in Section 3.1.3.1.5 and the first paragraph in the description of the management-lost readiness flag of an EE in Section 3.1.3.2.5 have been clarified.

⇒ In Section 3.1.3.1.5 on the readiness flags of an HE, the first paragraph in the description of the admin-operation-pending flag and the first paragraph in the description of the isolate-pending flag have been clarified. The same clarification has been added to the description of the `SA_AIS_ERR_TRY_AGAIN` error code of the administrative operation with `SA_PLM_ADMIN_RESET` as `operationId` in Section 5.4.9.

⇒ In Section 3.1.3.2.1 on the EE presence state, in the paragraph on page 38 describing the instantiation-failed state, “failed” was mistakenly used instead of “disabled”.

⇒ The last bullet in the explanation of the in-service readiness state of an EE in Section 3.1.3.2.4 on page 41 has been clarified.

⇒ In Section 3.1.3.2.5 on page 42 on the readiness flags, in the second bullet describing the admin-operation-pending flag, the text fragment “has applied the administrative state” has replaced the text fragment “has performed the administrative operation“. In the same section, a few clarifications have been added to the first paragraph in the description of the management-lost flag.

⇒ The first bullet in Section 3.1.4.2 on page 49 on the PLM Virtualization Support has been slightly rephrased.

⇒ In Section 3.1.5.1 on page 51 on the verification of the hardware configuration, it been clarified that the PLM Service may issue the Unmapped Hardware Entity Alarm.

⇒ In Section 3.3.13.3 on page 64, additional clarification is provided in the description of the `SA_PLM_CAUSE_MANAGEMENT_LOST` and `SA_PLM_CAUSE_MANAGEMENT_REGAINED` values.

⇒ The behavior of the `saPlmFinalize()` function in Section 3.4.4 on page 76 has been clarified in the second paragraph of the description section of this function.

⇒ The behavior of the `saPlmEntityGroupDelete()` function in Section 3.5.1.4 on page 83 has been clarified in the description section of this function.

⇒ A sentence has been added the end of the description subsection of Section 3.5.3.1 on page 99 on the `saPlmEntityReadinessImpact()` function.

1.3.3 Deleted Topics

None

1.3.4 Other Changes

⇒ Throughout the document, the term “isolation-pending flag” has been changed to “isolate-pending flag”.

⇒ In Section 3.3.13.3 on page 64, the new value `SA_PLM_CAUSE_STATUS_INFO` has been added to the `SaPlmTrackCauseT` enumeration. The meaning of this new value is explained in the mentioned section; this value is intended to be used, for instance, in connection with the track flag `SA_TRACK_CURRENT`. This modification has led to a change to the description of the `rootCorrelationId` parameter of the `SaPlmReadinessTrackCallbackT` function in Section 3.5.2.2 on page 92.

⇒ In the previous release, references to the `saPlmReadinessTrackResponse()` function were mistakenly written as `saPlmResponse()`. These corrections apply to Section 3.3.14, Section 3.5.2, Section 5.4.3, and in Appendix B.1.1, enumerations (14) and (24).

⇒ In Section 4.2, Table 1 on page 102 has been updated to add a root, "`safApp=safPlmService`", to the distinguished names for `SaPlmEEBaseType`, `SaPlmEEType`, `SaPlmHEBaseType`, and `SaPlmHEType` objects classes.

⇒ The object class `SaPlmEntity` has been made abstract. This is reflected in FIGURE 4 on page 103 by showing the `SaPlmEntity` in italics.

⇒ The `SaPlmDependency` object class has changed from an association class to a “regular” object class. In the previous version, only 1:1 dependencies between PLM entities were possible. This new definition allows for 1:N dependencies. Note also that the relationship between the `SaPlmEntity` and `SaPlmDependency` object classes has changed. These changes are shown in:

- Section 4.4, FIGURE 5 on page 104 depicting the PLM Instances and Types View and in
- Section 4.7, FIGURE 8 on page 112 depicting the PLM Other Classes diagram.

⇒ The code snippets for the `params` parameter in Section 5.4.2 on page 120 on the `SA_PLM_ADMIN_LOCK` and in Section 5.4.6 on page 129 on the `SA_PLM_ADMIN_RESTART` administrative operations have been changed to use correct C syntax.

⇒ In Table 5 on page 150 (Section 6.2.1.3) on the Hardware Element Security Alarm and in Table 6 on page 152 (Section 6.2.1.4) on the Execution Environment Security Alarm, the descriptions of the attributes Detector, Service User,

and Service Provider have changed. Note that a typo in the title of Table 5 on page 150 has also been corrected.

⇒ The place of the first quote in the LDAP DN of the notification object on page 160 has been corrected.

1.3.5 Superseded and Superseding Functions

None

1.3.6 Changes in Return Values of API and Administrative Functions

None

1.4 References

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

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

[2] Service Availability™ Forum, Service Availability Interface, C Programming Model, SAI-AIS-CPROG-B.05.02

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

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

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

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

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

[8] Service Availability™ Forum, SA Forum Information Model in XML Metadata Interchange (XMI) v2.1 format, SAI-IM-XMI-A.04.02.xml.zip

[9] Service Availability™ Forum, Application Interface Specification, Availability Management Framework, SAI-AIS-AMF-B.04.01

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

[11] CCITT Recommendation X.731 | ISO/IEC 10164-2, State Management Function

[12] CCITT Recommendation X.731 | ISO/IEC 10164-2 : 1992/Amd.2 : 2001

[13] Virtualization: State of the Art, Published by SCOPE Alliance, http://www.scope-alliance.org

[14] RFC 4506, XDR: External Data Representation Standard, http://www.rfc-archive.org/getrfc.php?rfc=4506

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

1.5 How to Provide Feedback on the Specification

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

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

1.6 How to Join the Service Availability™ Forum

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

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

1.7 Additional Information

1.7.1 Member Companies

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

1.7.2 Press Materials

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

2 Overview

This specification defines the Platform Management (PLM) Service within the Application Interface Specification (AIS).

2.1 Platform Management Service

The PLM Service provides a logical view of the hardware and low-level software of the system. Low-level software in this sense comprises the operating system and virtualization layers that provide execution environments for all kinds of software.
This logical view is presented in the Service Availability™ Forum Information Model by a set of objects that

- allow for the management of hardware entities and execution environments,
- allow other software to keep track of status changes of the hardware and execution environments, and
- allow the mapping of the HPI (see [3]) data to objects represented in the SA Forum Information Model.

The PLM Service typically uses HPI to derive all necessary information from the hardware. HPI discovers existing hardware and notifies its users about events (see [3]). The PLM Service may use implementation-specific means to derive hardware information which cannot be accessed via HPI.

The PLM Service not only provides the hardware information in the Service Availability™ Forum Information Model through the IMM Service (see [5]), but also provides objects that are administratively configurable. Additionally, the PLM Service is responsible for matching the configuration with the discovered hardware.

The main logical entities implemented by the PLM Service are:

- **Execution Environment** (**EE**)
  An EE is a logical entity that represents an environment capable of running software. An EE may or may not host one CLM node. In most cases, a CPU blade or an SMP machine runs one operating system modeled as an EE.
  When a hypervisor provides hardware virtualization, the hypervisor itself and each operating system running under its control are modeled as separate EEs.
- **Hardware Element** (**HE**)
  An HE is a logical entity that represents any kind of hardware entity, which can be, for instance, a chassis, a blade, or an I/O device.
  Typically, all FRUs (Field Replaceable Units) are modeled as HEs. If necessary, the system architect may model in the PLM Information Model additional entities

that are part of a FRU as hardware elements, for example, I/O ports, CPU cores, and so on.

The PLM Service maintains the state information of HE entities. For this purpose, it retrieves as necessary any information about the health of the hardware. The PLM Service may also map HPI events to notifications distributed by the Service Availability™ Forum Notification Service (see [4]) and generate these notifications.

Similarly, the PLM Service retrieves all necessary information about the health of the operating system and any available virtualization layer to maintain states of EE entities and generate necessary notifications about events of the EE entities.

The PLM Service allows application processes to register a callback function to receive notifications when PLM Service entities start or stop to provide service. This mechanism also allows application processes to gracefully shut down their own services when a PLM Service entity is about to terminate, for instance, when an extraction or an administrative LOCK operation is requested for the entity.

3 Platform Management Service API

3.1 Platform Management Service Model

3.1.1 Role of PLM in the Overall Architecture

The Platform Management Service merges the software world of AIS and the hardware world of HPI (Hardware Platform Interface, see [3]) to provide a homogeneous system view. It plugs the system view presented by the hardware platform interface into the information model used by AIS.

The system view of HPI represents the physical reality. HPI discovers the hardware entities that are present in the system and reflects them together with their states.

On the other side, AIS uses a preconfigured information model to represent the system. This view not only considers entities currently present, but also entities that are planned or entities that should be present but currently are not. So this configured system model exists regardless of whether the hardware is present or not.

It is the responsibility of the PLM Service to connect those views.

On the hardware side, the PLM Service typically uses HPI to determine the discovered system view. HPI discovers which hardware is actually present, independently from the configuration of the information model. HPI also reports hardware types and capabilities and provides means to assess the health states of hardware.

PLM provides a model to describe the desired or planned system configuration, that is, it provides object classes allowing the system architect to create object instances that represent the various pieces of hardware. This configuration is maintained by the SA Forum AIS Information Model Management Service (IMMS, see [5]). PLM is responsible for matching this configuration to the discovered hardware view. PLM checks whether the present hardware is of the right type and version, as specified in the SA Forum Information Model.

PLM represents a piece of hardware by a **hardware element** (**HE**) object. As the hardware architecture is typically organized in a hierarchical way, PLM reflects this organization with a containment tree of HE objects in the SA Forum Information Model. However, the mapping of HPI entities to HE objects is not necessarily one to one. It is left to the discretion of the system architect to decide which HPI entity or group of entities are mapped onto an HE object. Since PLM uses these objects in different maintenance tasks, it is strongly recommended to model at least every field

replaceable unit (FRU) as a separate HE object. For more details, see Section 3.1.3.4 on page 46.

Any software needs some sort of operating system or such an environment to organize the execution of the software. The PLM Service represents this environment with a logical entity, the **execution environment** (**EE**). Execution environments may or may not provide the capability to run SA Forum AIS middleware.
AIS Services need a CLM cluster node (see [6]) to run on, and every CLM cluster node needs an execution environment.

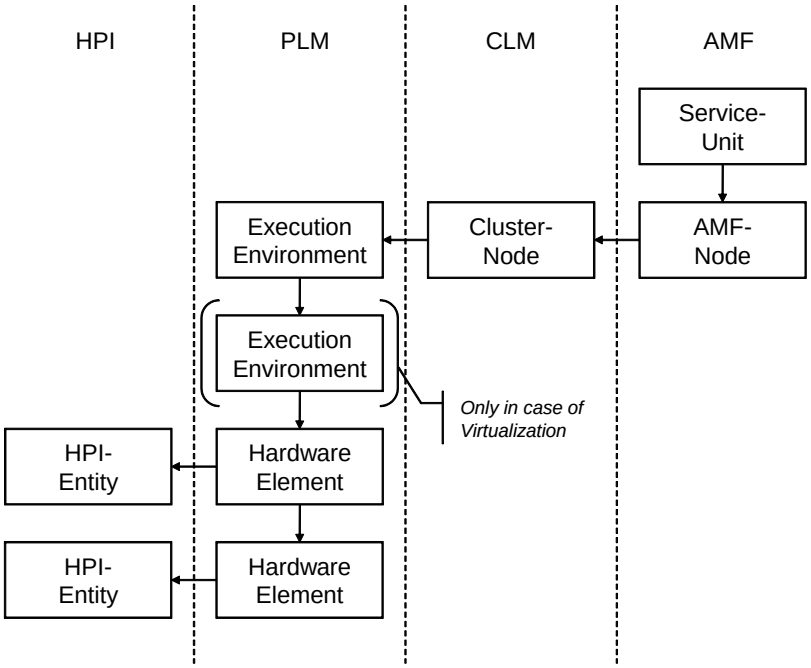
EEs are modeled as objects of the `SaPlmEE` object class in the SA Forum Information Model. An operating system running directly on an HE is thus represented by an EE object, its parent in the containment tree being the HE object.

EE objects are also used to model virtualized architectures in the PLM Information Model. In this case, there is also a containment hierarchy of EE objects. **Virtual machine monitors** (**VMM**), which are sometimes called **hypervisors**, are represented by **parent EEs**, **virtual machines** or **guest operating systems** by **child EEs**.

Note: In this document, the term “parent” of an object *X* means the object *Y* of which *X* is a (direct) child in the containment tree. A parent object may also have another parent. Thus, these parent objects form a list, and any parent in this list is termed an “ancestor” of *X*.
Similarly, the terms parent and ancestor are also used for logical entities: a logical entity *Y* is the parent (or an ancestor, respectively) of a logical entity *X* if— in the SA Forum Information Model—, the object corresponding to *Y* is the parent (or an ancestor, respectively) of the object corresponding to *X*.

Thus, the PLM Information Model connects the discovered hardware entities (modeled in HPI) with the AIS objects in the SA Forum Information Model as shown in FIGURE 1. The diagram shows in a simplified way the relationship of PLM and its objects with CLM and AMF on one side and with HPI on the other side.

FIGURE 1 Relating HPI Entities with the Rest of the Information Model Through PLM Objects



PLM also provides a state model for these entities and the necessary administrative operations. PLM is also responsible for maintaining these states. For HE entities, PLM needs to listen to HPI events and perform all necessary actions to retrieve from HPI the information about the hardware. For EE entities, implementation-specific interactions with the operating systems and virtualization facilities are needed.

Some operations on PLM entities may have a wide impact on services being provided. In many cases, when an operator or administrator performs maintenance operations, such as issuing a LOCK administrative operation or requesting a hot swap extraction, it is difficult to know whether the system provides sufficient redundancy to avoid a service outage. Therefore, the PLM Service not only informs its users about state changes of PLM Service entities, but it also provides the means to validate operations on hardware elements or execution environments before these operations are executed. The PLM Service also provides the means to force a LOCK administrative operation in urgent cases, accepting the service outage.

3.1.2 PLM Information Model

3.1.2.1 Hardware Elements

PLM uses the `SaPlmHE` object class for all kinds of hardware entities:

- container entities such as shelves, racks, or even slots;
- computing entities such as single board computer blades which are able to directly host an execution environment;
- resources of computing entities such as interfaces, chip-sets, memory, and CPUs;
- resources of the equipment such as fans and power-entry-modules.

Hardware elements are modeled in containments and should reflect the physical architecture of the hardware. For the highest containment level, the PLM Service provides the `SaPlmDomain` object class. The PLM Service implements a single instance of this class.

Since PLM maintains the state information and allows for administrative operations on hardware at the granularity of the HE objects, it is recommended to represent every FRU (field replaceable unit) by a separate HE. It is also possible to model the hardware in finer granularity.

PLM uses the `SaPlmHEType` and `SaPlmHEBaseType` object classes to support easy modeling of hardware units of the same hardware type or of similar hardware types and also to support hardware upgrade. HE types are also used to validate the configuration against the hardware entities that are present in the system (see Section 3.1.5 on page 51).

For a complete description of the classes and attributes that are used to configure an HE, refer to Section 4.5 on page 105.

3.1.2.2 Execution Environments

Without virtualization, a computing HE can run exactly one EE at a time.
With virtualization, a computing HE can run multiple EEs concurrently. Typically, when such an HE boots, a hypervisor or virtual machine monitor starts. This hypervisor or virtual machine monitor can control multiple operating systems, which constitute EEs that may run concurrently. There may be multiple instances of the same type of operating system or even of different types.
Many different architectures for virtual machine monitors are possible. In all cases, the PLM Service represents the virtual machine monitor by an EE as direct child of an HE. This EE is parent to all EEs representing the virtual machines. Note that a hyper-

visor EE can run multiple child EEs concurrently, whereas an HE, directly, can run only one EE at a time.

Thus, the following kinds of EEs are considered:

⇒ EEs running directly on an HE, which can be:

- Payload operating system (if there is no virtualization).
  The payload operating system may or may not have a configured CLM node.
- Parent EEs hosting child operating systems (with virtualization).
  The virtualization monitor (sometimes called hypervisor), running directly on an HE, may or may not also have a configured CLM node.

⇒ Child operating systems, which may run as virtual machines controlled by a virtualization monitor.
Child EEs may or may not run a configured CLM node.

In virtualized architectures, the containment tree shall reflect the relationships of virtualization monitors and virtual machines as parent—child relationships. In this version, the PLM Service does not support dynamic migration of virtual machines.
More details on virtualization can be found in Section 3.1.4.2.

All EEs are modeled using the `SaPlmEE` class. Every EE directly running on an HE is modeled as a child of this HE in the containment tree.

PLM uses the `SaPlmEEType` and `SaPlmEEBaseType` object classes to support easy modeling of multiple EEs of the same type and also to support upgrade of operating systems.
The EE types contain attributes that can be used by the PLM Service to validate the configuration against the types of execution environments that are present and installed in the system (see Section 3.1.5 on page 51).

For a complete description of the object classes and attributes that are used to configure an EE, refer to Section 4.6 on page 110.

For best practices on operating systems providing the notion of node name, it is recommended that the name of an EE be derived from the operating system node name (EE RDN: `safEE=`os_node_name). For details on naming issues, refer to [2].

3.1.2.3 Dependencies

There may be **dependencies** between PLM Service entities, that is, an entity cannot provide service without another entity being present and in-service. For instance, the outage of an entity may directly cause other entities to fail, or it may imply that PLM should actively terminate another entity. Dependencies may be caused by physical hardware conditions or by the way the system is configured. For instance, an EE may depend on a disk (which is located somewhere else in the system), because the disk is used as the system disk by the operating system.
These dependencies between PLM entities are reflected by dependencies between the objects representing these entities in the PLM Information Model. Some of the dependencies are expressed implicitly by the containment tree of the model. Other dependencies need to be explicitly specified.

Thus, a PLM object can only be in-service if all objects on which it depends are in-service. The dependency influences the readiness state. For the readiness state of HEs and EEs, see Section 3.1.3.1.4 and Section 3.1.3.2.4 respectively.

All objects depend implicitly on their ancestors, that is, if any ancestor gets out-of-service, all its children objects become out-of-service.

The following dependencies can be modeled explicitly in PLM, and PLM will react according to these dependencies and will also provide correlation of events of these objects.

- HEs may depend on HEs.
- EEs may depend on HEs and EEs.

Different “kinds” of dependencies are possible:

- **on-the-parent**
  Implicit dependency of an object on its parent or on any other ancestor.
- **one-on-one**
  Direct dependency of one object on another.
- **one-of-a-group**
  Dependency on one object of a set of objects.
  When the readiness state of at least one object in the dependency set is in-service, the readiness state of the entity that depends on the set can be in-service.
- **all-of-a-group**
  Dependency on all objects of a set of objects.
  When the readiness state of any object in the dependency set becomes out-of-service, the readiness state of the object that depends on the set also must be out-of-service.

- **n-of-a-group**
  Dependency on a subset of a set of objects.
  When fewer than n objects in the dependency set have their readiness state in-service, the readiness state of the object that depends on the set must not be in-service.
  For instance, object A needs at least two of the objects B, C, D, and E. Any two of them may be out-of-service while A can still be in-service. However, if a third object gets out-of-service, A will also be brought out-of-service by PLM.

Note also that one-on-one, one-of-a-group, and all-of-a-group are just special cases of n-of-a-group. The term **mandatory dependency** is used to identify any of the above dependencies if the readiness state of the depending entity would need to transition to out-of-service if the readiness state of an object the entity depends upon transitions to out-of-service.

Dependencies are modeled using the `SaPlmDependency` object class defined in Section 4.7 on page 112. An HE or EE object instance may be parent to any number of `SaPlmDependency` objects. An `SaPlmDependency` object specifies

- in its `saPlmDepNames` attribute the list of objects on which this particular dependency relies and
- in the `saPlmDepMinNumber` attribute the minimum number of objects that must be in-service in order to satisfy the dependency.

3.1.3 PLM State Model

The PLM state model is harmonized with the state model of the SA Forum™ AIS Services. In addition, the PLM state model includes readiness flags to qualify the readiness state.

The **PLM state model** defines the following types of states:

- Presence State
  The presence state is defined differently for HEs and EEs.
- Administrative State
  The administrative state is defined differently for HEs and EEs.
- Operational State
- Readiness State
- Readiness Flags

If a system needs to provide state management as defined in CCITT Recommendation X.731 (see [11]) to the outside world, the PLM state model can be mapped as described in Appendix A on page 165.

Each PLM entity is typically designed to provide one or several functions; these functions are called the **service** provided by the HE or EE.
An HE may include other parts providing, for example, control and management independently of the service the HE provides.
An EE typically provides its service when it is able to run application software.
All states and flags introduced in the next sections reflect—from different perspectives—the ability of an entity to provide its service.

In some cases, PLM disallows an entity to provide its service. This may be done with or without termination, power-off, or other operations on the entity itself. In fault situations, the PLM Service must **isolate** the faulty entity by using appropriate operations like hot swap or power down. Details on **isolation** of faulty entities and the necessary operations are described in Section 3.1.6 on page 52. PLM also provides administrative operations to isolate an entity.

3.1.3.1 HE States

3.1.3.1.1 HE Presence State

The **HE presence state** is used to manage physical presence of hardware. It resembles the HPI hot swap state in many ways but is not always identical. Possible values of the HE presence state of an HE are active, inactive, activating, deactivating, and not-present.

⇒ **not-present**

In the system, there is no hardware entity that matches the characteristics of this HE (see Section 4.5.1 on page 106 about mapping hardware entities to HEs).

For example, the following situations are reflected by this state:

- The hardware entity that is represented by the HE has been removed from the system. HPI has reported this fact to PLM by a state change of the entity to not-present hot swap state.
- There is no hardware entity in the system with an HPI entity path that can be matched to the `saPlmHEEntityPaths` configuration attribute of the HE (in most cases, this means the location is empty).
- There are hardware entities in the system with an HPI entity path that can be matched to the `saPlmHEEntityPaths` configuration attribute of the HE, but none of these hardware entities match the characteristics of the HE, as specified by the `saPlmHetIdr` attribute in the HE's `SaPlmHEType` object.

- There are hardware entities in the system with an HPI entity path that can be matched to the `saPlmHEEntityPaths` configuration attribute of the HE, and these hardware entities match the characteristics of the HE, as specified by the `saPlmHetIdr` attribute in the HE's `SaPlmHEType` object, but they were not mapped to any HE at the time the PLM Service performed the mapping (see Section 4.5.1 on page 106), as they were redundant.

If an HE is not-present, any child HE must also be not-present, and any child EE must have a presence state of uninstantiated.

PLM also needs to transition an HE to not-present if the management-lost readiness flag was set for the entity, and the operator issued a `SA_PLM_ADMIN_REMOVED` administrative operation (see Section 5.4.11).

Note that the `saPlmCurrHEType` and `saPlmHECurrEntityPath` attributes are not valid if an HE is not-present.

⇒ **inactive**

The hardware entity that is represented by the HE is physically located in the system but is functionally inactive and logically isolated from the platform.

The meaning of HE isolation is described in detail in Section 3.1.6 on page 52.

For example, PLM generally sets the presence state of an HE to inactive if one of the following conditions is met:

- The hardware entity supports hot swap, and its HPI hot swap state is inactive.
- The hardware entity supports power management and the entity is powered off.
- The entity supports reset management and reset state is asserted.
- Another hardware specific appropriate way is used to prevent the entity from providing service.

Typically, the HE presence state is inactive in the following situations:

- The HE is in administrative state locked-inactive.[1]
- The HE is isolated because it is faulty.
- The HE assumes the inactive state when PLM successfully completed an extraction request by opening the ejector latch (for instance, advancedTCA® and compactPCI™).

If an HE is inactive, any child HE must also be inactive or not-present, and any child EE must have a presence state of uninstantiated.

⇒ **activating**

This value is used as a transitional state when the presence state of the HE was inactive and should become active.

For example, the following situations are reflected by this state:

- When HPI notifies the PLM Service that an entity entered the hot swap insertion-pending state, the PLM Service—after performing implementation-specific checks—sets the HE presence state of the entity to activating and allows the entity to power on and become an active entity in the system.
- When the entity was inactive and is activated by software, for instance, by executing the `SA_PLM_ADMIN_ACTIVATE` (see Section 5.4.8) or `SA_PLM_ADMIN_REPAIRED` (see Section 5.4.10) administrative operations, the PLM Service also needs to transition the HE presence state of the entity to activating.

Note that PLM may also apply this state to HEs representing hardware that does not support the managed hot swap model.

⇒ **active**

From the hardware viewpoint, the entity is now an active member of the system.

If the hardware entity supports hot swap, its HPI hot swap state should be active. Generally, the entity must be powered on, and not have the reset state asserted or be prohibited from providing service by other means.

1. For the administrative state of HEs see Section 3.1.3.1.2 on page 31.

⇒ **deactivating**

This value is used as a transitional state when an HE was active and should become inactive or later not-present. When the presence state of an HE is set to deactivating, the hardware element may be undergoing a **graceful deactivation** or an **abrupt deactivation**.

Whether the deactivation is graceful or abrupt is not directly reflected in the state attributes of the HE, but PLM needs to handle these situations differently.

The deactivating state may be set as a result of a hardware state change or of an administrative operation.

PLM can detect whether a hardware entity is undergoing a graceful or abrupt deactivation by examining the hardware state communicated by HPI. For more details, see [3].

Generally, a graceful deactivation is detected when:

(a) A FRU that supports managed hot swap enters the EXTRACTION PENDING state, and PLM successfully cancels the HPI auto extraction policy.

(b) PLM detects a platform-specific status that indicates that a hardware entity is requesting a deactivation, and PLM has means to control the timing of the deactivation or to prevent the deactivation from happening.

An abrupt deactivation is detected when:

(a) A FRU that supports unmanaged hot swap enters the EXTRACTION PENDING state.

(b) A FRU that supports managed hot swap enters the EXTRACTION PENDING state, but PLM cannot cancel its auto extraction policy.

(c) PLM detects a platform-specific status that indicates that a hardware entity has begun a deactivation that cannot be interrupted.

When PLM detects a graceful deactivation of an HE, a number of actions are performed:

{1} PLM sets the presence state of the HE to deactivating.

{2} PLM checks the readiness state of the HE being deactivated and all dependent PLM entities[1]. If all are already out-of-service, processing continues with step {4}, below. If any are not out-of-service, the deactivation policy (see Section 3.3.11) determines how processing continues.

{3a} If the deactivation policy is `SA_PLM_DP_REJECT_NOT_OOS`, the deactivation is rejected; PLM signals HPI to change the entity back to an ACTIVE state and changes the HE presence state to active.

{3b} If the deactivation policy is `SA_PLM_DP_VALIDATE`, track callbacks are invoked for the `SA_PLM_CHANGE_VALIDATE` step to determine if the deactivation may proceed. If all processes receiving callbacks *accept* the deactivation, or if none of the entities that will be placed out-of-service by the deactivation are being tracked for the VALIDATE step, processing continues with step {3c}. If one or more processes reject the deactivation, the deactivation is rejected; PLM signals HPI to change the entity back to an ACTIVE state and changes the HE presence state to active, generating the appropriate state change notification, and track callbacks are invoked for the `SA_PLM_CHANGE_ABORTED` step.

Note: In the remainder of this document, saying that a track user *accepted* a track callback means that the track user agreed that the pending operation be performed, as it responded to the track callback by invoking the `saPlmReadinessTrackResponse()` function with the `response` parameter set to `SA_PLM_CALLBACK_RESPONSE_OK`.

{3c} If the deactivation policy is `SA_PLM_DP_UNCONDITIONAL`, or after all processes receiving VALIDATE callbacks accept the deactivation, track callbacks are invoked for the `SA_PLM_CHANGE_START` step. When all processes receiving callbacks at this step respond, processing continues at step {4}.

{4} If all PLM entities affected by the deactivation are out-of-service, or after all processes receiving START callbacks have responded, PLM initiates the necessary actions using HPI interfaces to actually deactivate the hardware.

1. Dependent PLM entities are the children and entities with a configured dependency relationship. The readiness state of an entity with a configured dependency is affected when—after the deactivation—fewer than `saPlmDepMinNumber` entities have a readiness state of in-service or stopping.

{5} When the deactivation is complete, PLM changes the presence state to inactive and the readiness state to out-of-service, and track callbacks are invoked for the SA_PLM_CHANGE_COMPLETED step.

When PLM detects an abrupt deactivation of a hardware element, no intervention by PLM or application programs is possible during the deactivation process. For abrupt deactivations, these actions are taken:

{1} PLM sets the presence state to deactivating.

{2} PLM monitors the hardware as it deactivates.

{3} When the deactivation is complete, PLM changes the presence state to inactive and the readiness state to out-of-service, and track callbacks are invoked for the SA_PLM_CHANGE_COMPLETED step.

PLM also sets the presence state to deactivating when a SA_PLM_ADMIN_DEACTIVATE administrative operation (see Section 5.4.7) is processed. The HE must be locked before SA_PLM_ADMIN_DEACTIVATE can be issued, so processing in PLM continues as with an abrupt deactivation, described above.

3.1.3.1.2 Administrative State

Like other AIS Services, the PLM Service defines the administrative state of an HE using an extension of the administrative state specified by ITU (see [11]). Possible values are locked, unlocked, locked-inactive, and shutting-down.

The **administrative state** for an HE has the following meaning:

⇒ **unlocked**

The HE has not been directly prohibited from providing service by the administrator.

⇒ **locked**

The administrator has prevented the HE from providing service, which also means that children of the HE may not provide service. The HE and its children may be active from the hardware viewpoint, but their readiness states report out-of-service. Other objects that have a mandatory dependency on this HE are also prevented from providing service. Hardware diagnostics can be started on the HE, but a configured EE is not allowed to execute.

⇒ **locked-inactive**

The administrator has prevented the HE from providing service by appropriate hardware means. PLM sets the hardware entity represented by the HE in a condition that causes its presence state to become inactive.

If the hardware does not provide appropriate means to support the presence state inactive, this administrative state may be not supported for an HE.

⇒ **shutting-down**

The administrator has prevented the hardware element and its contained and depending entities from providing service to new users. The hardware element's administrative state becomes locked as soon as all services it provides as well as all services provided by its contained and depending entities become unused by all their current users.

3.1.3.1.3 Operational State

The PLM Service defines the operational state as in other AIS specifications, which is different from the way ITU uses this term. As defined by the PLM Service, the **operational state** indicates whether or not an entity is faulty. Possible values are enabled and disabled.

The PLM Service detects failure conditions of HEs by analyzing the state of health of the hardware, listening to HPI events, and reading HPI state information. Additionally, PLM Service users can detect hardware failure conditions and report these conditions by invoking the `saPlmEntityReadinessImpact()` function (see Section 3.5.3.1).

⇒ **enabled**

The hardware entities represented by the HE are healthy and PLM is not aware of any failure conditions that would prevent the intended use of these entities. The operational state of an HE transitions from disabled to enabled when a successful repair action has been performed on the HE. Repair actions are reported to PLM with the `saPlmEntityReadinessImpact()` interface (see Section 3.5.3.1) or with the `SA_PLM_ADMIN_REPAIRED` administrative operation (see Section 5.4.10). The operational state may transition from disabled to enabled without a repair action being reported if PLM detects that the failure condition has cleared.

⇒ **disabled**

The operational state of an HE transitions to disabled if a failure condition is detected by analyzing the state of health of the hardware, by listening to HPI events, by reading HPI state information, or if a failure condition was reported with the `saPlmEntityReadinessImpact()` interface (see Section 3.5.3.1).

3.1.3.1.4 Readiness State

As for other AIS Services, the **readiness state** summarizes values of a set of states. Possible values are in-service, out-of-service, and stopping.

The readiness state indicates whether the HE provides its service. This state collects information from all other states, from the ancestor objects, and from mandatory dependencies.

⇒ **out-of-service**

The entity represented by the HE object does not provide service due to one or more of the following conditions:

- Its HE administrative state is locked or locked-inactive.
- Its HE presence state is neither active nor deactivating.
- Its operational state is disabled.
- The readiness state of the direct parent or of any other ancestor in the containment tree is out-of-service.
- The readiness state of a mandatory dependency is out-of-service, that is, fewer than `saPlmDepMinNumber` entities of a dependency have a readiness state of in-service or stopping.

If the readiness state transitions to out-of-service, because any of the preceding conditions change, PLM must actively terminate the entity. In case of the last two conditions, the dependency readiness flag is also set.

⇒ **in-service**

The readiness state of an HE is in-service if the entity can provide its service. So all of the following conditions must be met:

- Its administrative state is unlocked.
- Its HE presence state is active or deactivating.
- Its operational state is enabled.
- The readiness state of all ancestors is in-service.

- For each dependency, at least `saPlmDepMinNumber` entities are in-service.

⇒ **stopping**

The readiness state of an HE transitions from in-service to stopping when one of the following conditions is met:

- Its administrative state is set to shutting-down.
- The readiness state of an ancestor is set to stopping.
- The readiness state of an entity in a dependency is stopping, and the number of remaining in-service entities is lower than `saPlmDepMinNumber` while still `saPlmDepMinNumber` entities of that dependency are not out-of-service

The readiness state cannot transition from out-of-service to stopping.

3.1.3.1.5 Readiness Flags

The **readiness flags** complement the readiness state of an object by providing additional information. A flag is defined for each of the following situations:

⇒ **management-lost**

The PLM Service has management capability over an entity when the PLM Service is able to monitor and control the entity sufficiently to accurately model the entity via an HE or EE object that represents the entity. When the PLM Service loses some or all of its management capabilities such that this is not possible, the PLM Service sets the management-lost readiness flag for the entity to reflect this situation. In this case, the value of the operational, administrative, presence, and readiness states of the HE or EE object may or may not reflect the actual state of the entity.

The following applies when the management-lost readiness flag is set for the entity:

- Its administrative state may vary as a consequence of an administrative operation. If the PLM Service processes an administrative operation on the entity and attempts to change the administrative state of the entity but cannot perform all the necessary actions to apply the administrative state or cannot determine whether all these actions were completed, the administrative state of the entity is set to the intended value, and the PLM Service additionally sets the admin-operation-pending readiness flag for the entity.
  The PLM Service returns `SA_AIS_ERR_DEPLOYMENT` error code to the corresponding administrative operation on the entity.

- Its operational state may vary as a consequence of a failure detected by the PLM Service or reported by a PLM user with the invocation of the `saPlmEntityReadinessImpact()` function (see Section 3.5.3.1). If the failure of the entity is detected, and the PLM Service is not able to isolate the failed entity, the flag isolate-pending is additionally set. This flag is cleared by the PLM Service when it regains its management capability for the entity, or if the failure is cleared by an invocation of the `saPlmEntityReadinessImpact()` function (see Section 3.5.3.1) or by the execution of the `SA_PLM_ADMIN_REPAIRED` administrative operation (see Section 5.4.10).
- Its readiness state may vary as a consequence of changes in its administrative or operational states or as a consequence of changes in the readiness state of entities it depends upon.
- Its presence state is the last value known by the PLM Service, that is, the value of the presence state before the management-lost readiness flag was set for the entity.

When the PLM Service regains its capability to monitor and control the state of an entity for which the management-lost readiness flag was set, the PLM Service automatically clears this flag and updates the value of the operational, presence, and readiness states of the entity to reflect the current state of the entity. If the admin-operation-pending flag is set, the PLM Service must perform the pending administrative operation and must clear the flag. If the isolate-pending flag is set, and the operational state of the entity is still disabled, the PLM Service must isolate the failed entity.

When the operator issues an `SA_PLM_ADMIN_REMOVED` administrative operation (see Section 5.4.11) on the HE, PLM transitions the presence state of the HE to not-present and clears all readiness flags.

⇒ **dependency**

The dependency flag is set for an entity when one of its ancestors is not in-service, or fewer than `saPlmDepMinNumber` entities of one of its dependencies are not in-service. That is, this flag indicates that the entity is not in-service, at least in part due to the readiness state of an ancestor or of an entity upon which it is dependent.

⇒ **imminent-failure**

This flag is set for an entity when its operational state is enabled, and an imminent failure on the entity has been detected by the PLM Service or reported to the PLM Service with the `saPlmEntityReadinessImpact()` function (see Section 3.5.3.1).

The PLM Service clears this flag if the operational state of the entity is disabled or if the imminent-failure condition is cleared. The PLM Service may detect that the imminent-failure condition is cleared by analyzing the hardware state, or a user may report that an imminent-failure condition is cleared by calling `saPlmEntityReadinessImpact()` or by issuing the `SA_PLM_ADMIN_REPAIRED` administrative operation; however, if a user reports that an imminent-failure condition is cleared, the flag may remain set if PLM determines that the imminent-failure condition still exists.

⇒ **dependency-imminent-failure**

This flag is set for an entity when its operational state is enabled, but the imminent-failure or dependency-imminent-failure readiness flag is set for an ancestor or in enough entities upon which this entity depends, so that if they failed would cause the failure of this entity. In other words, this flag indicates that this entity is at risk of failure due to the failure of other entities that are known to be at risk of imminent-failure.

This flag is cleared if the operational state of the entity becomes disabled, or if the imminent-failure and dependency-imminent-failure readiness flags are cleared for all ancestor entities and for at least `saPlmDepMinNumber` entities that are not out-of-service in each of its dependencies.

⇒ **admin-operation-pending**

This flag is used together with the management-lost readiness flag, as has been explained previously. It indicates that the PLM Service was not able to perform all the actions necessary to apply the administrative state or could not determine whether all the actions were completed on the entity due to circumstances for which the management-lost readiness flag was set for the entity.
The admin-operation-pending flag is removed when

- the management-lost readiness flag has been removed, and
- the PLM Service has performed the administrative operation, or the operator has successfully executed the `SA_PLM_ADMIN_REMOVED` administrative operation.

⇒ **isolate-pending**

This flag is used together with the management-lost readiness flag, as has been explained previously. It indicates that the PLM Service was not able to issue the actions necessary to isolate the entity or could not determine whether the actions were completed on the entity due to circumstances for which the management-lost readiness flag was set for the entity.
The isolate-pending flag is removed when

- the management-lost readiness flag has been removed, and
- the PLM Service has isolated the entity, or the operator has successfully executed the `SA_PLM_ADMIN_REMOVED` administrative operation.

3.1.3.2 EE States

3.1.3.2.1 EE Presence State

The **presence state** of the EE represents its life-cycle. Possible values are instantiated, uninstantiated, terminating, instantiating, instantiation-failed, and termination-failed.

⇒ **uninstantiated**

The presence state of an EE is set to uninstantiated when the operating system or other software that provides the operating environment for that EE is not executing. Typically, this will be due to one or more of the following conditions:

- the administrative state of the EE is locked-instantiation;
- the operational state of the EE is disabled, so PLM has isolated the EE (the meaning of EE isolation is described in Section 3.1.6 on page 52 in detail);
- the readiness state of an ancestor or dependency object is out-of-service.

PLM also needs to set the presence state of the EE to uninstantiated if the management-lost readiness flag was set for the entity, and the operator issued an `SA_PLM_ADMIN_REMOVED` administrative operation (see Section 5.4.11).

⇒ **instantiating**

This value is used as a transitional state when an EE was uninstantiated and should become instantiated.

For example, the following situations may be reflected by this state:

- The parent HE just entered the active presence state, so the EE starts booting.

- The administrative state of the EE was changed from locked-instantiation to locked, allowing the EE to boot.
- The administrative state of the EE was changed from locked to unlocked and the implementation requires a reboot of the operating system.
- The readiness state of the EE or of one of its ancestors or a dependency changed, allowing the presence state of the EE to transition from uninstantiated to instantiated.
- The EE or one of its ancestors was restarted by the RESTART or RESET administrative operations, and the boot process for the EE started.

⇒ **instantiated**

The presence state of an EE is set to instantiated when the start-up of the EE completed, and the EE is able to provide its service.

⇒ **instantiation-failed**

If an EE cannot be instantiated within the configured time (specified by the `saPlmEEInstantiateTimeout` attribute of the `SaPlmEE` object class, shown in FIGURE 7 on page 111), the presence state of the EE is set to instantiation-failed, and its operational state is set to disabled.

⇒ **terminating**

The presence state of an EE is set to terminating when the operating system (or other software that provides the operating environment for that EE) was executing and is now in the process of stopping its execution. This value is used as a transitional state when an EE was instantiated and transitions to uninstantiated.

For example, the following situations may be reflected by this state:

- The parent HE just entered the deactivating presence state, and track users have accepted the deactivation request, so PLM starts to terminate all services and also terminates the EE.
- The administrative state of the EE was changed from locked to locked-instantiation, so PLM starts to terminate the EE.
- A LOCK administrative operation was issued on one of the ancestors, and track users have accepted the request, or the forced option was given, so PLM starts to terminate all services and will also eventually terminate the EE.
- A LOCK administrative operation on an entity upon which the EE depends will cause that fewer than `saPlmDepMinNumber` entities of one of the EE's dependencies will be in-service.

- The readiness state of the EE, of one of its ancestors, or of an entity upon which it depends changed to out-of-service (such that in the latter case fewer than `saPlmDepMinNumber` entities of one of the EE's dependencies will be in-service), so the EE must be terminated.
- A SHUTDOWN administrative operation was issued on the EE or on one of its ancestors, and all services have been terminated, so PLM will start to terminate the EE itself.
- PLM detects that the EE unexpectedly starts its termination process.

⇒ **termination-failed**

If an EE is not able to successfully terminate within the configured time (specified by the `saPlmEETerminateTimeout` attribute of the `SaPlmEE` object class, shown in FIGURE 7 on page 111), the presence state of the EE is set to termination-failed, and its operational state is set to failed.

3.1.3.2.2 Administrative State

As for other AIS Services, the PLM Service defines the administrative state of an EE using an extension of the administrative state specified by ITU (see [11]). Possible values for the **administrative state** of an EE are locked, unlocked, locked-instantiation, and shutting-down.

The administrative state for an EE has the following meaning:

⇒ **unlocked**

The EE has not been directly prohibited by the administrator from providing service.

⇒ **locked**

The administrator has prevented the EE from providing service.
The EE itself may be up and running, but none of its contained objects (for instance, child-EEs and cluster nodes) are allowed to run.
Software diagnostics could be started on the EE, but configured applications and AIS Services other than PLM are not allowed to execute.

⇒ **locked-instantiation**

The administrator has forced the EE itself to be not running at all.

In case of an EE running directly on an HE, the appropriate means may be to assert reset state of that EE.

In case of an EE running on a virtual machine, the PLM Service may use the virtual machine monitor (VMM) to prevent the EE from executing.

⇒ **shutting-down**

The administrator has prevented the execution environment and its contained and depending entities from providing service to new users. The execution environment's administrative state becomes locked as soon as all services the execution environment provides as well as all services provided by its contained and depending entities become unused by all their current users. PLM provides the API for the coordination with its clients (see Section 3.1.7).

3.1.3.2.3 Operational State

The PLM Service uses the operational state as in other AIS Services, which is different from the way ITU uses the state with the same name. As defined by the PLM Service, the **operational state** indicates whether or not an entity is faulty. Possible values are enabled and disabled.

⇒ **enabled**

The EE is healthy and capable of executing the intended software. PLM is not aware of any failure conditions that would prevent any intended use.
The operational state of an EE transitions from disabled to enabled when a successful repair action has been performed on the EE. Repair actions are reported to PLM with the `saPlmEntityReadinessImpact()` interface (see Section 3.5.3.1) or with the `SA_PLM_ADMIN_REPAIRED` administrative operation (see Section 5.4.10). The operational state may transition from disabled to enabled without a repair action being reported if the EE becomes instantiated again.

⇒ **disabled**

The operational state of an EE transitions to disabled if a failure condition is detected.

PLM detects failure conditions of EEs in different ways:

- PLM can detect EE failure conditions by EE-specific means.
- PLM declares an EE as failed if the EE's presence state was set to instantiation-failed because the EE was unable to start up successfully.
  PLM also declares an EE as failed if the EE's presence state was set to termination-failed because the EE was unable to terminate successfully.

- PLM users can detect failure conditions and report these conditions by invoking the `saPlmEntityReadinessImpact()` function (see Section 3.5.3.1).

3.1.3.2.4 Readiness State

As in other AIS Services, the readiness state summarizes values of a set of states. Possible values are in-service, out-of-service, and stopping.

The **readiness state** indicates whether the EE provides its service. This state collects information from all other states, from the ancestor objects, and from mandatory dependencies.

⇒ **out-of-service**

The entity represented by the EE object does not provide service due to one or more of the following conditions:

- Its administrative state is locked or locked-instantiation.
- Its EE presence state is neither instantiated nor terminating.
- Its operational state is disabled.
- The readiness state of any ancestor is out-of-service.
- Fewer than the `saPlmDepMinNumber` entities of a dependency are in-service or stopping.

In case of the last two conditions, the readiness flag “dependency” is also set.

⇒ **in-service**

The readiness state of an EE is in-service if the entity can provide its service. So all of the following conditions must be met:

- Its administrative state is unlocked.
- Its EE presence state is instantiated or terminating.
- Its operational state is enabled.
- The readiness states of all ancestors are in-service.
- For each of its dependencies, at least as many entities are in-service as the required number specified by `saPlmDepMinNumber`.

⇒ **stopping**

The readiness state of an HE transitions from in-service to stopping when one of the following conditions occur:

- Its administrative state is set to shutting-down.
- The readiness state of an ancestor is set to stopping.
- the readiness state of an entity in a dependency is stopping, and the number of remaining in-service entities is lower than `saPlmDepMinNumber` while still `saPlmDepMinNumber` entities of that dependencies are not out-of-service.

The readiness state cannot transition from out-of-service to stopping.

3.1.3.2.5 Readiness Flags

The **readiness flags** complement the readiness state of an object by providing additional information. A flag is defined for each of the following situations:

⇒ **management-lost**

The PLM Service has management capability over an entity when the PLM Service is able to monitor and control the entity sufficiently to accurately model the entity via an HE or EE object that represents the entity. When the PLM Service loses some or all of its management capabilities such that this is not possible, the PLM Service sets the management-lost readiness flag for the entity to reflect this situation. In this case, the value of the operational, administrative, presence, and readiness states of the HE or EE object may or may not reflect the actual state of the entity.

The following applies when the management-lost readiness flag is set for the entity:

- Its administrative state may vary as a consequence of an administrative operation. If the PLM Service processes an administrative operation on the entity and attempts to change the administrative state of the entity but cannot perform all the necessary actions to apply the administrative state or cannot determine whether all these actions were completed, the administrative state of the entity is set to the intended value, and the PLM Service additionally sets the admin-operation-pending readiness flag for the entity.
  The PLM Service returns `SA_AIS_ERR_DEPLOYMENT` error code to the corresponding administrative operation on the entity.
- Its operational state may vary as a consequence of a failure detected by the PLM Service or reported by a PLM user with the invocation of the `saPlmEntityReadinessImpact()` function (see Section 3.5.3.1). If the failure of the entity is detected, and the PLM Service is not able to isolate the failed entity, the flag isolate-pending is additionally set. This flag is cleared by the PLM Service when it regains its management capability for the entity, or if

the failure is cleared by an invocation of the `saPlmEntityReadinessImpact()` function (see Section 3.5.3.1) or by the execution of the `SA_PLM_ADMIN_REPAIRED` administrative operation (see Section 5.4.10).

- Its readiness state may vary as a consequence of changes in its administrative or operational states or as a consequence of changes in the readiness state of entities it depends upon.
- Its presence state is the last value known by the PLM Service, that is, the value of the presence state before the management-lost readiness flag was set for the entity.

When the PLM Service regains its capability to monitor and control the state of an entity for which the management-lost readiness flag was set, the PLM Service automatically clears this flag and updates the value of the operational, presence, and readiness states of the entity to reflect the current state of the entity. If the admin-operation-pending flag is set, the PLM Service must perform the pending administrative operation and must clear the flag. If the isolate-pending flag is set, and the operational state of the entity is still disabled, the PLM Service must isolate the failed entity.

When the operator issues an `SA_PLM_ADMIN_REMOVED` administrative operation (see Section 5.4.11) on the EE, PLM changes the presence state of the EE to uninstantiated and clears all readiness flags.

⇒ **dependency**

The dependency flag is set for an entity when one of its ancestors is not in-service, or fewer than `saPlmDepMinNumber` entities of one of its dependencies are not in-service. That is, this flag indicates that the entity is not in-service, at least in part due to the readiness state of an ancestor or of an entity upon which it is dependent.

⇒ **imminent-failure**

This flag is set for an entity when its operational state is enabled, and an imminent failure on the entity has been detected by the PLM Service or reported to the PLM Service with the `saPlmEntityReadinessImpact()` function (see Section 3.5.3.1).

The PLM Service clears this flag if the operational state of the entity is disabled or if the imminent-failure condition is cleared. The PLM Service may detect that the imminent-failure condition is cleared by analyzing the hardware state, or a user may report that an imminent-failure condition is cleared by calling

`saPlmEntityReadinessImpact()` or by issuing the `SA_PLM_ADMIN_REPAIRED` administrative operation; however, if a user reports that an imminent-failure condition is cleared, the flag should remain set if the PLM Service determines that the imminent-failure condition still exists.

⇒ **dependency-imminent-failure**

This flag is set for an entity when its operational state is enabled, but the imminent-failure or dependency-imminent-failure readiness flag is set for an ancestor or for enough entities upon which this entity depends, so that if they failed would cause the failure of this entity. In other words, this flag indicates that this entity is at risk of failure due to the failure of other entities that are known to be at risk of imminent-failure.

This flag is cleared if the operational state of the entity becomes disabled, or if the imminent-failure and dependency-imminent-failure readiness flags are cleared for all ancestor entities, and at least `saPlmDepMinNumber` entities that are not out-of-service in each of its dependencies.

⇒ **admin-operation-pending**

This flag is used together with the management-lost readiness flag, as has been explained previously. It indicates that the PLM Service was not able to perform all the actions necessary to apply the administrative state or could not determine whether all the actions were completed on the entity because the management-lost readiness flag was set for the entity.
The admin-operation-pending flag is removed when

- the management-lost readiness flag has been removed, and
- the PLM Service has applied the administrative state, or the operator has successfully executed the `SA_PLM_ADMIN_REMOVED` administrative operation.

⇒ **isolate-pending**

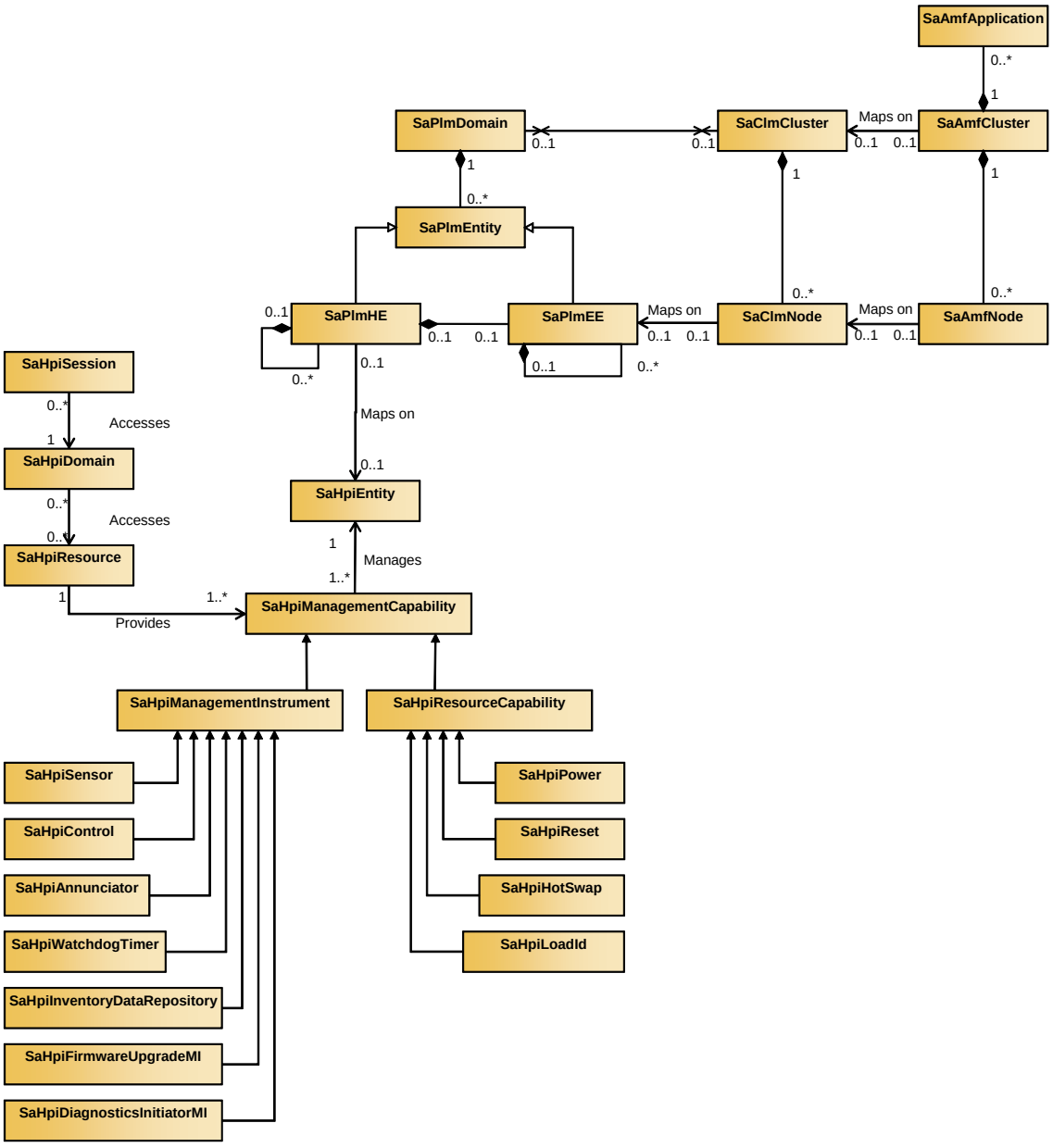
This flag is used together with the management-lost readiness flag, as has been explained previously. It indicates that the PLM Service was not able to issue the actions necessary to isolate the entity or could not determine whether the actions were completed on the entity because the management-lost readiness flag was set for the entity.
The isolate-pending flag is removed when

- the management-lost readiness flag has been removed, and
- the PLM Service has isolated the entity, or the operator has successfully executed the `SA_PLM_ADMIN_REMOVED` administrative operation.

3.1.3.3 Mapping Between PLM and HPI Objects

FIGURE 2 illustrates how PLM objects and objects of other SA Forum Services map to each other. Note that the multiplicities for HPI object classes below `SaHpiManagementCapability` are not shown in the figure. For details on HPI classes and objects, refer to [3].

FIGURE 2 Mapping Between PLM Objects and Objects of Other SA Forum Services



Each HE object is mapped to an HPI entity. In HPI, the entity may have multiple management capabilities. Each of these management capabilities is accessed through a specific resource, and that resource is a member of zero or more domains. The PLM Service will typically perform an HPI discovery operation, as described in [3], and build a table of available management capabilities, organized by physical entity. This table should note, for each capability, which HPI domain and resource to use to access that capability for the entity.

3.1.3.4 Recommendation for HE Modeling

When system architects design a system, they need to create HE types for the different levels of the hardware architecture. They also need to decide in which granularity hardware is visible for software in the information model. Typically, every field replaceable unit (FRU) is modeled as a separate HE object.

The PLM Service requires that an HE have at most one EE as a child. This is the EE that is booted when an HE starts up. If the EE runs a virtual machine monitor, then additional EEs for the virtual machines are modeled as children of that EE, rather than as direct children of the HE.

It is strongly recommended that every hardware entity that should be managed as a redundancy unit be modeled as an HE. That is, if there are maintenance procedures to exchange hardware units, the PLM Service can reflect the maintenance procedures to application software only if those FRUs are separately modeled as HEs. Thus, the PLM Service also provides the administrative operations to support such maintenance procedures.

This recommendation is also valid for HEs that are not parent to an EE. Even entities like power supplies or fans, which do not directly contribute to the computing capabilities of a system, should be modeled as separate HEs if they are FRUs. When entities are not modeled as HEs, the PLM Service must map alarms or notifications associated with those entities to an HE associated with an entity containing the affected entity; PLM should choose the HE closest to the entity in the containment tree.

System architects can decide to model with a finer granularity than the granularity of FRUs. With this modeling, the PLM Service can do a more detailed mapping of hardware alarms and other events.

The `SaPlmHEBaseType` object class can be used to represent the functionality of the HE from a user's perspective, and the `SaPlmHEType` object class reflects the implementation-specific control aspects. In other words, an object of the `SaPlmHEBaseType` class groups together different objects of the `SaPlmHEType`

class that represent different implementations of the same hardware functionality. Thus, from the perspective of entities of other layers, such as EEs, CLM nodes, or AMF nodes, HE entities referring to different HE types (objects of `SaPlmHEType` class) of the same HE base type (an object of `SaPlmHEBaseType` object class) should be transparently interchangeable.

In addition, at the `SaPlmHEType` level, the vendor should provide enough information, so that PLM can correctly manage the specific hardware that is installed.

3.1.3.5 Hardware Health Monitoring

The PLM Service monitors the **health state** of hardware entities with the help of HPI. HPI uses sensor states to represent the state of the hardware, and some sensors indicate the state of health of the hardware. Changes of sensor states may be reported by HPI using HPI events.

PLM needs to keep track of the relevant sensor states. Therefore, PLM needs to actively read sensor states when a resource giving access to the sensor is added or restored to a domain, or when PLM opens a new session to a domain. PLM needs to analyze HPI events and take appropriate actions like reading sensors or enabling sensor events.

PLM may need some specific sensors to monitor the state of health of the hardware. If those sensors cannot be read, and PLM cannot retrieve the necessary information by other means, PLM must set the management-lost readiness flag for the affected HE.

Typically, a PLM implementation will provide a means to configure how sensor states are evaluated. The user can configure which sensors of a hardware type are important for health checking. The configuration will specify which state values are considered to indicate a hardware failure, an imminent failure, or an alarm for the HE representing the hardware[1]. PLM may enable the related sensor events in HPI or may start polling sensors[2]. When PLM detects a sensor state changing to a value that indicates, for instance, a failure (by HPI event or by reading the sensor), PLM will assume that the hardware entity associated with the hardware entity monitored by that sensor is faulty. There may also be more complex conditions, for instance, when multiple sensors that can indicate failure, imminent failures, or alarm conditions need to be considered together.

1. This configuration is implementation-specific.
2. There may be sensors that do not support events.

3.1.3.6 Other Aspects of Interworking with HPI

HPI provides policies for automatic handling of extraction and insertion of entities supporting the hot swap model. PLM will typically stop the auto insertion and auto extraction policies to gain full control of the insertion and extraction process.

During start-up, the PLM Service needs to discover the present hardware, including its state, and map the present hardware to the configured HEs.

The PLM Service uses HPI events to be notified about changes in the hardware as fast as possible. The PLM Service needs to analyze all HPI events to detect their impact on the system and determine state changes of the configured objects. The PLM Service issues state change notifications (see Section 6.2.2 on page 155), as defined by the Notification Service, see [4].

The PLM Service may additionally provide the capability to issue notifications for all HPI events. These notifications cannot be used directly to indicate the hardware states without detailed analysis. Their only goal is to log all relevant events in a system at one place and output them if needed through the same channel. PLM should provide local suppression for these notifications. For details on the format of these notifications, refer to Section 6.2.3 on page 158.

3.1.4 EE Management

A PLM implementation should provide similar capabilities on EE level as provided on hardware level. However, no standard interface for the management of operating systems and virtual machine monitors is available. Thus, EE management at this point is implementation-specific and will vary for the operating systems and virtual machine monitors that a PLM implementation supports. However, the PLM implementation must provide appropriate administrative operations and implementation-specific actions according to the state changes. For instance, PLM typically needs to control the start-up of an operating system when the hardware element becomes in-service, and it also needs to detect when the EE is instantiated and in-service.

3.1.4.1 Recommendation for EE Modeling

In virtualized architectures, it is recommended that system architects model all virtual machines as separate EEs. In theory, it is also possible to map all software directly to a hypervisor EE without representing each virtual machine in the model. However, in that case, an administrative operation (for instance, restart) cannot be executed by PLM for individual virtual machines.

3.1.4.2 PLM Virtualization Support

Virtualization technologies provide means to aggregate multiple computing elements into a single virtual environment or to partition a computing element to provide multiple virtual environments. For an overview of virtualization architectures that are important for high availability platforms, see [13].

All architectures can be represented by EE objects in the PLM Information Model.

- In case of aggregation, a single EE depends on multiple hardware entities, which may be represented by HEs. The system architect could model this case by choosing a reasonable high level HE to be the parent element of the EE. Other dependencies must be modeled using dependency objects.
- In case of partitioning, EE objects are used to represent the virtual machine monitors and also to represent the virtual machines, as described below in more detail.
- For other architectures, even a combination of aggregations and partitioning may be used.

Virtualization is provided by a virtual machine monitor (VMM) or hypervisor, which

- may run directly on the physical hardware (bare-metal hypervisor) or
- requires an operating system to run on.

Usually, the VMM provides a set of virtual machines (VM). Every VM can run an operating system.

As explained above, the PLM Service uses EE objects in its information model to represent the VMM as well as the VMs. These object classes reflect the architecture of VMM and VMs in the information model and also enable the PLM Service to manage the corresponding entities.

There are several possible types of virtualization architectures.

- A VMM can directly run on the hardware element and provide VMs for operating systems running under its control.
- A VMM can be integrated in an operating system, allowing VMs to run in parallel with normal processes.
- A VMM can run as an application under control of an operating system and still provide VMs that run child operating systems.

The diagram in FIGURE 3 illustrates these architectures and how they are represented in the information model:

FIGURE 3 Virtualized Architectures in the PLM Information Model

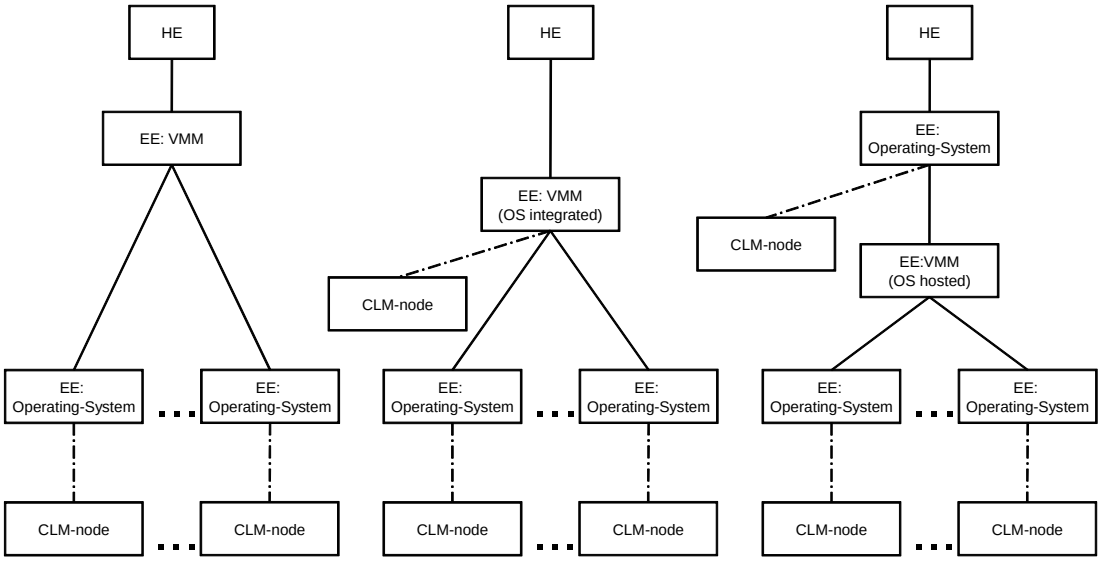


FIGURE 3 also shows the possible CLM nodes. Note that an EE does not necessarily host CLM nodes. Note also that CLM nodes are not nested, that is, every CLM node runs directly on an EE. PLM uses interfaces specific to the operating systems and to VMM implementations to provide management for the EE objects.

PLM thus hides the proprietary interfaces of VMMs and operating systems from AIS middleware and higher layers. PLM reflects all changes in the virtual machines in its information model and notifies its users about the changes using the track callback interface. That way, AIS middleware configuration can be kept independent from the VMM implementation and still be aware of changes in the cluster architecture.

At the same time, the information model shows dependencies that are important for high availability. For instance, CLM nodes that run on the same HE, should not host active and standby service units for the same service instance (for details, see [9]). However, it is the task of the system configurator or of a configuration application to avoid this situation. PLM only provides information about which EEs share the same hardware or have dependencies to the same hardware entities.

3.1.5 Verification of the System Configuration

3.1.5.1 Verification of the Hardware Configuration

When attempting to match a hardware element with a hardware entity, the PLM Service performs the following actions:

- It checks that the hardware entity is located in one of the potential locations that have been configured for the hardware element. The entity path of the hardware entity is used by the PLM Service to perform this check.
- It checks that the characteristics of the hardware entity match the configured characteristics of the hardware element. The entity type of the HPI entity and the contents of its Inventory Data Repositories (IDR) are used by the PLM Service to perform this check.

For more details, refer to Section 4.5.1 on page 106.

If there is no configured HE object for a hardware entity that is present in the system, the PLM Service will ignore this entity. However, the PLM Service may issue the Unmapped Hardware Entity Alarm, which is described in Section 6.2.1.5 on page 153.

3.1.5.2 Verification of Execution Environments

The various attributes of the EE types and base types are used to validate the installed and automatically booted operating system or virtualization monitor against the configuration. These attributes include:

- Vendor name
- Product name
- Release
- Version

It is implementation-specific how the PLM Service uses these attributes to match an execution environment with a particular operating system or virtualization monitor.

Details on the EE type (`SaPlmEEType`) and EE base type (`SaPlmEEBaseType`) object classes are presented in Section 4.6.

If there is no configured EE object matching an execution environment that is present in the system, the PLM Service will ignore this entity.

3.1.6 Isolation of Entities

When a PLM entity is faulty (its operational state is set to disabled), the PLM Service is responsible for **isolating** this entity from the system. An isolated entity cannot provide service and does not impact other parts of the system.

PLM may attempt automatic repair actions on a faulty entity, for instance, restart or reset of the entity or even of its parent. These repair actions are implementation-specific.

Isolation is done in the same way for the following causes:

- Call to the `saPlmEntityReadinessImpact()` function (see Section 3.5.3.1).
- Analysis of HPI events
- Analysis of hardware states
- EE health monitoring

PLM must attempt to isolate the entity before PLM notifies track users about the failure of the entity. If PLM fails to isolate the entity because the management-lost readiness flag is set for the entity, the isolate-pending flag is set in the readiness flag of the entity.

PLM informs track users in the completed step of the track interface about the isolation or pending isolation.

Isolation is not only applicable to faulty entities, PLM also provides an administrative operation to allow the operator to isolate an entity. Hardware elements are isolated by the `SA_PLM_ADMIN_DEACTIVATE` administrative operation (see Section 5.4.7), and execution environments are isolated by the `SA_PLM_ADMIN_LOCK_INSTANTIATION` operation (see Section 5.4.4).

Isolation of HEs:

When PLM isolates a hardware element, it forces the hardware element's presence state to become inactive. The actions taken depend on the hardware capabilities. PLM will take actions such as the following ones:

- Typically, if the hardware element provides managed hot swap capabilities, PLM isolates the hardware element by hot swap management and forces the hardware element to be set inactive (HPI hot swap state typically includes power off).

- Hardware elements that do not support managed hot swap may still support power management. PLM usually isolates these elements by powering them down.
- On hardware elements that neither support hot swap nor power management, PLM may attempt to assert the reset state.
- For hardware elements that do not support the aforementioned measures, PLM may attempt other implementation-specific actions to isolate the entity.

If the hardware element hosts an EE, at least the EE must be terminated, that is, its presence state must change to uninstantiated when the presence state of the hardware element goes to inactive.

Isolation of EEs that are Directly Hosted by an HE

Again, isolation depends on the capabilities. At least the EE must be terminated and its presence state set to uninstantiated. If it is not possible to terminate the EE, PLM may assert reset state on the HE or isolate the HE.

Isolation of EEs Running in a Virtual Machine

Again, isolation depends on the capabilities. At least the EE must be terminated and its presence state set to uninstantiated.

- PLM may use the hypervisor to terminate the EE.
- If it is not possible to terminate the EE, and other EEs on that HE are still active, PLM should not isolate the HE.

3.1.7 Overview of the PLM Interfaces

The PLM Service provides the following types of interfaces to its users, which can be applications, AIS Services, and management applications:

- **Track Interface**

  The track interface allows users to subscribe and be notified when the readiness status (that is, readiness state or readiness flags) of an entity changes.

  PLM users can subscribe for groups of PLM objects and select the cases when they are notified.

  Administrative operations on PLM objects may affect several services, and—in many cases—an operator does not know how the locking of a PLM entity or the extraction of some hardware will impact the services being provided. To allow users of the PLM track API to reject a particular operation or to relocate some services before the operation is performed, the PLM track interface notifies its users in several steps.

  The track interface provides four options:

  - **Validate**: subscribed users are asked to validate, that is, to accept or reject the operation that will cause the change.
  - **Start**: subscribed users should now take appropriate actions before the entity is locked or deactivated. They can, for instance, relocate their services.
  - **Completed**: subscribed users are notified that the operation has been performed.
  - **Aborted**: subscribed users are notified that the operation was rejected during the validate step.

  When calling the `saPlmReadinessTrack()` function, subscribers can choose for which of the above steps they want to be notified.

  For more details, refer to Section 3.5.2 on page 85. Operational scenarios illustrating how these options can be used are included in Appendix B on page 167.

- **Notifications**

  PLM users that need more detailed information about state changes of PLM entities can subscribe for notifications using the NTF Service (see [4]). For details, refer to Section 6.2.2 on page 155.

- **Administrative Operations**

  The PLM Service provides administrative operations to manage the administrative state of its objects.

  Additionally, administrative operations provide a means to restart an EE or reset an HE. This interface can, for instance, be used during AMF repair procedures.

  All administrative operations are described in Section 5.4 on page 115.

- **Error Reporting**

  Not all errors on PLM entities can be detected by PLM itself. AIS Services and applications can also detect errors. The `saPlmEntityReadinessImpact()` interface is provided to report errors (see Section 3.5.3.1).

3.1.8 PLM Service and Cluster Membership

The PLM Service has no knowledge about cluster membership. It provides its service independently from the Cluster Membership Service.

3.2 Include File and Library Names

The following statement containing declarations of data types and function prototypes must be included in the source of a process using the PLM Service API:

```
#include <saPlm.h>
```

To use the PLM Service API, a process must be bound with the following library:

```
libSaPlm.so
```

3.3 Type Definitions

The APIs of the PLM Service use the types described in the following sections.

3.3.1 PLM Handles

3.3.1.1 SaPlmHandleT

```
typedef SaUint64T SaPlmHandleT;
```

The `SaPlmHandleT` type is used for the handle to the PLM Service. A process acquires this handle by invoking `saPlmInitialize()` and uses it in subsequent invocations of the functions of the PLM Service.

3.3.1.2 SaPlmEntityGroupHandleT

```
typedef SaUint64T SaPlmEntityGroupHandleT;
```

The `SaPlmEntityGroupHandleT` type is used for the handle to a group of PLM entities tracked through `saPlmReadinessTrack()`.

3.3.2 HE Administrative State

```
typedef enum {
     SA_PLM_HE_ADMIN_UNLOCKED             = 1,
     SA_PLM_HE_ADMIN_LOCKED               = 2,
     SA_PLM_HE_ADMIN_LOCKED_INACTIVE      = 3,
     SA_PLM_HE_ADMIN_SHUTTING_DOWN        = 4
} SaPlmHEAdminStateT;
```

The `SaPlmHEAdminStateT` type is used to represent the administrative state of a PLM hardware element.

3.3.3 EE Administrative State

```
typedef enum {
     SA_PLM_EE_ADMIN_UNLOCKED              = 1,
     SA_PLM_EE_ADMIN_LOCKED                = 2,
     SA_PLM_EE_ADMIN_LOCKED_INSTANTIATION  = 3,
     SA_PLM_EE_ADMIN_SHUTTING_DOWN         = 4
} SaPlmEEAdminStateT;
```

The `SaPlmEEAdminStateT` type is used to represent the administrative state of a PLM execution environment.

3.3.4 Operational State

```
typedef enum {
     SA_PLM_OPERATIONAL_ENABLED  = 1,
     SA_PLM_OPERATIONAL_DISABLED = 2
} SaPlmOperationalStateT;
```

The `SaPlmOperationalStateT` type is used to represent the operational state of a PLM entity.

3.3.5 HE Presence State

```
typedef enum {
     SA_PLM_HE_PRESENCE_NOT_PRESENT        = 1,
     SA_PLM_HE_PRESENCE_INACTIVE           = 2,
     SA_PLM_HE_PRESENCE_ACTIVATING         = 3,
     SA_PLM_HE_PRESENCE_ACTIVE             = 4,
     SA_PLM_HE_PRESENCE_DEACTIVATING       = 5
} SaPlmHEPresenceStateT;
```

The `SaPlmHEPresenceStateT` type is used to represent the presence state of a PLM hardware element.

3.3.6 EE Presence State

```
typedef enum {
     SA_PLM_EE_PRESENCE_UNINSTANTIATED          = 1,
     SA_PLM_EE_PRESENCE_INSTANTIATING           = 2,
     SA_PLM_EE_PRESENCE_INSTANTIATED            = 3,
     SA_PLM_EE_PRESENCE_TERMINATING             = 4,
     SA_PLM_EE_PRESENCE_INSTANTIATION_FAILED    = 5,
     SA_PLM_EE_PRESENCE_TERMINATION_FAILED      = 6
} SaPlmEEPresenceStateT;
```

The `SaPlmEEPresenceStateT` type is used to represent the presence state of a PLM execution environment.

3.3.7 Readiness State

```
typedef enum {
     SA_PLM_READINESS_OUT_OF_SERVICE  = 1,
     SA_PLM_READINESS_IN_SERVICE      = 2,
     SA_PLM_READINESS_STOPPING        = 3
} SaPlmReadinessStateT;
```

The `SaPlmReadinessStateT` type is used to represent the readiness state of a PLM entity.

3.3.8 Readiness Flags

```
#define SA_PLM_RF_MANAGEMENT_LOST              0x00001
#define SA_PLM_RF_ADMIN_OPERATION_PENDING      0x00002
#define SA_PLM_RF_ISOLATE_PENDING              0x00004
#define SA_PLM_RF_DEPENDENCY                   0x00100
#define SA_PLM_RF_IMMINENT_FAILURE             0x00200
#define SA_PLM_RF_DEPENDENCY_IMMINENT_FAILURE  0x00400
typedef SaUint64T SaPlmReadinessFlagsT;
```

The `SaPlmReadinessFlagsT` type complements the readiness state of an entity by providing additional information regarding the readiness status of the entity.

3.3.9 Readiness Status

```
typedef struct {
     SaPlmReadinessStateT readinessState;
     SaPlmReadinessFlagsT readinessFlags;
} SaPlmReadinessStatusT;
```

The `SaPlmReadinessStatusT` type holds both the readiness state and readiness flags of a PLM entity.

3.3.10 Readiness Impact

```
typedef enum {
     SA_PLM_RI_FAILURE                    = 1,
     SA_PLM_RI_IMMINENT_FAILURE           = 2,
     SA_PLM_RI_FAILURE_CLEARED            = 101,
     SA_PLM_RI_IMMINENT_FAILURE_CLEARED   = 102
} SaPlmReadinessImpactT;
```

The `SaPlmReadinessImpactT` type is used to report an event that affects the readiness status (readiness state and readiness flags) of a PLM entity. The values of `SaPlmReadinessImpactT` have the following interpretation:

- `SA_PLM_RI_FAILURE`
  This value is used to report the failure of a PLM entity.
- `SA_PLM_RI_IMMINENT_FAILURE`
  This value is used to report an imminent failure of a PLM entity.
- `SA_PLM_RI_FAILURE_CLEARED`
  This value is used to report that the failure of a PLM entity has been repaired.
- `SA_PLM_RI_IMMINENT_FAILURE_CLEARED`
  This value is used to report that the imminent failure of a PLM entity has been cleared.

3.3.11 HE Deactivation Policy

```
typedef enum {
     SA_PLM_DP_REJECT_NOT_OOS          = 1,
     SA_PLM_DP_VALIDATE                = 2,
     SA_PLM_DP_UNCONDITIONAL           = 3
} SaPlmHEDeactivationPolicyT;
```

This type is used to configure the PLM policy used in conjunction with a graceful deactivation of hardware elements.

The values of `SaPlmHEDeactivationPolicyT` have the following interpretation:

- `SA_PLM_DP_REJECT_NOT_OOS`
  If the readiness state of the HE to be deactivated is not `SA_PLM_READINESS_OUT_OF_SERVICE`, PLM rejects the deactivation.
- `SA_PLM_DP_VALIDATE`
  If the readiness state of the HE to be deactivated is not `SA_PLM_READINESS_OUT_OF_SERVICE`, PLM relies on its clients that track (through `saPlmReadinessTrack()`) the readiness status of this entity or of entities that depend on it to validate during the `SA_PLM_CHANGE_VALIDATE` step whether the deactivation can proceed.
- `SA_PLM_DP_UNCONDITIONAL`
  If the readiness state of the HE to be deactivated is not `SA_PLM_READINESS_OUT_OF_SERVICE`, PLM notifies its clients that track (through `saPlmReadinessTrack()`) the readiness status of this entity or of entities that depend on it that the deactivation will occur (`SA_PLM_CHANGE_START` step).

3.3.12 Entity Groups

```
typedef enum {
     SA_PLM_GROUP_SINGLE_ENTITY       = 1,
     SA_PLM_GROUP_SUBTREE             = 2,
     SA_PLM_GROUP_SUBTREE_HES_ONLY    = 3,
     SA_PLM_GROUP_SUBTREE_EES_ONLY    = 4
} SaPlmGroupOptionsT;
```

The `SaPlmGroupOptionsT` type is used by the `saPlmEntityGroupAdd()` function. The values of `SaPlmGroupOptionsT` have the following interpretation:

- `SA_PLM_GROUP_SINGLE_ENTITY`
  This option is used to indicate that only the entities directly designated by their names in the `entities` array are to be added to the entity group.
- `SA_PLM_GROUP_SUBTREE`
  This option is used to indicate that all entities contained in the subtrees that have as a root the entities designated by their names in the `entities` array are to be added to the entity group.
- `SA_PLM_GROUP_SUBTREE_HES_ONLY`
  This option is used to indicate that all hardware elements contained in the subtrees that have as a root the entities designated by their names in the `entities` array are to be added to the entity group.
- `SA_PLM_GROUP_SUBTREE_EES_ONLY`
  This option is used to indicate that all execution environments contained in the subtrees that have as a root the entities designated by their names in the `entities` array are to be added to the entity group.

3.3.13 State Tracking

The various types defined in this section are used by the PLM APIs that track changes of the readiness status of a group of PLM entities.

3.3.13.1 SaPlmGroupChangesT

```
typedef enum {
     SA_PLM_GROUP_NO_CHANGE                = 1,
     SA_PLM_GROUP_MEMBER_ADDED             = 2,
     SA_PLM_GROUP_MEMBER_REMOVED           = 3,
     SA_PLM_GROUP_MEMBER_READINESS_CHANGE  = 4
} SaPlmGroupChangesT;
```

The `SaPlmGroupChangesT` type reflects the status of a PLM entity that is contained in a track notification array, that is, in the array referred to by the `entities` pointer in a structure of `SaPlmReadinessTrackedEntitiesT` type. The values of `SaPlmGroupChangesT` have the following interpretation:

- `SA_PLM_GROUP_NO_CHANGE`
  The readiness state and flags of the PLM entity have not changed. This value is used when the `trackFlags` parameter of the `saPlmReadinessTrack()` function is
  - either `SA_TRACK_CURRENT`
  - or `SA_TRACK_CHANGES`, and the PLM entity was already a member of the tracked entity group, and none of its readiness state and flags have changed (or are about to change in the case of the `SA_PLM_CHANGE_VALIDATE` or `SA_PLM_CHANGE_START` tracking steps).
- `SA_PLM_GROUP_MEMBER_ADDED`
  The PLM entity has been added to the tracked entity group.
- `SA_PLM_GROUP_MEMBER_REMOVED`
  The PLM entity has been removed from the tracked entity group.
- `SA_PLM_GROUP_MEMBER_READINESS_CHANGE`
  The readiness state or flags of the PLM entity have changed (or are about to change in the case of the `SA_PLM_CHANGE_VALIDATE` or `SA_PLM_CHANGE_START` tracking steps).

3.3.13.2 SaPlmChangeStepT

```
typedef enum {
     SA_PLM_CHANGE_VALIDATE     = 1,
     SA_PLM_CHANGE_START        = 2,
     SA_PLM_CHANGE_ABORTED      = 3,
     SA_PLM_CHANGE_COMPLETED    = 4
} SaPlmChangeStepT;
```

The `SaPlmChangeStepT` type is used to indicate in which step of the readiness change tracking process the track callback is invoked. The values of `SaPlmChangeStepT` have the following interpretation:

- `SA_PLM_CHANGE_VALIDATE`
  The track callback is invoked to allow the invoker to reject the change.
- `SA_PLM_CHANGE_START`
  The change is occurring (it has not been rejected) and the track callback is invoked to let the invoker perform all necessary actions for this change to occur with no service impact.
- `SA_PLM_CHANGE_ABORTED`
  A proposed change has been rejected.
- `SA_PLM_CHANGE_COMPLETED`
  A change in the readiness state or flags of the PLM entity has occurred.

3.3.13.3 SaPlmTrackCauseT

```
typedef enum {
     /* Causes that may trigger all change steps */
     SA_PLM_CAUSE_HE_DEACTIVATION          = 1,
     SA_PLM_CAUSE_LOCK                     = 2,
     /* Causes that may trigger only START and COMPLETED steps */
     SA_PLM_CAUSE_SHUTDOWN                 = 101,
     /* Causes that only trigger a COMPLETED step */
     SA_PLM_CAUSE_GROUP_CHANGE             = 201,
     SA_PLM_CAUSE_MANAGEMENT_LOST          = 202,
     SA_PLM_CAUSE_MANAGEMENT_REGAINED      = 203,
     SA_PLM_CAUSE_FAILURE                  = 204,
     SA_PLM_CAUSE_FAILURE_CLEARED          = 205,
     SA_PLM_CAUSE_IMMINENT_FAILURE         = 206,
     SA_PLM_CAUSE_IMMINENT_FAILURE_CLEARED = 207,
     SA_PLM_CAUSE_UNLOCKED                 = 208,
     SA_PLM_CAUSE_HE_ACTIVATED             = 209,
     SA_PLM_CAUSE_HE_RESET                 = 210,
     SA_PLM_CAUSE_EE_INSTANTIATED          = 211,
     SA_PLM_CAUSE_EE_UNINSTANTIATED        = 212,
     SA_PLM_CAUSE_EE_RESTART               = 213,
     SA_PLM_CAUSE_STATUS_INFO              = 214
} SaPlmTrackCauseT;
```

The `SaPlmTrackCauseT` type is used in the `SaPlmReadinessTrackCallBackT` callback function to indicate the cause of the readiness state of flags changes. The values of `SaPlmTrackCauseT` have the following interpretation:

- `SA_PLM_CAUSE_HE_DEACTIVATION`
  The hardware element designated by `rootCauseEntity` in the `SaPlmReadinessTrackCallBackT` callback is going through a deactivation process. In the case of a graceful deactivation, callbacks with this cause may be invoked at all steps of the process, depending on the configuration of the PLM deactivation policy. In the case of an abrupt deactivation, callbacks will be invoked only at the `SA_PLM_CHANGE_COMPLETED` step. For more details on the deactivation process, see Section 3.1.3.1.1. For the steps used in the `SaPlmReadinessTrackCallBackT` function, refer to Section 3.1.7.

- SA_PLM_CAUSE_LOCK
  The entity designated by `rootCauseEntity` in the `SaPlmReadinessTrackCallBackT` callback is the target of a LOCK administrative operation. Depending on the option selected with that administrative operation, callbacks with this cause may be invoked at all steps of the process. For more details on the LOCK administrative operation, see Section 5.4.2.
- SA_PLM_CAUSE_SHUTDOWN
  The entity designated by `rootCauseEntity` in the `SaPlmReadinessTrackCallBackT` callback is the target of a SHUTDOWN administrative operation. The track callback is invoked during the SA_PLM_CHANGE_START step to force the shutdown process to happen. After responses (successful or not) from all callback invocations have been received, the entity is administratively locked, and the track callback is invoked in the SA_PLM_CHANGE_COMPLETED step. For more details on the SHUTDOWN administrative operation, see Section 5.4.3
- SA_PLM_CAUSE_GROUP_CHANGE
  This cause is used to notify tracking processes about changes in the membership of the tracked entity group. Entities may be dynamically added to the tracked entity group or removed from it by invoking the `saPlmEntityGroupAdd()` or `saPlmEntityGroupRemove()` functions, by configuring new entities within a tracked subtree, or by removing tracked entities from the configuration. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step, and the `rootCauseEntity` parameter is set to NULL.
- SA_PLM_CAUSE_MANAGEMENT_LOST
  The PLM Service has lost its management capabilities for one or several entities. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step, and the `rootCauseEntity` parameter is set to NULL.
- SA_PLM_CAUSE_MANAGEMENT_REGAINED
  The PLM Service has regained its management capabilities for one or several entities. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step, and the `rootCauseEntity` parameter is set to NULL.
- SA_PLM_CAUSE_FAILURE
  The entity designated by `rootCauseEntity` in the `SaPlmReadinessTrackCallBackT` callback has failed. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step.
- SA_PLM_CAUSE_FAILURE_CLEARED
  A failure on the entity designated by `rootCauseEntity` in the

SaPlmReadinessTrackCallBackT callback has been cleared. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step.

- SA_PLM_CAUSE_IMMINENT_FAILURE
  An imminent failure has been detected for the entity designated by rootCauseEntity in the SaPlmReadinessTrackCallBackT callback. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step.
- SA_PLM_CAUSE_IMMINENT_FAILURE_CLEARED
  An imminent failure on the entity designated by rootCauseEntity in the SaPlmReadinessTrackCallBackT callback has been cleared. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step.
- SA_PLM_CAUSE_UNLOCKED
  The entity designated by rootCauseEntity in the SaPlmReadinessTrackCallBackT callback has been administratively unlocked. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step.
- SA_PLM_CAUSE_HE_ACTIVATED
  The presence state of the hardware element designated by rootCauseEntity in the SaPlmReadinessTrackCallBackT callback has changed to SA_PLM_HE_PRESENCE_ACTIVE. As a consequence, the readiness state of this hardware element became in-service. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step.
- SA_PLM_CAUSE_HE_RESET
  The hardware element designated by rootCauseEntity in the SaPlmReadinessTrackCallBackT callback was reset due to an SA_PLM_ADMIN_RESET administrative operation. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step.
- SA_PLM_CAUSE_EE_INSTANTIATED
  The presence state of the execution environment designated by rootCauseEntity in the SaPlmReadinessTrackCallBackT callback has changed to SA_PLM_EE_PRESENCE_INSTANTIATED. As a consequence, the readiness state of this execution environment became in-service. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step.
- SA_PLM_CAUSE_EE_UNINSTANTIATED
  The presence state of the execution environment designated by rootCauseEntity in the SaPlmReadinessTrackCallBackT callback has changed to SA_PLM_EE_PRESENCE_UNINSTANTIATED. As a consequence, the readiness state of this execution environment became out-of-service. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step.

- SA_PLM_CAUSE_EE_RESTART
  The execution environment designated by `rootCauseEntity` in the `SaPlmReadinessTrackCallBackT` callback was restarted due to an SA_PLM_ADMIN_RESTART administrative operation. For this cause, the callback is only invoked in the SA_PLM_CHANGE_COMPLETED step.
- SA_PLM_CAUSE_STATUS_INFO
  The value is used in the `SaPlmReadinessTrackCallBackT` callback if the callback was triggered by specifying the flag SA_TRACK_CURRENT in the call to `saPlmReadinessTrack`. In this case, the callback is invoked in the SA_PLM_CHANGE_COMPLETED step, and the `rootCauseEntity` parameter is set to NULL.

3.3.13.4 SaPlmReadinessTrackedEntityT

```
typedef struct {
     SaPlmGroupChangesT change;
     SaNameT entityName;
     SaPlmReadinessStatusT currentReadinessStatus;
     SaPlmReadinessStatusT expectedReadinessStatus;
     SaNtfIdentifierT plmNotificationId;
} SaPlmReadinessTrackedEntityT;
```

The `SaPlmReadinessTrackedEntityT` type is used to return a descriptor of a tracked entity designated by its name, `entityName`.
The `change` field indicates the kind of change being reported.
The `currentReadinessStatus` field contains the current readiness status of the entity. When the track callback is invoked in the context of the SA_PLM_CHANGE_VALIDATE or SA_PLM_CHANGE_START step, the `expectedReadinessStatus` field provides the expected value of the readiness status that the entity will have if the action is completed.
When the track callback is invoked in the context of the SA_PLM_CHANGE_COMPLETED or SA_PLM_CHANGE_ABORTED step, both `expectedReadinessStatus` and `currentReadinessStatus` are set to the current value of the readiness status.
The `plmNotificationId` field is the identifier of the first NTF notification the PLM Service has sent (or will send) to notify the readiness status change of the tracked entity.

3.3.13.5 SaPlmReadinessTrackedEntitiesT

```
typedef struct {
     SaUint32T numberOfEntities;
     SaPlmReadinessTrackedEntityT *entities;
} SaPlmReadinessTrackedEntitiesT;
```

The SaPlmReadinessTrackedEntitiesT type is used in the saPlmReadinessTrack() function and in the SaPlmReadinessTrackCallBackT callback to return a set of descriptors for tracked entities.
numberOfEntities holds the number of SaPlmReadinessTrackedEntitiesT descriptors in the descriptor array pointed to by entities.

3.3.14 Callback Response

```
typedef enum {
     SA_PLM_CALLBACK_RESPONSE_OK           = 1,
     SA_PLM_CALLBACK_RESPONSE_REJECTED     = 2,
     SA_PLM_CALLBACK_RESPONSE_ERROR        = 3
} SaPlmReadinessTrackResponseT;
```

The SaPlmReadinessTrackResponseT type is used by the saPlmReadinessTrackResponse() function to provide the response to a callback previously invoked by the PLM Service for an SA_PLM_CHANGE_VALIDATE or SA_PLM_CHANGE_START tracking step. The values of SaPlmReadinessTrackResponseT have the following interpretation:

- SA_PLM_CALLBACK_RESPONSE_OK
  When provided as a response to a readiness track callback invocation in the SA_PLM_CHANGE_VALIDATE step, this response indicates that the process accepts the pending operation (graceful HE deactivation or SA_PLM_ADMIN_LOCK administrative operation with the trylock option).
  When provided as a response to a readiness track callback invocation in the SA_PLM_CHANGE_START step, this response indicates either that
  - the process is ready for the pending operation to be executed (graceful HE deactivation or SA_PLM_ADMIN_LOCK administrative operation with the trylock option or with no option), or that
  - the quiescing triggered by a SA_PLM_ADMIN_SHUTDOWN administrative operation is now completed by the calling process.

- SA_PLM_CALLBACK_RESPONSE_REJECTED
  This response is valid only as a response to a readiness track callback invocation in the SA_PLM_CHANGE_VALIDATE step, and it indicates that the process rejects the pending operation (graceful HE deactivation or SA_PLM_ADMIN_LOCK administrative operation with the `trylock` option).
- SA_PLM_CALLBACK_RESPONSE_ERROR
  When provided as a response to a readiness track callback invocation, this response indicates that the process encountered an error and is not able to provide a meaningful response to the callback invocation. As the PLM Service waits for responses from all processes invoked with a readiness track callback with the SA_PLM_CHANGE_VALIDATE or SA_PLM_CHANGE_START steps, such processes must provide a timely response even if they encounter an error that prevents them from completing the callback processing.

3.3.15 Notification Related Types

3.3.15.1 SaPlmNotificationMinorIdT

```
typedef enum {
     SA_PLM_NTFID_HE_ALARM           = 0x01,
     SA_PLM_NTFID_EE_ALARM           = 0x02,
     SA_PLM_NTFID_HE_SEC_ALARM       = 0x03,
     SA_PLM_NTFID_EE_SEC_ALARM       = 0x04,
     SA_PLM_NTFID_UNMAPPED_HE_ALARM  = 0x05,
     SA_PLM_NTFID_STATE_CHANGE_ROOT  = 0x65,
     SA_PLM_NTFID_STATE_CHANGE_DEP   = 0x66,
     SA_PLM_NTFID_HPI_NORMAL_MSB     = 0x201,
     SA_PLM_NTFID_HPI_NORMAL_LSB     = 0x202,
     SA_PLM_NTFID_HPI_XDR            = 0x203
} SaPlmNotificationMinorIdT;
```

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

3.3.15.2 SaPlmAdditionalInfoIdT

```
typedef enum {
     SA_PLM_AI_ENTITY_PATH      = 1,
     SA_PLM_AI_ROOT_OBJECT      = 2,
     SA_PLM_AI_HPI_DOMAIN_ID    = 3,
     SA_PLM_AI_HPI_EVENT_DATA   = 4,
     SA_PLM_AI_HPI_RDR_DATA     = 5,
     SA_PLM_AI_HPI_RPT_DATA     = 6
} SaPlmAdditionalInfoIdT;
```

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

3.3.15.3 SaPlmStateT

```
typedef enum {
     SA_PLM_HE_ADMIN_STATE      = 1,
     SA_PLM_EE_ADMIN_STATE      = 2,
     SA_PLM_OPERATIONAL_STATE   = 3,
     SA_PLM_HE_PRESENCE_STATE   = 4,
     SA_PLM_EE_PRESENCE_STATE   = 5,
     SA_PLM_READINESS_STATE     = 6,
     SA_PLM_READINESS_FLAGS     = 7
} SaPlmStateT;
```

This type is used in Platform Management Service state change notifications to identify which state values (or the readiness flags) are changing for a PLM entity.

3.3.16 SaPlmCallbacksT

```
typedef struct {
     SaPlmReadinessTrackCallbackT saPlmReadinessTrackCallback;
} SaPlmCallbacksT;
```

The `SaPlmCallbacksT` callbacks structure is supplied to the PLM Service by a process and contains the callback functions that the PLM Service can invoke.

3.4 Library Life Cycle

3.4.1 saPlmInitialize()

Prototype

```
SaAisErrorT saPlmInitialize(
      SaPlmHandleT *plmHandle,
      const SaPlmCallbacksT *plmCallbacks,
      SaVersionT *version
);
```

Parameters

`plmHandle` - [`out`] A pointer to the handle which identifies this particular initialization of the PLM Service and which is to be returned by the PLM Service. The `SaPlmHandleT` type is defined in Section 3.3.1.1.

`plmCallbacks` - [`in`] If `plmCallbacks` is set to NULL, no callbacks are registered; if `plmCallbacks` is not set to NULL, it is a pointer to an `SaPlmCallbacksT` structure which contains the callback functions of the process that the PLM Service may invoke. Only non-NULL callback functions in this structure will be registered. The `SaPlmCallbacksT` is defined in Section 3.3.16.

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

Description

This function initializes the PLM Service for the invoking process and registers the various callback functions. This function must be invoked prior to the invocation of any other PLM Service API function. The handle pointed to by `plmHandle` is returned by the PLM Service as the reference to this association between the process and the PLM Service. The process uses this handle in subsequent communication with the PLM Service.
The `plmCallbacks` parameter points to a structure containing the callbacks that the PLM Service can invoke.

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

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

If the preceding condition cannot be met, `SA_AIS_ERR_VERSION` is returned, and the version to which the `version` parameter points is set to:

if (implementation supports the required `releaseCode`)

    `releaseCode` = required `releaseCode`

else {

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

        `releaseCode` = the lowest value of the supported release codes that is higher than the required `releaseCode`

    else

        `releaseCode` = the highest value of the supported release codes that is lower than the required `releaseCode`

}

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

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

Return Values

`SA_AIS_OK` - The function completed successfully.

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

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - Either the PLM Service library or a process that is providing the service is out of memory and cannot provide the service.

SA_AIS_ERR_NO_RESOURCES - The system is out of required resources (other than memory).

SA_AIS_ERR_VERSION - The version provided in the structure to which the `version` parameter points is not compatible with the version of the PLM Service implementation.

See Also

`saPlmSelectionObjectGet()`, `saPlmDispatch()`, `saPlmFinalize()`

3.4.2 saPlmSelectionObjectGet()

Prototype

```
SaAisErrorT saPlmSelectionObjectGet(
      SaPlmHandleT plmHandle,
      SaSelectionObjectT *selectionObject
);
```

Parameters

`plmHandle` - [in] The handle which was obtained by a previous invocation of the `saPlmInitialize()` function and which identifies this particular initialization of the PLM Service. The `SaPlmHandleT` type is defined in Section 3.3.1.1.

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

Description

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

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

The operating system handle returned by `saPlmSelectionObjectGet()` is valid until `saPlmFinalize()` is invoked on the same handle `plmHandle`.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

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

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

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

`SA_AIS_ERR_INVALID_PARAM` - A parameter is not set correctly.

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

`SA_AIS_ERR_NO_RESOURCES` - The system is out of required resources (other than memory).

See Also

`saPlmInitialize()`, `saPlmDispatch()`

3.4.3 saPlmDispatch()

Prototype

```
SaAisErrorT saPlmDispatch(
      SaPlmHandleT plmHandle,
      SaDispatchFlagsT dispatchFlags
);
```

Parameters

`plmHandle` - [in] The handle which was obtained by a previous invocation of the `saPlmInitialize()` function and which identifies this particular initialization of the PLM Service. The `SaPlmHandleT` type is defined in Section 3.3.1.1.

`dispatchFlags` - [in] Flags that specify the callback execution behavior of the `saPlmDispatch()` function, which have the values `SA_DISPATCH_ONE`, `SA_DISPATCH_ALL`, or `SA_DISPATCH_BLOCKING`. These flags are values of the `SaDispatchFlagsT` enumeration type, which is described in [2].

Description

In the context of the calling thread, this function invokes pending callbacks for the handle `plmHandle` in a way that is specified by the `dispatchFlags` parameter.

Return Values

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

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

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

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

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

`SA_AIS_ERR_INVALID_PARAM` - The `dispatchFlags` parameter is invalid.

See Also

`saPlmInitialize()`, `saPlmSelectionObjectGet()`

3.4.4 saPlmFinalize()

Prototype

```
SaAisErrorT saPlmFinalize(
     SaPlmHandleT plmHandle
);
```

Parameters

`plmHandle` - [in] The handle which was obtained by a previous invocation of the `saPlmInitialize()` function and which identifies this particular initialization of the PLM Service. The `SaPlmHandleT` type is defined in Section 3.3.1.1.

Description

The `saPlmFinalize()` function closes the association represented by the `plmHandle` parameter between the invoking process and the PLM Service. The process must have invoked `saPlmInitialize()` before it invokes this function. A process must call this function once for each handle it acquired by invoking `saPlmInitialize()`.

If the `saPlmFinalize()` function completes successfully, it cancels all pending callbacks related to the particular handle, stops tracking of entity groups associated with the handle, and deletes those entity groups. Moreover, it releases all resources acquired for that handle or for entity groups associated with the handle, including the memory allocated for the process in the `saPlmReadinessTrack()` function, if this memory has not yet been freed by a call to the `saPlmReadinessNotificationFree()` function.
Note that because the callback invocation is asynchronous, it is still possible that some callback calls are processed after this call returns successfully.

After `saPlmFinalize()` completes successfully, the handle `plmHandle` and the selection object associated with it are no longer valid.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

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

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

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

See Also

`saPlmInitialize()`

3.5 PLM Operations

3.5.1 Entity Group Management

For tracking purposes, PLM entities can be grouped together in entity groups. The scope of this grouping is local to the client process that created the entity group and not visible from other processes. This grouping is not visible in the IMM Service.

Tracking a group of PLM entities ensures that when a single cause affects the state of several entities of that group, a single invocation of the track callback will notify the client process of the state transitions.

3.5.1.1 saPlmEntityGroupCreate()

Prototype

```
SaAisErrorT saPlmEntityGroupCreate(
      SaPlmHandleT plmHandle,
      SaPlmEntityGroupHandleT *entityGroupHandle
);
```

Parameters

`plmHandle` - [`in`] The handle which was obtained by a previous invocation of the `saPlmInitialize()` function and which designates this particular initialization of the PLM Service. The `SaPlmHandleT` type is defined in Section 3.3.1.1.

`entityGroupHandle` - [`out`] A pointer to a memory area (provided by the invoking process in the address space of the process) to hold the entity group handle. If the entity group is created successfully, the PLM Service stores its handle in this memory area. This handle is used by the process to designate the entity group in subsequent invocations of the functions of the PLM Service API. The `SaPlmEntityGroupHandleT` type is defined in Section 3.3.1.2.

Description

The `saPlmEntityGroupCreate()` function creates an entity group that may be used later by the invoking process to track readiness status changes of entities in the group. The entity group is created as empty, and the `saPlmEntityGroupAdd()` function must be used to add entities into the group.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

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

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

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

See Also

`saPlmInitialize()`, `saPlmEntityGroupAdd()`, `saPlmEntityGroupDelete()`

3.5.1.2 saPlmEntityGroupAdd()

Prototype

```
SaAisErrorT saPlmEntityGroupAdd(
      SaPlmEntityGroupHandleT entityGroupHandle,
      const SaNameT *entityNames,
      SaUint32T entityNamesNumber,
      SaPlmGroupOptionsT options
);
```

Parameters

`entityGroupHandle` - [in] The handle for an entity group which was obtained by a previous invocation of the `saPlmEntityGroupCreate()` function. The `SaPlmEntityGroupHandleT` type is defined in Section 3.3.1.2.

`entityNames` - [`in`] Pointer to an array of entity names. The `SaNameT` type is defined in [2].

`entityNamesNumber` - [`in`] Number of names contained in the array referred to by `entityNames`. The `SaUint32T` type is defined in [2].

`options` – [`in`] Indicates how entity names provided in the array referred to by `entityNames` must be interpreted:

- If `options` is set to `SA_PLM_GROUP_SINGLE_ENTITY`, only the entities referred to by `entityNames` are added to the group.
- If `options` is set to `SA_PLM_GROUP_SUBTREE`, all subtrees that are rooted at the entities referred to by `entityNames` are added to the group.
- If `options` is set to `SA_PLM_GROUP_SUBTREE_HES_ONLY`, all hardware elements that are part of the subtrees rooted at the entities referred to by `entityNames` are added to the group.
- If `options` is set to `SA_PLM_GROUP_SUBTREE_EES_ONLY`, all execution environments that are part of the subtrees rooted at the entities referred to by `entityNames` are added to the group.

  The `SaPlmGroupOptionsT` type is defined in Section 3.3.12.

Description

The `saPlmEntityGroupAdd()` function adds a set of entities referred to by `entityNames` to the entity group designated by `entityGroupHandle`.

If new entities are configured later in these subtrees, the newly configured entities are automatically considered part of the group.

A given entity name can only be added once into an entity group, either as a single entity or as the root of a subtree. It is possible for the subtrees identified by entity names added into an entity group to overlap.

If this function fails and returns an error, the content of the entity group is not changed.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

SA_AIS_ERR_BAD_HANDLE - The handle `entityGroupHandle` is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained with the `saPlmEntityGroupCreate()` function, or the corresponding entity group associated with that handle has been deleted.
- The handle `plmHandle` that was passed in to the `saPlmEntityGroupCreate()` function has already been finalized.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

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

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

SA_AIS_ERR_EXIST - One of the names referred to by `entityNames` has already been added to the group, or it appears more than once in the array referred to by `entityNames`.

SA_AIS_ERR_NOT_EXIST - One of the names referred to by `entityNames` does not designate an entity configured for the PLM Service.

See Also

`saPlmEntityGroupCreate()`, `saPlmEntityGroupRemove()`

3.5.1.3 saPlmEntityGroupRemove()

Prototype

```
SaAisErrorT saPlmEntityGroupRemove(
      SaPlmEntityGroupHandleT entityGroupHandle,
      const SaNameT *entityNames,
      SaUint32T entityNamesNumber
);
```

Parameters

`entityGroupHandle` - [in] The handle for an entity group which was obtained by a previous invocation of the `saPlmEntityGroupCreate()` function. The `SaPlmEntityGroupHandleT` type is defined in Section 3.3.1.2.

`entityNames` - [in] Pointer to an array of entity names. The `SaNameT` type is defined in [2].

`entityNamesNumber` - [in] Number of names contained in the array `entityNames`. The `SaUint32T` type is defined in [2].

Description

The `saPlmEntityGroupRemove()` function removes entities from the entity group designated by `entityGroupHandle`. Entities may only be removed from the group in the same manner they have been added to the group: as a single entity or as a subtree. This means in particular that if a subtree has been added to the group using the `SA_PLM_GROUP_SUBTREE`, `SA_PLM_GROUP_SUBTREE_HES_ONLY`, or `SA_PLM_GROUP_SUBTREE_EES_ONLY` options, entities of that subtree cannot be removed from the group one by one. The single entities or subtrees of entities to be removed from the group are designated by the names referred to by `entityNames`.

If this function fails and returns an error, the content of the entity group is not changed.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

SA_AIS_ERR_BAD_HANDLE - The handle `entityGroupHandle` is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained with the `saPlmEntityGroupCreate()` function, or the corresponding entity group associated with that handle has been deleted.
- The handle `plmHandle` that was passed in to the `saPlmEntityGroupCreate()` function has already been finalized.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly.

SA_AIS_ERR_NOT_EXIST – One of the names referred to by `entityNames` does not match any name contained in the entity group referred to by `entityGroupHandle`.

See Also

`saPlmEntityGroupCreate()`, `saPlmEntityGroupAdd()`

3.5.1.4 saPlmEntityGroupDelete()

Prototype

```
SaAisErrorT saPlmEntityGroupDelete(
     SaPlmEntityGroupHandleT entityGroupHandle
);
```

Parameters

`entityGroupHandle` - [in] The handle for an entity group which was obtained by a previous invocation of the `saPlmEntityGroupCreate()` function. The `SaPlmEntityGroupHandleT` type is defined in Section 3.3.1.2.

Description

The `saPlmEntityGroupDelete()` function deletes the entity group designated by its handle `entityGroupHandle`. Moreover, it frees all resources allocated for it, including the memory allocated for the process in the `saPlmReadinessTrack()` function, if this memory has not yet been freed by a call to the `saPlmReadinessNotificationFree()` function.
If the `saPlmEntityGroupDelete()` function fails and returns an error, the entity group is not deleted, its content is not changed, and resources are not freed.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

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

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

`SA_AIS_ERR_BAD_HANDLE` - The handle `entityGroupHandle` is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained with the `saPlmEntityGroupCreate()` function, or the corresponding entity group associated with that handle has been deleted.
- The handle `plmHandle` that was passed in to the `saPlmEntityGroupCreate()` function has already been finalized.

`SA_AIS_ERR_BUSY` – The entity group designated by `entityGroupHandle` has been used to start a track request by invoking `saPlmReadinessTrack()`, and the track request is still in effect, as `saPlmReadinessTrackStop()` has not been called to terminate it.

See Also

`saPlmEntityGroupCreate()`, `saPlmReadinessTrack()`, `saPlmReadinessTrackStop()`

3.5.2 Readiness Status Tracking

The readiness status of a PLM entity is reflected by the value of its readiness state and readiness flags.

The readiness status of an entity can change as a consequence of:

- an administrative operation,
- a failure or a repair,
- an activation or a deactivation of a hardware element,
- the instantiation or termination of an execution environment,
- and of a problem or the clearance of a problem reported to the PLM Service with the `saPlmEntityReadinessImpact()` function targeted to the entity or another entity on which the entity depends.

The track API allows a process to track changes of the readiness status of a group of PLM entities.

In most cases, tracking processes are notified of a change after it already happened. This is done with a single invocation of their track callback, using the `SA_PLM_CHANGE_COMPLETED` step.

However, in some situations, the callback of tracking processes is also invoked before the change happens (during `SA_PLM_CHANGE_VALIDATE` or `SA_PLM_CHANGE_START` steps) allowing these tracking processes to validate the pending change and also to get ready before the change is effective.

The sequence of tracking steps is:

(1) `SA_PLM_CHANGE_VALIDATE`
The track callbacks are invoked requesting the tracking processes to validate the pending action and prepare themselves to perform the action. The invoked processes must provide a response (`SA_PLM_CALLBACK_RESPONSE_OK` or `SA_PLM_CALLBACK_RESPONSE_REJECTED`) to the PLM Service by invoking the `saPlmReadinessTrackResponse()` function.
Processes that respond with an error (`SA_PLM_CALLBACK_RESPONSE_ERROR`) or call `saPlmReadinessTrackStop()` with the entity group handle used to initiate the track operation are ignored by the PLM Service (that is, the PLM Service proceeds as if these processes had accepted the pending operation).

(2) `SA_PLM_CHANGE_START` or `SA_PLM_CHANGE_ABORTED`
If at least one process invoked during the `SA_PLM_CHANGE_VALIDATE` step rejects the operation, the PLM Service invokes the track callbacks indicating that the pending action has been aborted (`SA_PLM_CHANGE_ABORTED` step); otherwise, the PLM Service invokes the track callbacks again requesting the processes to now perform the action (`SA_PLM_CHANGE_START` step). Processes

must respond to PLM when the operation is completed (SA_PLM_CALLBACK_RESPONSE_OK), or if they fail to complete the operation (SA_PLM_CALLBACK_RESPONSE_ERROR).
When processes are not allowed to reject the pending change, they may be directly notified by an SA_PLM_CHANGE_START step without any prior tracking notification with an SA_PLM_CHANGE_VALIDATE step.

(3) SA_PLM_CHANGE_COMPLETED
When all clients involved in the SA_PLM_CHANGE_START step reported that they have completed the action, PLM performs actions required to complete the action and updates the readiness states of impacted entities. When this is done, PLM notifies tracking processes that the action has been completed (SA_PLM_CHANGE_COMPLETED step).

Responses to the SA_PLM_CHANGE_VALIDATE and SA_PLM_CHANGE_START steps are not time-bounded. The operator should initiate and monitor all operations leading to the SA_PLM_CHANGE_VALIDATE or SA_PLM_CHANGE_START steps. If a graceful LOCK administrative operation does not complete in a timely manner, the operator can issue a forced LOCK; if hot swap indicators do not indicate deactivation in time, the operator can abruptly extract hardware or also issue a forced LOCK. The PLM Service must wait for responses to the SA_PLM_CHANGE_VALIDATE and SA_PLM_CHANGE_START steps until:

- all invoked processes have responded, or until
- tracking for the entity group has stopped, or until
- the current tracking notification has been superseded by a new tracking notification, which is triggered by the same entity having a different expected or new readiness status as a consequence of another operation on the same entity.

When a process initiates the track operation, the PLM Service always notifies the process in the SA_PLM_CHANGE_COMPLETED step. In addition, the process can request to receive additional callback notifications, as described next:

- A process that wants to be able to reject a change request in certain circumstances may use the SA_TRACK_VALIDATE_STEP flag in the call to saPlmReadinessTrack(). The track callback function of the process will then be invoked in the SA_PLM_CHANGE_VALIDATE step. If a change is rejected by a client of the track API, the process will also receive callbacks in the SA_PLM_CHANGE_ABORTED step.
- A process that needs to get ready before a change is effective may use the SA_TRACK_START_STEP flag in the call to saPlmReadinessTrack(). The track callback function of the process will then be invoked in the SA_PLM_CHANGE_START step.

Only changes in the readiness status of a tracked entity are reported, so if a particular cause does not change the readiness status of any entity in a tracked group, the track callback of the tracking process is not invoked.

For example, if a board is administratively locked, extracting the board or reporting a failure of the board triggers no callback. Depending on the order of these actions, a process may be notified of a board going out-of-service with an SA_PLM_CAUSE_LOCK cause and then be notified that the board is back in service with an SA_PLM_CAUSE_FAILURE_CLEARED cause without being notified of the intermediate failure and unlock events.

3.5.2.1 saPlmReadinessTrack()

Prototype

```
SaAisErrorT saPlmReadinessTrack(
      SaPlmEntityGroupHandleT entityGroupHandle,
      SaUint8T trackFlags,
      SaUint64T trackCookie,
      SaPlmReadinessTrackedEntitiesT *trackedEntities
);
```

Parameters

`entityGroupHandle` - [in] The handle for an entity group which was obtained by a previous invocation of the `saPlmEntityGroupCreate()` function. The `SaPlmEntityGroupHandleT` type is defined in Section 3.3.1.2.

`trackFlags` – [in] The kind of tracking that is requested, which is the bitwise OR of one or more of the following flags (as defined in [2]), which have the following interpretation here:

- SA_TRACK_CURRENT - Request the current readiness information of the entity group designated by the handle `entityGroupHandle`.
  If `trackedEntities` is NULL, information about all entities that are currently in this entity group is returned by a single subsequent invocation of the `SaPlmReadinessTrackCallbackT` tracking callback; otherwise, this information is returned in the structure to which `trackedEntities` points when the `saPlmReadinessTrack()` call completes successfully. The `change` field in each entry of the array pointed to by the `entities` pointer in the structure referred to by `trackedEntities` is set to SA_PLM_GROUP_NO_CHANGE. The

fields `currentReadinessStatus` and `expectedReadinessStatus` both contain the current readiness status of this entity.

- `SA_TRACK_CHANGES` - Start readiness tracking, requesting a complete picture of all entities in the entity group.
  The `SaPlmReadinessTrackCallbackT` callback function is invoked each time the readiness status of one of the entities in this entity group changes (or is about to change) or when entities are added to or removed from the entity group. The structure to which `trackedEntities` points in an invocation of the `SaPlmReadinessTrackCallbackT` function contains information about all entities that are currently in the entity group and also about entities that have been removed from the entity group since the last invocation of `SaPlmReadinessTrackCallbackT`.
- `SA_TRACK_CHANGES_ONLY` - Start readiness tracking, requesting a list of only the changed entities with each callback.
  The `SaPlmReadinessTrackCallbackT` callback function is invoked each time the readiness status of one of the entities in this group changes (or is about to change) or when entities are added to or removed from the entity group. The structure to which `trackedEntities` points in an invocation of the `SaPlmReadinessTrackCallbackT` function contains only information about entities whose readiness status have changed or have been added or removed from the entity group since the last invocation of `SaPlmReadinessTrackCallbackT`.
- `SA_TRACK_START_STEP` - Request additionally the `SA_PLM_CHANGE_START` step. This flag is ignored, if neither `SA_TRACK_CHANGES` nor `SA_TRACK_CHANGES_ONLY` is set.
- `SA_TRACK_VALIDATE_STEP` - Request additionally the `SA_PLM_CHANGE_VALIDATE` step. This flag is ignored, if neither `SA_TRACK_CHANGES` nor `SA_TRACK_CHANGES_ONLY` is set.

If both `SA_TRACK_CHANGES` and `SA_TRACK_CHANGES_ONLY` are set in an invocation of this function, the function returns `SA_AIS_ERR_BAD_FLAGS`, and tracking is not started. An invocation of this function is also invalid and returns `SA_AIS_ERR_BAD_FLAGS` if none of the flags `SA_TRACK_CHANGES`, `SA_TRACK_CHANGES_ONLY`, or `SA_TRACK_CURRENT` are set.
The PLM Service does not support the `SA_TRACK_LOCAL` flag, and it will be ignored.
The `SaUint8T` type is defined in [2].

`trackCookie` - [in] Value provided by the invoking process, and which will be passed as a parameter to all invocations of the `SaPlmReadinessTrackCallbackT` function triggered by this invocation of the `saPlmReadinessTrack()` function. This parameter can be used to pass process specific information related to the group of entities being tracked. The `SaUint64T` type is defined in [2].

`trackedEntities` – [in/out] A pointer to a structure of type `SaPlmReadinessTrackedEntitiesT` (defined in Section 3.3.13.5). This parameter is ignored if SA_TRACK_CURRENT is not set in `trackFlags`; otherwise, if `trackedEntities` is not NULL, the structure will contain information about all entities in the entity group when `saPlmReadinessTrack()` returns. The meaning of the fields of the `SaPlmReadinessTrackedEntitiesT` structure is:

- `numberOfEntities` – [in/out] If `entities` is NULL, `numberOfEntities` is ignored as input parameter; otherwise, it specifies that the array to which `entities` points provides memory for information about the readiness status of `numberOfEntities` entities. When `saPlmReadinessTrack()` returns with SA_AIS_OK or with SA_AIS_ERR_NO_SPACE, `numberOfEntities` contains the number of entities in the entity group.
- `entities` - [in/out] If `entities` is NULL, memory for the readiness status information of the tracked entities is allocated by the PLM Service. The caller is responsible for freeing the allocated memory by calling the `saPlmReadinessNotificationFree()` function.

Description

The `saPlmReadinessTrack()` function can be used to retrieve the current readiness status of all PLM entities that are contained in the entity group referred to by `entityGroupHandle`, to start tracking changes of the readiness status of these entities, or to perform both actions.

PLM provides this information to the process by invoking its `saPlmReadinessTrackCallback()` callback, which must have been supplied when the process invoked the `saPlmInitialize()` call.

A process may call `saPlmReadinessTrack()` repeatedly for the same value of `entityGroupHandle`, regardless of whether the call initiates a one-time status request or a series of callback invocations.
If a process has enabled tracking by calling `saPlmReadinessTrack()` with either SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY set and then calls `saPlmReadinessTrack()` again with the same value of `entityGroupHandle`, the following applies, depending on the flags in the second call:

- If the second call has either SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY set, the new combination of flags is used to change the settings for the tracking. For example, if the first call had SA_TRACK_START_STEP set, and the second call does not have this flag set, the process will not receive further callbacks for the SA_TRACK_START_STEP. The `trackCookie` of the second call will be used from now on in invocations of the `saPlmReadinessTrackCallback()` callback function of the process.
- If the second call has neither SA_TRACK_CHANGES nor SA_TRACK_CHANGES_ONLY set, but rather only SA_TRACK_CURRENT, the tracking started by the first call will proceed unchanged, and the process will additionally receive the current information about all entities that are currently in the entity group designated by its `entityGroupHandle`.

Note that it is possible for entities to be added or removed from the entity group while a track operation is in progress on the group.

Return Values

SA_AIS_OK - The function completed successfully.

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

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

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

SA_AIS_ERR_BAD_HANDLE - The handle `entityGroupHandle` is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained with the `saPlmEntityGroupCreate()` function, or the corresponding entity group associated with that handle has been deleted.
- The handle `plmHandle` that was passed in to the `saPlmEntityGroupCreate()` function has already been finalized.

SA_AIS_ERR_INIT – The initialization of the PLM Service library with `saPlmInitialize()` used later on to create the entity group designated by `entityGroupHandle` was incomplete, since the `saPlmReadinessTrackCallback()` callback function of the process is missing. This value is returned only if `saPlmReadinessTrack()` is called in a way that requires the callback; that is, with SA_TRACK_CHANGES or SA_TRACK_CHANGES_ONLY set, or with SA_TRACK_CURRENT set and `trackedEntities` set to NULL.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this applies if in the structure to which `trackedEntities` points the entities pointer is not NULL and `numberOfEntities` is 0.

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

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

SA_AIS_ERR_NO_SPACE - The SA_TRACK_CURRENT flag is set, and the `entities` pointer in the structure referred to by `trackedEntities` is not NULL, but the value of `numberOfEntities` in this structure is smaller than the number of entries to be provided in the array referred to by the `entities` pointer.

SA_AIS_ERR_BAD_FLAGS – The `trackFlags` parameter is invalid. In particular, this applies if

- the SA_TRACK_CHANGES and SA_TRACK_CHANGES_ONLY flags are both specified or if
- none of the flags SA_TRACK_CHANGES, SA_TRACK_CHANGES_ONLY, or SA_TRACK_CURRENT are set.

See Also

`saPlmEntityGroupCreate()`, `SaPlmReadinessTrackCallbackT`, `saPlmReadinessTrackStop()`, `saPlmReadinessNotificationFree()`

3.5.2.2 SaPlmReadinessTrackCallbackT

Prototype

```
typedef void (*SaPlmReadinessTrackCallbackT)(
      SaPlmEntityGroupHandleT entityGroupHandle,
      SaUint64T trackCookie,
      SaInvocationT invocation,
      SaPlmTrackCauseT cause,
      const SaNameT *rootCauseEntity,
      SaNtfIdentifierT rootCorrelationId,
      const SaPlmReadinessTrackedEntitiesT *trackedEntities,
      SaPlmChangeStepT step,
      SaAisErrorT error
);
```

Parameters

`entityGroupHandle` - [in] The handle for an entity group which was obtained by a previous invocation of the `saPlmEntityGroupCreate()` function. The `SaPlmEntityGroupHandleT` type is defined in Section 3.3.1.2.

`trackCookie` - [in] Value that was provided to the `saPlmReadinessTrack()` function when the tracking was initialized. This value complements the handle `entityGroupHandle` and holds process specific information related to the group of entities being tracked. The `SaUint64T` type is defined in [2].

`invocation` – [in] This parameter is used by the invoked process to provide a response to the PLM Service with the `saPlmReadinessTrackResponse()` function. This parameter enables the PLM Service to associate the `saPlmReadinessTrackResponse()` invocation with this particular callback invocation. The `SaInvocationT` type is defined in [2].

`cause` – [in] Indicates the action or event that caused the invocation of the callback. The `SaPlmTrackCauseT` type is defined in Section 3.3.13.3.

`rootCauseEntity` – [in] Pointer to the name of the entity directly targeted by the action or the event identified by the `cause` parameter. It is set to NULL if `cause` is set to SA_PLM_CAUSE_GROUP_CHANGE, SA_PLM_CAUSE_MANAGEMENT_LOST, or SA_PLM_CAUSE_MANAGEMENT_REGAINED. The `SaNameT` type is defined in [2].

`rootCorrelationId` – [in] Correlation identifier associated with the root cause. It is set to `SA_NTF_IDENTIFIER_UNUSED` if `cause` is set to `SA_PLM_CAUSE_GROUP_CHANGE`, `SA_PLM_CAUSE_MANAGEMENT_LOST`, `SA_PLM_CAUSE_MANAGEMENT_REGAINED`, or `SA_PLM_CAUSE_STATUS_INFO`. The `SaNtfIdentifierT` type is defined in [4].

`trackedEntities` – [in] Pointer to a structure that contains information about the changes in the entity group designated by `entityGroupHandle`. The `SaPlmReadinessTrackedEntitiesT` type is defined in Section 3.3.13.5.

`step` – [in] Indicates the tracking step in which the callback is invoked. The `SaPlmChangeStepT` type is defined in Section 3.3.13.2.

`error` – [in] This parameter indicates whether the PLM Service was able to perform the operation. The `SaAisErrorT` type is defined in [2].

The parameter `error` has one of the values:

- `SA_AIS_OK` – No error has been encountered by the PLM Service during the tracking process.
- `SA_AIS_ERR_LIBRARY` - An unexpected problem occurred in the library (such as corruption). The library cannot be used anymore.
- `SA_AIS_ERR_TRY_AGAIN` - The service cannot be provided at this time. The process may retry the `saPlmReadinessTrack()` call later.
- `SA_AIS_ERR_BAD_HANDLE` - The handle `entityGroupHandle` is invalid, due to one or both of the reasons below:
  - It is corrupted, or the corresponding entity group associated with that handle has been deleted.
  - The handle `plmHandle` that was passed in to the `saPlmEntityGroupCreate()` function has already been finalized.
- `SA_AIS_ERR_NO_MEMORY` - Either the PLM Service library or the provider of the service is out of memory and cannot provide the service. The process that invoked `saPlmReadinessTrack()` might have missed one or more tracking notifications.
- `SA_AIS_ERR_NO_RESOURCES` - There are insufficient resources (other than memory). The process that invoked `saPlmReadinessTrack()` might have missed one or more tracking notifications.

Description

This callback is invoked in the context of a thread issuing an `saPlmDispatch()` call with the PLM library handle that was specified when the entity group designated by the handle `entityGroupHandle` was created. If successful, the `saPlmReadinessTrackCallback()` function passes information about tracked entities in the structure pointed to by the `trackedEntities` parameter. The kind of information passed depends on the setting of the `trackFlags` parameter of the `saPlmReadinessTrack()` function.

This callback is invoked when:

- entities are added to or removed from the entity group, either as a result of calls to `saPlmEntityGroupAdd()` or `saPlmEntityGroupRemove()`, or because PLM entities are created or deleted on subtrees that are selected for tracking in the entity group, or
- the readiness status of at least one of the entities of the group is about to change (`SA_PLM_CHANGE_VALIDATE` or `SA_PLM_CHANGE_START` steps), or has changed (`SA_PLM_CHANGE_COMPLETED` step), or
- a pending change has been rejected (`SA_PLM_CHANGE_ABORTED` step).

A single invocation of this callback contains information about all entities of the tracked group whose readiness status is or will be affected by the same cause identified by the `cause` and `rootCauseEntity` parameters.

When the `step` parameter is set to `SA_PLM_CHANGE_VALIDATE` or `SA_PLM_CHANGE_START`, the invoked process must provide a response to the PLM Service with the `saPlmReadinessTrackResponse()` function. When the `step` parameter is set to `SA_PLM_CHANGE_ABORTED` or `SA_PLM_CHANGE_COMPLETED`, the invoked process must not provide a response to the PLM Service.

If an error that prevents the PLM Service from satisfying the tracking request initiated by a previous call to `saPlmReadinessTrack()` occurs, the error is returned in the `error` parameter.

A process may concurrently track changes of the readiness status of several entity groups. If an entity is part of more than one of these groups, a change of the entity's readiness status will be notified several times to the process, that is, its `saPlmReadinessTrackCallback()` function will be invoked once for each affected group.

Return Values

None

See Also

`saPlmReadinessTrack()`, `saPlmInitialize()`, `saPlmDispatch()`, `saPlmReadinessTrackResponse()`

3.5.2.3 saPlmReadinessTrackResponse()

Prototype

```
SaAisErrorT saPlmReadinessTrackResponse(
      SaPlmEntityGroupHandleT entityGroupHandle,
      SaInvocationT invocation,
      SaPlmReadinessTrackResponseT response
);
```

Parameters

`entityGroupHandle` - [in] The handle for an entity group which was obtained by a previous invocation of the `saPlmEntityGroupCreate()` function. The `SaPlmEntityGroupHandleT` type is defined in Section 3.3.1.2.

`invocation` – [in] This parameter was provided to the process by the PLM Service in the `SaPlmReadinessTrackCallbackT` callback. It enables the PLM Service to associate this `saPlmReadinessTrackResponse()` invocation with the previous callback invocation. The `SaInvocationT` type is defined in [2].

`response` – [in] This parameter provides the response expected by the PLM Service to a previous invocation of the `SaPlmReadinessTrackCallbackT` track callback. The `SaPlmReadinessTrackResponseT` type is defined in Section 3.3.14.

Description

This function is used by a process to provide a response to an `SaPlmReadinessTrackCallbackT` callback previously invoked with a `step` parameter equal to either `SA_PLM_CHANGE_VALIDATE` or `SA_PLM_CHANGE_START`. The `invocation` parameter must be set to the value passed in the `invocation` parameter of the track callback. The `response` parameter holds the response of the process.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

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

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

SA_AIS_ERR_BAD_HANDLE - The handle `entityGroupHandle` is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained with the `saPlmEntityGroupCreate()` function, or the corresponding entity group associated with that handle has been deleted.
- The handle `plmHandle` that was passed in to the `saPlmEntityGroupCreate()` function has already been finalized.

SA_AIS_ERR_INVALID_PARAM - A parameter is not set correctly. In particular, this applies if `invocation` is invalid, or there is no outstanding response for a track callback with the same values of `invocation` and `entityGroupHandle`.

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

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

See Also

`SaPlmReadinessTrackCallbackT`

3.5.2.4 saPlmReadinessTrackStop()

Prototype

```
SaAisErrorT saPlmReadinessTrackStop(
      SaPlmEntityGroupHandleT entityGroupHandle
);
```

Parameters

`entityGroupHandle` - [in] The handle for an entity group which was obtained by a previous invocation of the `saPlmEntityGroupCreate()` function. The `SaPlmEntityGroupHandleT` type is defined in Section 3.3.1.2.

Description

The `saPlmReadinessTrackStop()` function stops any further tracking notifications of readiness status changes of the group of entities designated by `entityGroupHandle` and which were requested by specifying the handle `entityGroupHandle` when invoking the `saPlmReadinessTrack()` function, and which are still in effect. Pending callbacks are removed.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

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

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

`SA_AIS_ERR_BAD_HANDLE` - The handle `entityGroupHandle` is invalid, due to one or both of the reasons below:

- It is corrupted, was not obtained with the `saPlmEntityGroupCreate()` function, or the corresponding entity group associated with that handle has been deleted.
- The handle `plmHandle` that was passed in to the `saPlmEntityGroupCreate()` function has already been finalized.

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

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

`SA_AIS_ERR_NOT_EXIST` – No track of readiness status changes in the entity group designated by `entityGroupHandle` was started by invoking the `saPlmReadinessTrack()` function with track flags `SA_TRACK_CHANGES` or `SA_TRACK_CHANGES_ONLY` and is still in effect.

See Also

`saPlmReadinessTrack()`

3.5.2.5 saPlmReadinessNotificationFree()

Prototype

```
SaAisErrorT saPlmReadinessNotificationFree(
      SaPlmEntityGroupHandleT entityGroupHandle,
      SaPlmReadinessTrackedEntityT *entities
);
```

Parameters

`entityGroupHandle` - [in] The handle for an entity group which was obtained by a previous invocation of the `saPlmEntityGroupCreate()` function. The `SaPlmEntityGroupHandleT` type is defined in Section 3.3.1.2.

`entities` – [in] A pointer to the memory array that was allocated by the PLM Service library in the `saPlmReadinessTrack()` function and is to be deallocated. The `SaPlmReadinessTrackedEntityT` type is defined in Section 3.3.13.4.

Description

This function frees the memory to which `entities` points and which was allocated by the PLM Service library in a previous call to the `saPlmReadinessTrack()` function. For details, refer to the description of the `entities` pointer in the structure referred to by the `trackedEntities` parameter in the corresponding invocation of the `saPlmReadinessTrack()` function.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

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

`SA_AIS_ERR_INVALID_PARAM` – A parameter is not set correctly. In particular, this applies if the `entities` parameter does not point to memory allocated by PLM in a previous call to the `saPlmReadinessTrack()` function using the same value of the `entityGroupHandle` parameter.

See Also

`saPlmReadinessTrack()`

3.5.3 Entity Readiness Impact

3.5.3.1 saPlmEntityReadinessImpact()

Prototype

```
SaAisErrorT saPlmEntityReadinessImpact(
      SaPlmHandleT plmHandle,
      const SaNameT *impactedEntity,
      SaPlmReadinessImpactT impact,
      SaNtfCorrelationIdsT *correlationIds
);
```

Parameters

`plmHandle` - [`in`] The handle which was obtained by a previous invocation of the `saPlmInitialize()` function and which designates this particular initialization of the PLM Service. The `SaPlmHandleT` type is defined in Section 3.3.1.1.

`impactedEntity` – [`in`] Pointer to the name of the entity whose readiness status should be updated. The `SaNameT` type is defined in [2].

`impact` – [`in`] Impact being reported. The `SaPlmReadinessImpactT` type is defined in Section 3.3.10.

`correlationIds` – [`in`/`out`] Pointer to a structure that contains correlation identifiers. The `rootCorrelationId` and `parentCorrelationId` fields are `in` parameters and hold respectively the root and parent correlation identifiers to be included by the PLM Service when it generates NTF notifications directly related to this invocation. The `notificationId` field is an `out` parameter. If the invocation of the `saPlmEntityReadinessImpact()` function triggers a state change of the target entity, the PLM Service returns in `notificationId` the identifier of the corresponding state change NTF notification; otherwise PLM sets `notificationId` to `SA_NTF_IDENTIFIER_UNUSED`. The `SaNtfCorrelationIdsT` type is defined in [4].

Description

The `saPlmEntityReadinessImpact()` function is used by processes to report that the state of health of an entity has changed. The change may result in changes to the operational state and readiness status of the entity being reported as well as in changes to the readiness status of other entities that are children of or dependent on the entity being reported.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

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

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

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

`SA_AIS_ERR_INVALID_PARAM` - A parameter is not set correctly.

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

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

`SA_AIS_ERR_NOT_EXIST` – The name referred to by the `impactedEntity` parameter does not designate an entity configured for the PLM Service.

See Also

`saPlmInitialize()`

4 PLM Service UML Information Model

The PLM Service Information Model is described in UML and has been organized in UML class diagrams.

The PLM Service Information Model is implemented by the SA Forum IMM Service ([5]). For further details on this implementation, refer to the SA Forum Overview document ([1]).

The classes in the PLM Service UML class diagrams show the contained attributes and their type, multiplicity, default values, and constraints. The description of each attribute is provided in the SA Forum XMI document (see [8]). The class diagrams additionally show the administrative operations (if any) applicable on these classes.

To simplify references, this description uses for the UML diagrams the same names used in [8].

The UML diagrams defined for the PLM Service are:

- “Cluster View”
- “PLM Instances and Types View”
- “PLM HE Classes”
- “PLM EE Classes”
- “PLM Other Classes”

These diagrams will be described starting with Section 4.2.

4.1 Notes on the Conventions Used in UML Diagrams

A general explanation of the conventions used in the UML diagrams, such as the use of constraints, default values, and the like is presented in [1].

4.2 DN Formats for PLM Service UML Classes

Table 1 provides the format of the various DNs used to name PLM objects of the SA Forum Information Model. One format is defined for each object class.
The '`[safXX=...,]*`' notation indicates that an RDN in the form '`safXX=...`' may occur zero or more times in the DN at a particular position. The '`[safXX=...,]+`' notation indicates that an RDN in the form '`safXX=...`' may occur 1 or more times in the DN at a particular position.

Table 1 DN Formats

Object Class	DN Format for Objects of that Class
`SaPlmDependency`	`"safDependency=...,[safEE=...,]+[safHE=...,]*safDomain=..."` or `"safDependency=...,[safHE=...,]+safDomain=..."`
`SaPlmDomain`	`"safDomain=..."`
`SaPlmEE`	`"[safEE=...,]+[safHE=...,]*safDomain=..."`
`SaPlmEEBaseType`	`"safEEType=..., safApp=safPlmService"`
`SaPlmEEType`	`"safVersion=...,safEEType=..., safApp=safPlmService"`
`SaPlmHE`	`"[safHE=...,]+safDomain=..."`
`SaPlmHEBaseType`	`"safHEType=..., safApp=safPlmService"`
`SaPlmHEType`	`"safVersion=...,safHEType=..., safApp=safPlmService"`

4.3 PLM Classes and Other Services' Classes

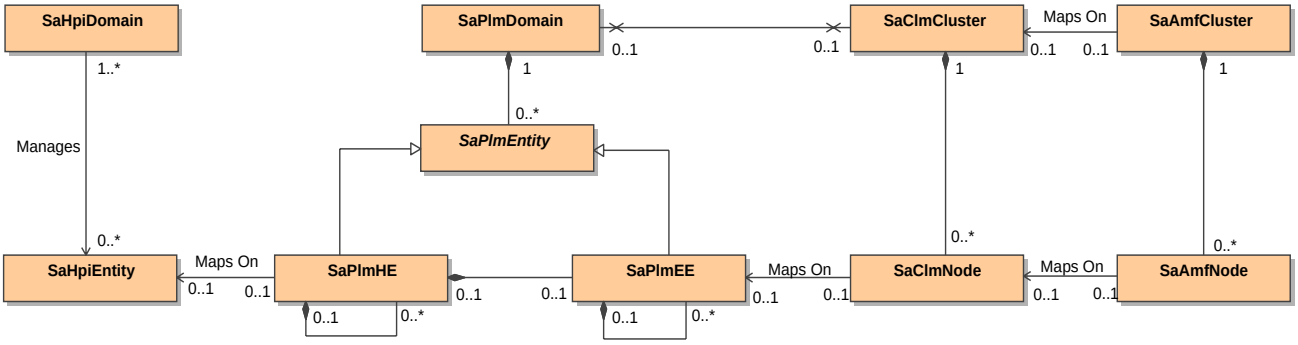
FIGURE 4 shows the relationships among the main classes of the PLM Service and the main classes of HPI, CLM, and AMF.

Attributes and operations of the PLM classes `SaPlmHE`, `SaPlmEE`, and `SaPlmDomain` are shown in Section 4.5, Section 4.6, and Section 4.7 respectively.

The `SaPlmEntity` class is only used to facilitate the UML description and is not implemented in the IMM Service.

HPI, CLM, and AMF object classes are not shown in this document.

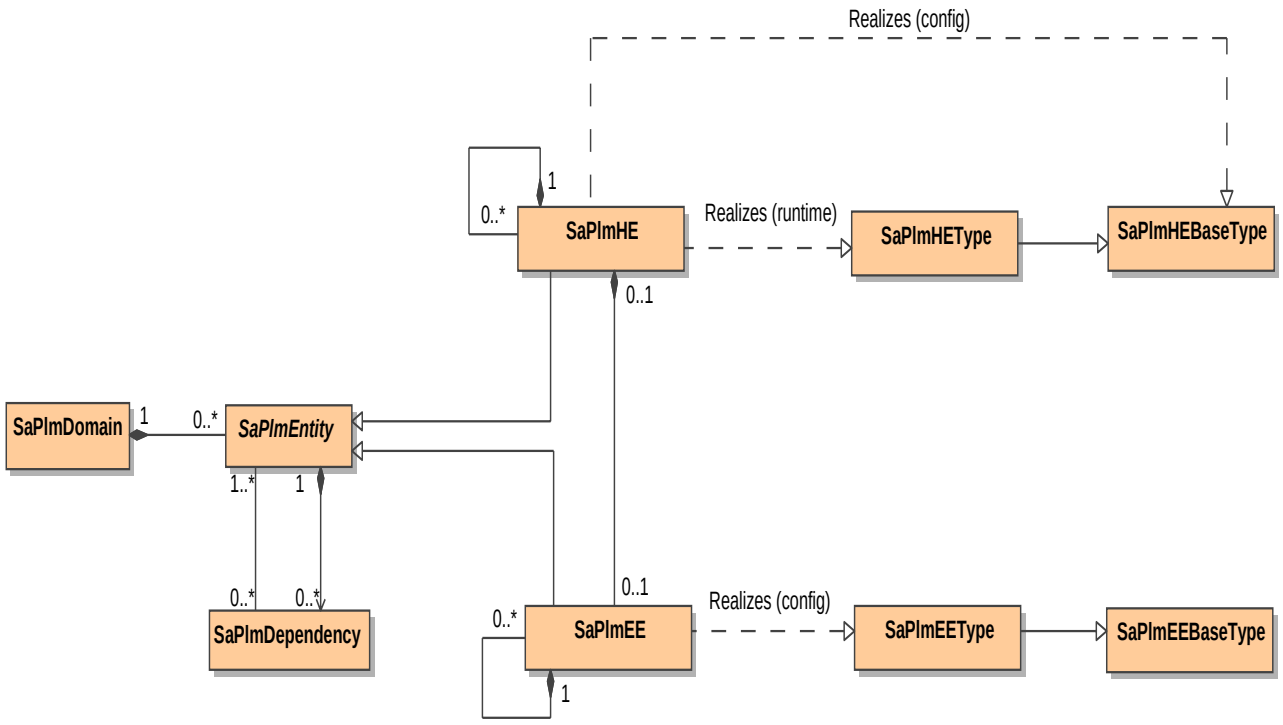
FIGURE 4 Cluster View



4.4 PLM Instances and Types View

FIGURE 5 shows all classes of the PLM Service. The `SaPlmEntity` class is only used to facilitate the UML description and is not implemented in the IMM Service.

FIGURE 5 PLM Instances and Types View



Note that HE and EE classes relate in a different way to their respective type and base type classes:

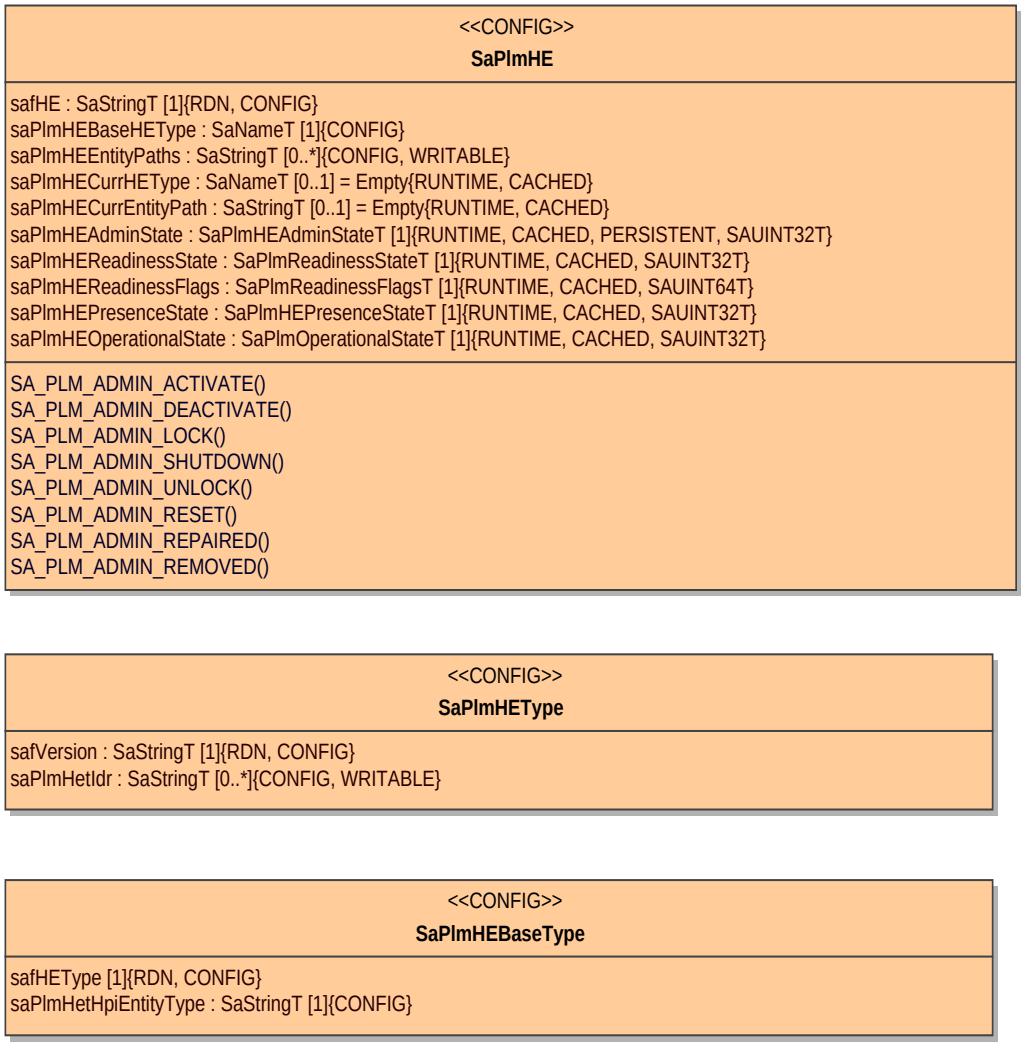
- An HE instance is configured to relate to a particular base type. The relationship between an HE instance and its type is established dynamically at run time by the PLM Service after the matching process described in Section 4.5.1 completes successfully.
- An EE instance is configured to relate to a particular type, providing no dynamic mapping capability at runtime.

4.5 PLM HE Classes Diagram

The diagram shown in FIGURE 6 contains the following classes:

- `SaPlmHE` — This configuration object class defines configuration and runtime attributes of a hardware element and the operations that can be applied on the hardware element. For each hardware element, an object of this class must be configured, and its `saPlmHEBaseHEType` attribute must contain the DN of a valid object of the `SaPlmHEBaseType` object class. When hardware matching is completed, as described in Section 4.5.1, its `saPlmCurrHEType` will contain the DN of an object of the `SaPlmHEType` object class.
- `SaPlmHEType` — This configuration object class defines configuration attributes of a hardware element type. All hardware elements of the same type share the attribute values defined in the hardware element type configuration.
- `SaPlmHEBaseType` — This configuration object class defines the configuration attributes common to different hardware element types. In particular, a base hardware element type defines the common name of versioned hardware element types. A hardware element type x belongs to a base hardware element type y based on the DN of x, which is the concatenation of the RDN of x (representing its version) with the DN of y.

FIGURE 6 PLM HE Classes



4.5.1 Matching Configured HEs to Hardware Entities

At system startup and at each time new hardware is added, the PLM Service attempts to match the newly discovered hardware entities to the configuration described in the information model. The PLM Service must not remap any hardware element, while it is in-service. When attempting to match a hardware element with a hardware entity, the PLM Service performs the following checks:

- It checks that the hardware entity is located in one of the potential locations that have been configured for the hardware element. The entity path of the hardware entity is used by the PLM Service to perform this check.

- It checks that the characteristics of the hardware entity match the configured characteristics of the hardware element. The contents of its Inventory Data Repositories are used by the PLM Service to perform this check.

The following sections describe in more details how these checks are performed.

4.5.1.1 Hardware Entity Location Check

Each `saPlmHE` object class contains two attributes that are used by the PLM Service to check if the entity path of a hardware entity matches the configuration of a particular hardware element:

- The `saPlmHECurrEntityPath` runtime attribute holds the entity path (using the standard text string representation for entity paths defined in [3]) of the hardware entity that has been matched with the hardware element. This attribute is set by the PLM Service when the matching is completed.
- The `saPlmHEEntityPaths` configuration attribute holds one or several entity path fragments that are relative to the location of the parent of the PLM hardware element entity in the PLM Service containment tree.

When checking that an HPI entity is properly located for a particular hardware element, the PLM Service combines the entity path of the parent of the hardware element (`saPlmHECurrEntityPath` attribute of the parent) with each of the entity path fragments contained in the `saPlmHEEntityPaths` attribute and checks that they match the entity path of the hardware entity.

So, for example, if the parent's `saPlmHECurrEntityPath` attribute is "`SUBRACK.1,RACK.2`", and the `saPlmHEEntityPaths` attribute of a hardware element includes: "`SYSTEM_BLADE.2`" and "`SYSTEM_BLADE.3`", then that hardware element could match a hardware entity with an entity path "`SYSTEM_BLADE.2,SUBRACK.1,RACK.2`" or "`SYSTEM_BLADE.3,SUBRACK.1,RACK.2`".

As another example, the parent's `saPlmHECurrEntityPath` attribute might be the same as above, but the hardware element's `saPlmHEEntityPaths` attribute could contain: "`PICMG_FRONT_BLADE.0,PHYSICAL_SLOT.3`" and "`PICMG_FRONT_BLADE.0,PHYSICAL_SLOT.4`". Now, it could match: "`PICMG_FRONT_BLADE.0,PHYSICAL_SLOT.3,SUBRACK.1,RACK.2`" or "`PICMG_FRONT_BLADE.0,PHYSICAL_SLOT.4,SUBRACK.1,RACK.2`".

The entity path fragments contained in the `saPlmHEEntityPaths` attribute can be specified exactly or with a syntax that allows multiple locations to match. The numbers that indicate the entity location in the entity path fragments can have:

- a number, for an exact match,
- an asterisk (*), which matches any value, or
- a list of numbers and ranges of numbers separated by colons (ranges specified with hyphens).

Examples

"`PROCESSOR.3`" matches the `PROCESSOR` entity type at location `3` prepended to the parent's entity path.
"`PROCESSOR.*`" matches the `PROCESSOR` entity type, any location, prepended to the parent's entity path.
"`PROCESSOR.1:3:5-8`" matches the `PROCESSOR` entity type, entity locations `1, 3, 5, 6, 7, 8` prepended to the parent's entity path.
"`PROCESSOR.1:3-`" matches the `PROCESSOR` entity type, entity locations `1`, or entity locations greater than or equal to `3` prepended to the parent's entity path.
"`AMC.*,PHYSICAL_SLOT.1-4`" match an entity path with two additional elements prepended to the parent's entity path. The first element in the resulting entity path is of `AMC` entity type at any location, the second is of `PHYSICAL_SLOT` entity type, entity location `1,2,3,` or `4`.

4.5.1.2 HPI Entity Characteristics Check

After checking that the location of a hardware entity matches the hardware element's configuration, the PLM Service checks that the other characteristics of the hardware entity are also matching the configuration.

Each `saPlmHE` object class contains an `saPlmHEBaseHEType` configuration attribute that associates a hardware element with a particular base type (the attribute contains the name of an object of class `SaPlmHEBaseType`).

All hardware element types (represented by objects of class `SaPlmHEType`) that correspond to the different versions of the hardware element's base type are checked for a potential match with the hardware entity. According to the naming conventions listed in Section 4.2, the name of these hardware element types is built from the name of their base type.

The SaPlmHEType object class contains an saPlmHetIdr configuration attribute that is used to specify the characteristics of the hardware entities that can be represented by a particular hardware element type. The saPlmHetIdr configuration attribute holds values that are matched to fields of an Inventory Data Repositories (IDR) associated with the hardware entity.

One IDR field setting is represented using the following format:

"IDR_Area_Name/IDR_Field_Name=IDR_Field_Value"

where IDR_Area_Name and IDR_Field_Name may take one of the values defined in Table 2.

Table 2 IDR Names and Values

IDR Names	IDR Values
IDR_Area_Name	"BOARD","CHASSIS","INTERNAL","OEM", "PRODUCT"
IDR_Field_Name	"ASSET_TAG","CHASSIS_TYPE","CUSTOM","FILE_ID", "MANUFACTURER","MFG_DATETIME","PART_NUMBER", "PRODUCT_NAME","PRODUCT_VERSION","SERIAL_NUMBER"

If the setting of several IDR fields must be specified to identify a particular hardware entity, they must all be concatenated to form a single string value using "," as a a separator. Below is an example of a value held by the saPlmHetIdr configuration attribute:

"PRODUCT/MANUFACTURER=*name_of_the_manufacturer*,
BOARD/PRODUCT_NAME=*board_product_name*"

A given hardware entity matches a hardware element type if all IDR settings specified in one value of the saPlmHetIdr attribute match fields having corresponding IDR_Area_Names and IDR_Field_Names in an IDR associated with the hardware entity.

If the characteristics of the hardware entity match one version of the hardware element's base type, the PLM Service

- sets the value of the hardware element's `saPlmHECurrEntityPath` runtime attribute to the entity path of the hardware entity and
- sets the value of the hardware element's `saPlmHECurrHEType` runtime attribute to the name of the hardware element type that matches the hardware entity.

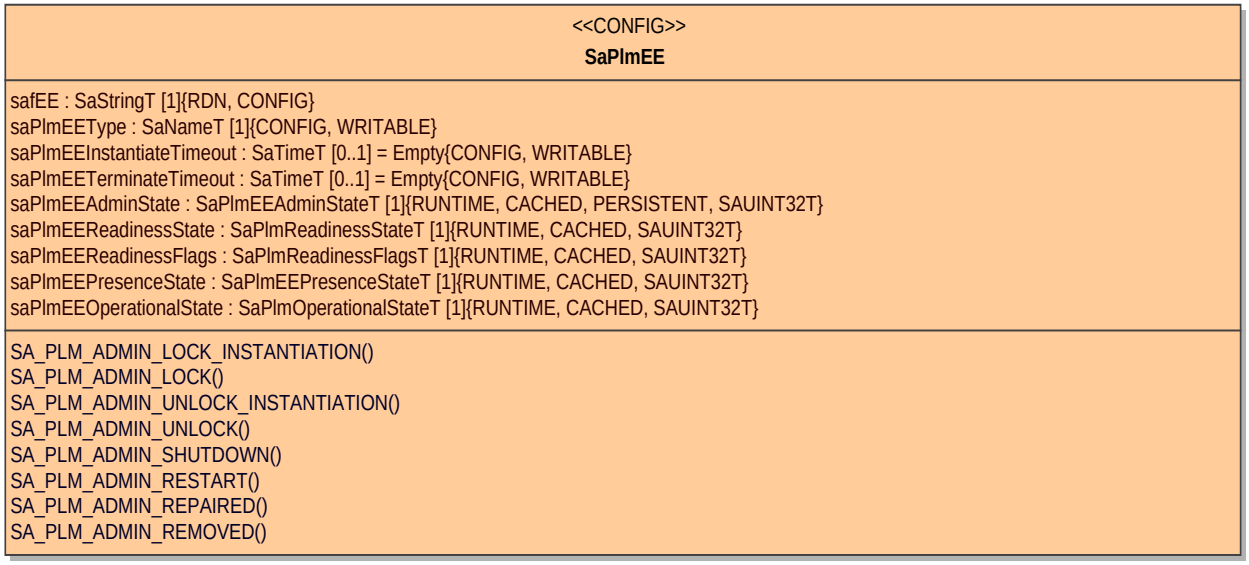
For hardware entities not providing an IDR, the `saPlmHetIdr` attribute should be empty, that is, it should have no strings.

4.6 PLM EE Classes Diagram

The diagram shown in FIGURE 7 contains the following classes:

- `SaPlmEE` — This configuration object class defines configuration and runtime attributes of an execution environment and the operations that can be applied on the execution environment. For each execution environment, an object of this class must be configured, and its `saPlmEEType` attribute must contain the DN of a valid object of the `SaPlmEEType` object class. Additional configuration attributes of an execution environment are defined in the `SaPlmEEType` class.
- `SaPlmEEType` — This configuration object class defines configuration attributes of an execution environment type. All execution environments of the same type share the attribute values defined in the execution environment type configuration.
- `SaPlmHEBaseType` — This configuration object class defines the configuration attributes common to different execution environment types. In particular, a base execution environment type defines the common name of versioned execution environment types. An execution environment type `x` belongs to a base execution environment type `y` based on the DN of `x`, which is the concatenation of the RDN of `x` (representing its version) with the DN of `y`.

FIGURE 7 PLM EE Classes

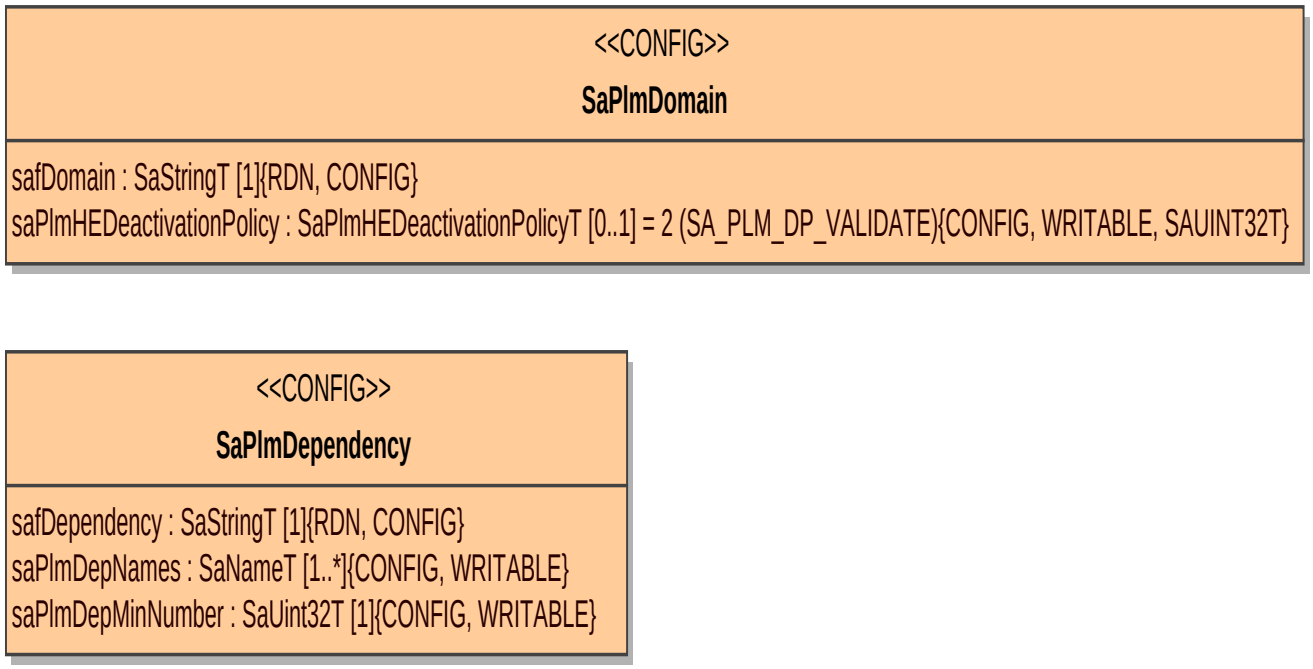


4.7 PLM Other Classes Diagram

The diagram shown in FIGURE 8 contains the following classes:

- `SaPlmDomain` — This configuration object class defines the root object of all hardware elements and execution environments managed by the PLM Service.
- `SaPlmDependency` — This is a configuration association class used to define dependencies between PLM entities. It is used to specify the dependencies that a particular PLM entity has on other PLM entities. Use of the `SaPlmDependency` class is described in Section 3.1.2.3.

FIGURE 8 PLM Other Classes



5 PLM Service Administration API

This section describes the administrative states and API functions that the IMM Service exposes on behalf of the PLM Service to a system administrator. These API functions are described using a 'C' API syntax. The main clients of this administrative API are system management applications, which typically convert system administration commands (invoked from a management station) to the correct administrative API sequence to yield the wanted result that is expected upon execution of the system administration command.

5.1 Include File and Library Name

The appropriate IMM Service header file and the PLM Service header file must be included in the source of an application using the PLM Service administration API; for the name of the IMM Service header file, see [5]).

To use the PLM Service administration API, an application must be bound to the IMM Service library (for the library name, see [5]).

5.2 Type Definitions

The specification of PLM Service Administration API requires the following types, in addition to the ones already described.

5.2.1 SaPlmAdminOperationIdT

```
typedef enum {
     SA_PLM_ADMIN_LOCK                = 1,
     SA_PLM_ADMIN_UNLOCK              = 2,
     SA_PLM_ADMIN_LOCK_INSTANTIATION  = 3,
     SA_PLM_ADMIN_UNLOCK_INSTANTIATION= 4,
     SA_PLM_ADMIN_SHUTDOWN            = 5,
     SA_PLM_ADMIN_REPAIRED            = 6,
     SA_PLM_ADMIN_RESTART             = 7,
     SA_PLM_ADMIN_ACTIVATE            = 8,
     SA_PLM_ADMIN_DEACTIVATE          = 9,
     SA_PLM_ADMIN_RESET               = 10,
     SA_PLM_ADMIN_REMOVED             = 11
} SaPlmAdminOperationIdT;
```

This type defines the identifiers of administrative operations that can be invoked on hardware element or execution environment objects.

5.2.2 Parameter lockOption for the LOCK Administrative Operation

```
#define SA_PLM_ADMIN_LOCK_OPTION "lockOption"

#define SA_PLM_ADMIN_LOCK_OPTION_TRYLOCK    "trylock"
#define SA_PLM_ADMIN_LOCK_OPTION_FORCED     "forced"
```

5.2.3 Parameter restartOption for the Restart Administrative Operation

```
#define SA_PLM_ADMIN_RESTART_OPTION "restartOption"
#define SA_PLM_ADMIN_RESTART_OPTION_ABRUPT       "abrupt"
```

5.3 Interface to the Information Model Management Service

As explained earlier, the administrative API shall be exposed by the IMM Service library.

The administrative APIs are described with the assumption that the PLM Service is an object implementer (runtime owner) for the various administrative operations that will be initiated as a consequence of invoking the `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()` functions (see [5]) with the appropriate `operationId` (described in Section 5.2.1) on the entity designated by the name to which `objectName` points.

The return values explained in the following sections for various administrative operations shall be passed by the `operationReturnValue` parameter, which is provided by the invoker of the `saImmOmAdminOperationInvoke_3()` or `saImmOmAdminOperationInvokeAsync_3()` functions to obtain return codes from the object implementer, which in this case is the PLM Service.

5.4 Administrative Operations

A fair number of administrative operations involve the manipulation of the administrative state. Possible values for the administrative state are unlocked, locked, locked-instantiation, and shutting-down for EEs and unlocked, locked, locked-inactive, and shutting-down for HEs.

For more details on the administrative states of HEs and EEs, see Section 3.1.3.1.2 on page 31 and Section 3.1.3.2.2 on page 39 respectively.

To aid in the description of the administrative operations, FIGURE 9 illustrates the various administrative states and the various operations that are applicable on an EE when it is in a particular administrative state. The abbreviations used in this figure and their meaning are:

- UL = SA_PLM_ADMIN_UNLOCK
- L = SA_PLM_ADMIN_LOCK
- ULI = SA_PLM_ADMIN_UNLOCK_INSTANTIATION
- LI = SA_PLM_ADMIN_LOCK_INSTANTIATION
- SD = SA_PLM_ADMIN_SHUTDOWN

The dotted line in the figure represents the internal (spontaneous) transition corresponding to the completion of the shutting down operation; this transition moves the entity into locked state without further external intervention.

FIGURE 9 Administrative States and Related Operations for PLM EE Entities

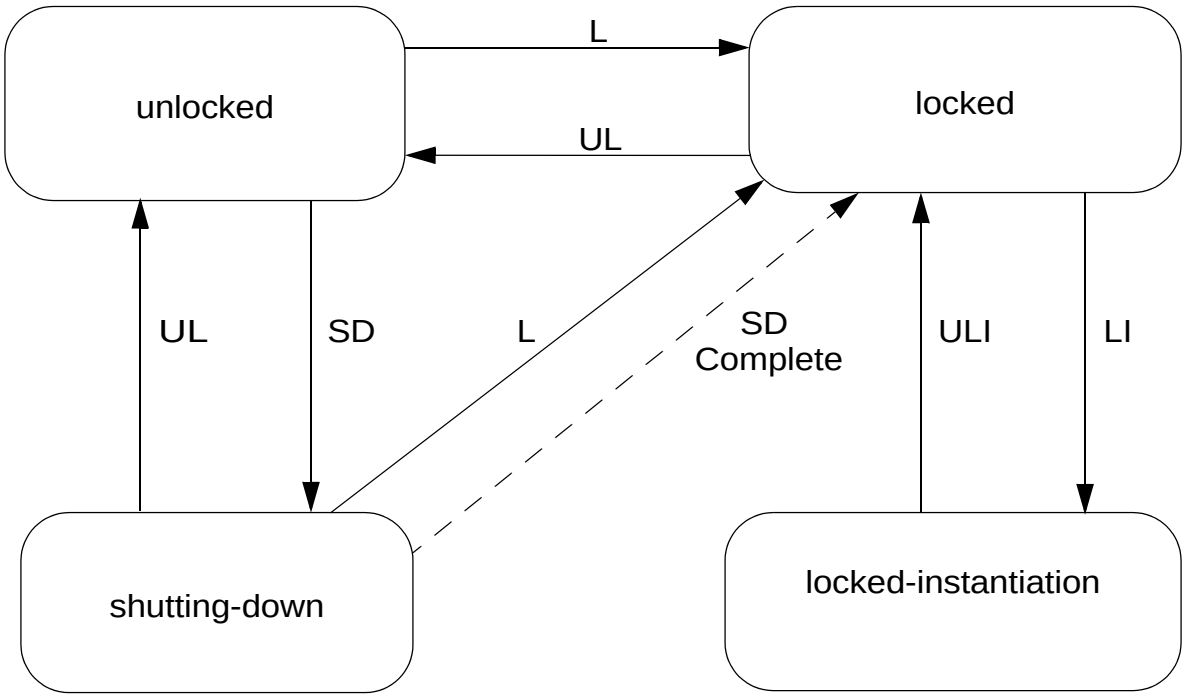
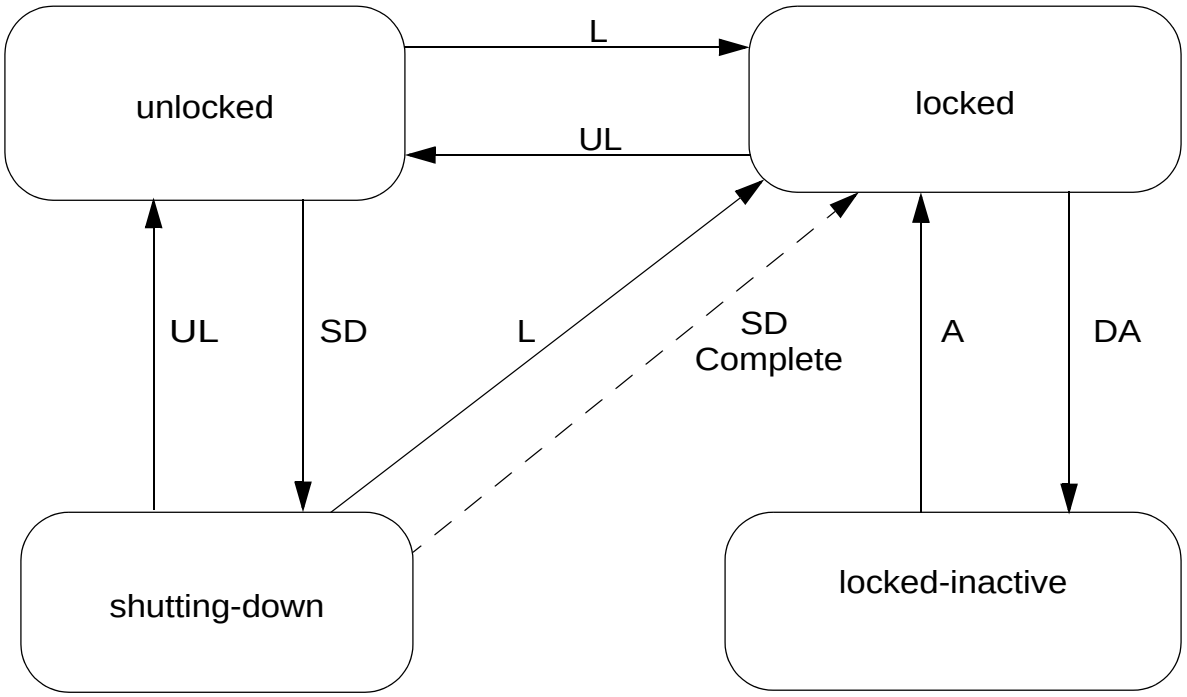


FIGURE 10 illustrates the various administrative states and the various operations that are applicable on an HE when it is in a particular administrative state. This figure uses the same abbreviations as for FIGURE 9, except for A and DA, which replace LI and ULI respectively:

- A = SA_PLM_ADMIN_ACTIVATE
- DA = SA_PLM_ADMIN_DEACTIVATE

FIGURE 10 Administrative States and Related Operations for PLM HE Entities



5.4.1 SA_PLM_ADMIN_UNLOCK

Parameters

`operationId` = `SA_PLM_ADMIN_UNLOCK`, as defined in Section 5.2.1.

`objectName` - [`in`] A pointer to the name of the logical entity (HE or EE) to be unlocked. The name is expressed as a LDAP DN.

Description

This administrative operation sets the administrative state of the logical entity (HE or EE) designated by the name to which `objectName` points to unlocked. For more details regarding the respective status of the logical entities that results as a consequence of invoking this administrative operation on these entities, refer to Section 3.1.3 on page 25.

This administrative operation can be issued on any logical entity, even if the entity is not present in the system (for instance, a hardware element is currently not present or an entity is not instantiated because its parent is locked).

If this operation is invoked on an entity that is already unlocked, there is no change in the status of such an entity, that is, it remains in unlocked state, and the caller is returned a benign `SA_AIS_ERR_NO_OP` error code.

If this operation is invoked on an entity that is in the locked-inactive or locked-instantiation state, there is no change in the status of such an entity, that is, it remains in the locked-inactive or locked-instantiation state, and the caller is returned an `SA_AIS_ERR_BAD_OPERATION` error value.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

`SA_AIS_ERR_TRY_AGAIN` - The service cannot be provided at this time. The client may retry later. This error generally should be returned when the requested action is valid but not currently possible, probably because another operation is acting upon the logical entity on which the administrative operation is invoked. Such an operation can be another administrative operation.

`SA_AIS_ERR_NO_MEMORY` - The PLM Service or a library is out of memory and cannot provide the service.

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

SA_AIS_ERR_NOT_SUPPORTED - This administrative operation is not supported by the type of entity denoted by the name to which objectName points.

SA_AIS_ERR_NO_OP - The invocation of this administrative operation has no effect on the current state of the logical entity, as it is already in unlocked state.

SA_AIS_ERR_BAD_OPERATION - The operation was not successful because the target entity is in locked-instantiation administrative state.

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 because the management-lost readiness flag is set for the entity.

See Also

SA_PLM_ADMIN_LOCK, SA_PLM_ADMIN_SHUTDOWN

5.4.2 SA_PLM_ADMIN_LOCK

Parameters

`operationId` = `SA_PLM_ADMIN_LOCK`, as defined in Section 5.2.1.

`objectName` - [`in`] A pointer to the name of the logical entity (HE or EE) to be locked. The name is expressed as a LDAP DN.

`params` - [`in`] A pointer to a NULL-terminated array of pointers to parameter descriptors. The first parameter descriptor is shown next, and its format is specified in [5].

```
SaStringT operationOption;
operationOption = SA_PLM_ADMIN_LOCK_OPTION_FORCED;
     or
operationOption = SA_PLM_ADMIN_LOCK_OPTION_TRYLOCK;
params[0]->paramName = SA_PLM_ADMIN_LOCK_OPTION;
params[0]->paramType = SA_IMM_ATTR_SASTRINGT;
params[0]->paramBuffer = &operationOption;
```

This parameter descriptor specifies the kind of LOCK operation to be carried out on the object referred to by `objectName`.
The `SA_PLM_ADMIN_LOCK_OPTION` parameter is an optional parameter. If it is not specified, that is, the NULL-terminated array of pointers has NULL in the first element, this administrative function will execute the default LOCK operation (see below).
The types `SA_PLM_ADMIN_LOCK_OPTION`, `SA_PLM_ADMIN_LOCK_OPTION_FORCED`, and `SA_PLM_ADMIN_LOCK_OPTION_TRYLOCK` are defined in Section 5.2.2.

Description

This administrative operation sets the administrative state of the logical entity (HE or EE) designated by the name to which `objectName` points to locked.

When this operation is applied to an entity that is in-service, the PLM Service informs clients of the readiness track API about changes of the readiness state of affected entities. The number of steps in which readiness track callbacks (see Section 3.5.2 on page 85) are invoked depends on the `params` parameter. If `SA_PLM_ADMIN_LOCK_OPTION_TRYLOCK` is specified, consumers may reject the

LOCK operation. In that case, the administrative operation will be rejected, and all affected entities remain in their previous states.

The detailed actions of the LOCK administrative operation are:

- PLM generates a list of all PLM entities that are affected by the LOCK operation.
- The following action is performed if the SA_PLM_ADMIN_LOCK_OPTION_TRYLOCK option is selected: PLM calls readiness track callbacks to validate the operation and to check whether it is possible to evacuate services from the entity before it is locked. If services cannot be relocated, the LOCK request may be rejected. If so, the administrative operation is rejected with SA_AIS_ERR_FAILED_OPERATION; otherwise, the action is performed.
- If the LOCK operation was accepted in the validation step by all users, or the operation was invoked with the default option, PLM calls the readiness track callbacks to start the evacuation of all services. As a result, for example, CLM calls its track callback and AMF switches over the services as necessary. This action is performed if the `trylock` or the default option is selected.
- When all responses have been received for the START step, or the LOCK operation was invoked with the SA_PLM_ADMIN_LOCK_OPTION_FORCED option, the LOCK operation is completed at PLM level by placing the entity into locked state, which includes the termination of all EEs that are children of a locked EE or that depend on a locked hardware element.
- All affected objects are taken out-of-service (readiness state).
- PLM calls readiness track callbacks to inform clients about all entities whose readiness state changed to out-of-service.

This administrative operation can be issued on any logical entity, even if the entity is not present in the system (for instance, a hardware element is currently not present, or an entity is not instantiated because its parent is locked).

If this operation is invoked on an entity that is already locked, there is no change in the status of such an entity, that is, it remains in the locked state, and a benign error value SA_AIS_ERR_NO_OP is returned to the client to convey that the entity in question designated by the name to which `objectName` points is already in locked state.

If this operation is invoked on an entity that is in the locked -inactive or locked-instantiation state, there is no change in the status of such an entity, that is, it remains in the locked-inactive or locked-instantiation state, and the caller is returned an SA_AIS_ERR_BAD_OPERATION error value.

Appendix B.1 on page 167 illustrates a scenario in which a computing blade is physically extracted from the system. The shown sequence is a superset of the actions

necessary for processing a LOCK administrative operation, and it is intended as an aid in understanding PLM actions for a LOCK operation.

Return Values

SA_AIS_OK - The function completed successfully.

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

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The client may retry later. This error generally should be returned when the requested action is valid but not currently possible, probably because another operation is acting upon the logical entity on which the administrative operation is invoked. Such an operation can be another administrative operation.

SA_AIS_ERR_INVALID_PARAM - The SA_PLM_ADMIN_LOCK_OPTION parameter is not set correctly.

SA_AIS_ERR_NO_MEMORY - The PLM Service or a library is out of memory and cannot provide the service.

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

SA_AIS_ERR_NOT_SUPPORTED - This administrative operation is not supported by the type of entity denoted by the name to which objectName points.

SA_AIS_ERR_NO_OP - The invocation of this administrative operation has no effect on the current state of the logical entity, as it is already in locked state.

SA_AIS_ERR_BAD_OPERATION - The operation was not successful because the target entity is in locked-instantiation administrative state.

SA_AIS_ERR_FAILED_OPERATION - The LOCK request was specified with the SA_PLM_ADMIN_LOCK_OPTION_TRYLOCK option, and at least one of the applications rejected the LOCK request. The administrative operation has no effect on the current state of the logical entity.

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 because the management-lost readiness flag is set for the entity.

See Also

SA_PLM_ADMIN_UNLOCK, SA_PLM_ADMIN_SHUTDOWN

5.4.3 SA_PLM_ADMIN_SHUTDOWN

Parameters

`operationId` = `SA_PLM_ADMIN_SHUTDOWN`, as defined in Section 5.2.1.

`objectName` - [`in`] A pointer to the name of the logical entity (HE or EE) to be shut down. The name is expressed as a LDAP DN.

Description

This operation is only applicable to entities that have an administrative state of unlocked. Note, however, that an entity may have an administrative state of unlocked, but a readiness state of stopping, because an ancestor or an entity upon which this entity depends is already in the shutting down state.

When this operation is applied to an entity (HE or EE) with a readiness state of out-of-service, the PLM Service sets the administrative state of the entity immediately to locked. No further processing is required.

When this operation is applied to an entity with a readiness state of in-service or stopping, the administrative state is set to shutting down and the readiness state to stopping, if not already set. If the readiness state is changed, the PLM Service propagates that change through entities that are dependent on the one being shut down and invokes track callbacks in the COMPLETED step for all entity groups that contain entities whose readiness state changed from in-service to stopping.

After this initial state change, the PLM Service invokes track callbacks in the START step for all entity groups that contain entities whose readiness state will change to out-of-service when this shutdown is complete. PLM then waits for the processes receiving those callbacks to respond with the `saPlmReadinessTrackResponse()` function.

This waiting period ends when one of the following conditions is met:

- responses are received for all callbacks,
- the readiness state of the target entity has changed to out-of-service due to other changes in the system, or
- the SHUTDOWN operation is canceled by a subsequent UNLOCK administrative operation.

At this point, if the SHUTDOWN operation was canceled, no further processing is required; otherwise, the administrative state is changed to locked, and the readiness state is changed to out-of-service, if required. If the readiness state was changed as a result of the administrative state changing to locked, then that readiness state change is propagated through entities that are dependent on the one being shut down, and track callbacks in the COMPLETED step are invoked for all entity groups that contain entities whose readiness state changed to out-of-service.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

`SA_AIS_ERR_TRY_AGAIN` - The service cannot be provided at this time. The client may retry later. This error generally should be returned when the requested action is valid but not currently possible, probably because another operation is acting upon the logical entity on which the administrative operation is invoked. Such an operation can be another administrative operation.

`SA_AIS_ERR_NO_MEMORY` - The PLM Service or a library is out of memory and cannot provide the service.

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

`SA_AIS_ERR_NOT_SUPPORTED` - This administrative operation is not supported by the type of entity denoted by the name to which `objectName` points.

`SA_AIS_ERR_NO_OP` - The invocation of this administrative operation has no effect on the current state of the logical entity, as it is already in shutting-down state.

`SA_AIS_ERR_BAD_OPERATION` - The operation was not successful because the target entity is in the locked or locked-instantiation administrative state.

`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 because the management-lost readiness flag is set for the entity.

See Also

`SA_PLM_ADMIN_UNLOCK`, `SA_PLM_ADMIN_LOCK`

5.4.4 SA_PLM_ADMIN_LOCK_INSTANTIATION

Parameters

`operationId` = `SA_PLM_ADMIN_LOCK_INSTANTIATION`, as defined in Section 5.2.1.

`objectName` - [`in`] A pointer to the name of the EE logical entity to be set in the locked-instantiation administrative state. The name is expressed as a LDAP DN.

Description

The invocation of this administrative operation sets the administrative state of the EE logical entity designated by the name to which `objectName` points to locked-instantiation, provided that the EE was in the locked administrative state.

The effect of this operation can only be reversed by issuing an `SA_PLM_ADMIN_UNLOCK_INSTANTIATION` operation on the entity.

The operation can be issued on EEs only. If this operation is issued on an HE, it is rejected with the `SA_AIS_ERR_NOT_SUPPORTED` error code.

This administrative operation can be issued on any EE, even if it has a presence state of `SA_PLM_EE_PRESENCE_UNINSTANTIATED` (for instance, because its parent is locked).

If this operation is invoked by a client on a logical entity that is already in the locked-instantiation state, the status of such an entity does not change, that is, the entity remains in that state, and a benign error value `SA_AIS_ERR_NO_OP` is returned to the client to convey that the state of the concerned entity in question did not change.

If this operation is invoked by a client on a logical entity that is either in the shutting-down or unlocked administrative state, the status of such an entity does not change, that is, the entity remains in the respective state, and the caller is returned an `SA_AIS_ERR_BAD_OPERATION` error value.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The client may retry later. This error generally should be returned when the requested action is valid but not currently possible, probably because another operation is acting upon the logical entity on which the administrative operation is invoked. Such an operation can be another administrative operation.

SA_AIS_ERR_NO_MEMORY - The PLM Service or a library is out of memory and cannot provide the service.

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

SA_AIS_ERR_NOT_SUPPORTED - This administrative operation is not supported by the type of entity denoted by the name to which objectName points.

SA_AIS_ERR_NO_OP - The invocation of this administrative operation has no effect on the current state of the logical entity and it remains in the current state.

SA_AIS_ERR_BAD_OPERATION - The operation was not successful because the target entity is either in the shutting-down or unlocked administrative state.

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 because the management-lost readiness flag is set for the entity.

See Also

SA_PLM_ADMIN_LOCK, SA_PLM_ADMIN_UNLOCK_INSTANTIATION

5.4.5 SA_PLM_ADMIN_UNLOCK_INSTANTIATION

Parameters

`operationId` = `SA_PLM_ADMIN_UNLOCK_INSTANTIATION`, as defined in Section 5.2.1.

`objectName` - [`in`] A pointer to the name of the EE logical entity to be unlocked for instantiation. The name is expressed as a LDAP DN.

Description

The invocation of this administrative operation sets the administrative state of the EE logical entity designated by the name to which `objectName` points to locked.

If the current administrative state of the target entity is `SA_PLM_ADMIN_LOCKED_INSTANTIATION`, the invocation of this operation changes the administrative state to `SA_PLM_ADMIN_LOCKED`. If the entity has an operational state of enabled, and the readiness states of all ancestors and of all other required dependency objects are in-service, the presence state of the target entity is changed to instantiating, and PLM takes actions to cause the EE to start up. Because the administrative state is locked, the readiness state of the target entity remains out-of-service, and the presence states of contained and dependent EEs remain uninstantiated.

The operation can be issued on EEs only. If this operation is issued on an HE, it is rejected with the `SA_AIS_ERR_NOT_SUPPORTED` error code.

This operation is only valid for an EE entity that is currently in the locked-instantiation administrative state.

If this operation is invoked by a client on a logical entity that is already locked, the status of such an entity does not change, that is, it remains in the locked state, and a benign error value `SA_AIS_ERR_NO_OP` is returned to the client to convey that the entity (designated by the name to which `objectName` points) is already in locked state.

If this operation is invoked by a client on a logical entity that is either in the shutting-down or unlocked administrative state, the status of such an entity does not change, that is, the entity remains in the respective state, and the caller is returned an `SA_AIS_ERR_BAD_OPERATION` error value.

Return Values

SA_AIS_OK - The function completed successfully.

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

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The client may retry later. This error generally should be returned when the requested action is valid but not currently possible, probably because another operation is acting upon the logical entity on which the administrative operation is invoked. Such an operation can be another administrative operation.

SA_AIS_ERR_NO_MEMORY - The PLM Service or a library is out of memory and cannot provide the service.

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

SA_AIS_ERR_NOT_SUPPORTED - This administrative operation is not supported by the type of entity denoted by the name to which `objectName` points.

SA_AIS_ERR_NO_OP - The invocation of this administrative operation has no effect on the current state of the logical entity, as it is already in locked state.

SA_AIS_ERR_BAD_OPERATION - The operation was not successful because the target entity is either in the shutting-down or unlocked administrative state.

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 because the management-lost readiness flag is set for the entity.

See Also

SA_PLM_ADMIN_LOCK, SA_PLM_ADMIN_LOCK_INSTANTIATION

5.4.6 SA_PLM_ADMIN_RESTART

Parameters

`operationId` = `SA_PLM_ADMIN_RESTART`, as defined in Section 5.2.1.

`objectName` - [`in`] A pointer to the name of the EE logical entity to be restarted. The name is expressed as a LDAP DN.

`params` - [`in`] A pointer to a NULL-terminated array of pointers to parameter descriptors. The first parameter descriptor is shown next, and its format is specified in [5].

```
SaStringT operationOption;
operationOption = SA_PLM_ADMIN_RESTART_OPTION_ABRUPT;
params[0]->paramName = SA_PLM_ADMIN_RESTART_OPTION;
params[0]->paramType = SA_IMM_ATTR_SASTRINGT;
params[0]->paramBuffer = &operationOption;
```

This parameter descriptor specifies the kind of restart operation to be carried out on the object referred to by `objectName`.
The `SA_PLM_RESTART_OPTION` parameter is an optional parameter. If it is not specified, that is, the NULL-terminated array of pointers has NULL in the first element, this administrative function will execute the default (graceful) restart.
The types `SA_PLM_ADMIN_RESTART_OPTION` and `SA_PLM_ADMIN_RESTART_OPTION_ABRUPT` are defined in Section 5.2.3.

Description

The invocation of this administrative operation involves the termination and immediate re-instantiation of the target EE. This operation is intended to be used as a repair action for the service running on the EE. The processing depends on whether the `SA_PLM_RESTART_OPTION` parameter is specified:

⇒ `SA_PLM_ADMIN_RESTART_OPTION_ABRUPT` is specified:

If `SA_PLM_ADMIN_RESTART_OPTION_ABRUPT` is specified for an EE, the PLM Service uses a specific operation of the parent entity to abruptly terminate and reboot the target EE.
The execution of the operation depends on the capabilities of the parent: If the target EE's parent is an HE, the PLM Service typically uses an HPI operation to reset the HE. If the parent is a virtualization monitor, the PLM Service uses the implementation-specific operation of the virtualization monitor to reboot the tar-

get EE. This operation should be carried out such that none of the siblings of the target EE are affected. After initiating the restart operation, the PLM Service sets the target EE's presence state to instantiating and its readiness state to out-of-service. Additionally, the presence states in dependent EEs are set to uninstantiated and their readiness states to out-of-service. The PLM Service then invokes the `saPlmReadinessTrackCallback()` callback of all processes that requested this callback for the target EE or dependent EEs that change readiness state. The callback is invoked with the `step` parameter set to `SA_PLM_CHANGE_COMPLETED` and `cause` set to `SA_PLM_CAUSE_EE_RESTART`.

⇒ No parameter is specified:

If no parameter is specified, the PLM Service executes a graceful termination and a reboot of the target EE. As opposed to the abrupt restart operation, which uses an operation on the parent entity to force the restart, a graceful termination of an EE is initiated by using an appropriate operation on the target entity itself. That is, the PLM Service instructs the target EE to reboot itself.

As a result of rebooting the target EE, dependent EE entities will necessarily be terminated. Before initiating the reboot operation, the PLM Service sets the presence state of the target EE to terminating and keeps this state until all dependent entities are terminated. As each dependent EE terminates, the PLM Service sets the presence state of the dependent EE to uninstantiated and its readiness state to out-of-service. When the target EE begins to reboot, the PLM Service sets the target EE's presence state to instantiating and its readiness state to out-of-service.

It is implementation-specific how these termination and reboot operations are initiated and coordinated. As the readiness state of any affected EE changes, the PLM Service invokes the `saPlmReadinessTrackCallback()` callback of all processes that requested this callback for the appropriate EE. The callback is invoked with the `step` parameter set to `SA_PLM_CHANGE_COMPLETED` and `cause` set to `SA_PLM_CAUSE_EE_RESTART`.

For both abrupt and graceful restarts, when the start-up of the target EE is completed, its presence state becomes instantiated. If there is not some other reason for it to remain out-of-service, its readiness state is changed to in-service, and the PLM Service invokes the `saPlmReadinessTrackCallback()` callback of all processes that requested this callback for the target EE. The callback is invoked with the `step` parameter set to `SA_PLM_CHANGE_COMPLETED` and `cause` set to `SA_PLM_CAUSE_EE_INSTANTIATED`.

Subsequently, the PLM Service also ensures that all dependent entities are instantiated, as required. When each dependent entity reaches the instantiated presence state, its readiness state becomes in-service, unless there is some other reason causing it to remain out-of-service. If the readiness state of a dependent EE changes, the PLM Service invokes the `saPlmReadinessTrackCallback()` callback of all processes that requested this callback for the EE being set in-service. The callback is invoked with the `step` parameter set to `SA_PLM_CHANGE_COMPLETED` and `cause` set to `SA_PLM_CAUSE_EE_INSTANTIATED`.

After the target EE and all dependent EEs that are to be instantiated reach a presence state of instantiated, the operation returns with `SA_AIS_OK`. If at least one EE was unable to start up within the configured time (specified by the `saPlmEEInstantiateTimeout` attribute of the `SaPlmEE` object class, shown in FIGURE 7 on page 111), the operation returns with `SA_AIS_ERR_FAILED_OPERATION`.

This administrative operation is applicable only to those entities whose presence state is instantiated.

This operation can be issued on EEs only. If this operation is issued on an HE, it is rejected with the `SA_AIS_ERR_NOT_SUPPORTED` error code.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

`SA_AIS_ERR_TRY_AGAIN` - The service cannot be provided at this time. The client may retry later. This error generally should be returned when the requested action is valid but not currently possible, probably because another operation is acting upon the logical entity on which the administrative operation is invoked. Such an operation can be another administrative operation. This error code is also returned when the PLM Service was unable to initiate the restart operation because the management-lost readiness flag is set for the entity.

`SA_AIS_ERR_INVALID_PARAM` - The `SA_PLM_ADMIN_RESTART_OPTION` parameter is not set correctly.

`SA_AIS_ERR_NO_MEMORY` - The PLM Service or a library is out of memory and cannot provide the service.

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

SA_AIS_ERR_BAD_OPERATION - The target logical entity for this operation identified by the name to which `objectName` points could not be restarted for various reasons like the presence state of the entity to be restarted being other than instantiated.

SA_AIS_ERR_NOT_SUPPORTED - This administrative operation is not supported by the type of entity denoted by the name to which `objectName` points.

SA_AIS_ERR_FAILED_OPERATION - The requested operation was accepted and applied to the object; however, not all affected EEs were successfully restarted.

See Also

None

5.4.7 SA_PLM_ADMIN_DEACTIVATE

Parameters

`operationId` = `SA_PLM_ADMIN_DEACTIVATE`, as defined in Section 5.2.1.

`objectName` - [`in`] A pointer to the name of the HE logical entity whose administrative state is to be transitioned to locked-inactive. The name is expressed as a LDAP DN.

Description

The invocation of this administrative operation sets the administrative state of the HE logical entity designated by the name to which `objectName` points to locked-inactive, provided that the logical entity's administrative state was locked.
When setting the administrative state of an HE to locked-inactive, the PLM Service will initiate the required actions to deactivate the hardware entity mapped to that HE, if its presence state is not already inactive or not present.

The operation can be issued on HEs only. If this operation is issued on an EE, it is rejected with the `SA_AIS_ERR_NOT_SUPPORTED` error code.

The effect of this operation on a logical entity can be reversed only by issuing an `SA_PLM_ADMIN_ACTIVATE` operation on the logical entity.

This administrative operation can be issued on any HE logical entity, even if its presence state is set to not present.

If this operation is invoked by a client on a logical entity that is already in the locked-inactive state, the status of such an entity does not change, that is, the entity remains in that state, and a benign error value `SA_AIS_ERR_NO_OP` is returned to the client to convey that the state of the concerned entity in question did not change.

If this operation is invoked by a client on a logical entity that is either in the shutting-down or unlocked administrative state, the status of such an entity does not change, that is, the entity remains in the respective state, and the caller is returned an `SA_AIS_ERR_BAD_OPERATION` error value.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

`SA_AIS_ERR_TRY_AGAIN` - The service cannot be provided at this time. The client may retry later. This error generally should be returned when the requested action is valid but not currently possible, probably because another operation is acting upon the logical entity on which the administrative operation is invoked. Such an operation can be another administrative operation.

`SA_AIS_ERR_NO_MEMORY` - The PLM Service or a library is out of memory and cannot provide the service.

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

`SA_AIS_ERR_NOT_SUPPORTED` - This administrative operation is not supported by the type of entity denoted by the name to which `objectName` points.

`SA_AIS_ERR_NO_OP` - The invocation of this administrative operation has no effect on the current state of the logical entity and it remains in the current state.

`SA_AIS_ERR_BAD_OPERATION` - The operation was not successful because the target entity is either in the shutting-down or unlocked administrative state.

`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 because the management-lost readiness flag is set for the entity.

See Also

`SA_PLM_ADMIN_LOCK`, `SA_PLM_ADMIN_ACTIVATE`

5.4.8 SA_PLM_ADMIN_ACTIVATE

Parameters

`operationId` = `SA_PLM_ADMIN_ACTIVATE`, as defined in Section 5.2.1.

`objectName` - [`in`] A pointer to the name of the HE logical entity whose administrative state is to be transitioned to locked. The name is expressed as a LDAP DN.

Description

The invocation of this administrative operation transitions the administrative state of the HE logical entity designated by the name to which `objectName` points to locked.

This administrative operation is only valid when the current administrative state of the target entity is locked-inactive. When this operation is processed, the administrative state of the target entity is set to locked, and the PLM Service takes any required actions to begin activation of a hardware entity mapped to the addressed HE entity, as well as contained entities, provided they do not have an administrative state of locked-inactive. Generally, this will result in a change of the presence state of the affected entities to activating, and eventually to active. However, the readiness states will remain out-of-service, because the new administrative state of the target entity is locked.

The operation can be issued on HEs only. If this operation is issued on an EE, it is rejected with the `SA_AIS_ERR_NOT_SUPPORTED` error code.

This administrative operation can be issued on any HE logical entity, even if its presence state is set to `SA_PLM_HE_PRESENCE_NOT_PRESENT`.

If this operation is invoked by a client on a logical entity that is already locked, the status of such an entity does not change, that is, it remains in the locked state, and a benign error value `SA_AIS_ERR_NO_OP` is returned to the client to convey that the entity (designated by the name to which `objectName` points) is already in locked state.

If this operation is invoked by a client on a logical entity that is either in the shutting-down or unlocked administrative state, the status of such an entity does not change, that is, the entity remains in the respective state, and the caller is returned an `SA_AIS_ERR_BAD_OPERATION` error value.

Return Values

SA_AIS_OK - The function completed successfully.

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

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The client may retry later. This error generally should be returned when the requested action is valid but not currently possible, probably because another operation is acting upon the logical entity on which the administrative operation is invoked. Such an operation can be another administrative operation.

SA_AIS_ERR_NO_MEMORY - The PLM Service or a library is out of memory and cannot provide the service.

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

SA_AIS_ERR_NOT_SUPPORTED - This administrative operation is not supported by the type of entity denoted by the name to which `objectName` points.

SA_AIS_ERR_NO_OP - The invocation of this administrative operation has no effect on the current state of the logical entity, as it is already in locked state.

SA_AIS_ERR_BAD_OPERATION - The operation was not successful because the target entity is either in the shutting-down or unlocked administrative state.

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 because the management-lost readiness flag is set for the entity.

See Also

SA_PLM_ADMIN_LOCK, SA_PLM_ADMIN_DEACTIVATE

5.4.9 SA_PLM_ADMIN_RESET

Parameters

`operationId` = `SA_PLM_ADMIN_RESET`, as defined in Section 5.2.1.

`objectName` - [`in`] A pointer to the name of the HE logical entity to be reset. The name is expressed as a LDAP DN.

Description

The invocation of this administrative operation is intended to be used to bring a failed entity back to a known state.

The PLM Service typically uses the `saHpiResourceResetStateSet()` HPI function to perform a reset on the target entity. The operation also results in dependent EEs becoming uninstantiated and subsequently re-instantiated. If the target entity can be activated, and all dependent EEs are successfully instantiated, the success of the administrative operation is signalized with `SA_AIS_OK`.

When the reset is initiated, the PLM Service sets the presence state of the target HE to inactive and its readiness state to out-of-service. Additionally, the presence state of dependent entities become inactive or uninstantiated, and their readiness states become out-of- service. The PLM Service invokes the `saPlmReadinessTrackCallback()` callback function of all processes that requested this callback for any of the entities that changed readiness state. The callback is invoked with the `step` parameter set to `SA_PLM_CHANGE_COMPLETED` and `cause` set to `SA_PLM_CAUSE_HE_RESET`.

When the target entity has restarted, the PLM Service sets its presence state to active and its readiness state to in-service, if there is no other reason to have it out-of-service. Additionally, the presence states and readiness states of dependent HE entities are changed to active and in-service, as appropriate. The PLM Service invokes again the `saPlmReadinessTrackCallback()` callback function of all processes that requested this callback for any of the entities that changed readiness state to in-service. The callback is invoked with the `step` parameter set to `SA_PLM_CHANGE_COMPLETED` and `cause` set to `SA_PLM_CAUSE_HE_ACTIVATED`. Subsequently, the PLM Service also ensures that dependent EE entities are instantiated, as required. When each dependent EE reaches the instantiated presence state, its readiness state becomes in-service, unless there is some other reason causing it to remain out-of-service. If the readiness state changes, the PLM Service invokes the `saPlmReadinessTrackCallback()` callback function of all processes that requested this callback for the EE being set in-service. The callback is invoked with the `step` parameter set to `SA_PLM_CHANGE_COMPLETED`, `cause` set to

SA_PLM_CAUSE_HE_ACTIVATED, and with rootCauseEntity identifying the target HE that was reset.

This administrative operation is applicable only to those entities whose presence state is active; otherwise, SA_AIS_ERR_BAD_OPERATION is returned.

This operation can be issued on HEs only. If this operation is issued on an EE, it is rejected with the SA_AIS_ERR_NOT_SUPPORTED error code.

Return Values

SA_AIS_OK - The function completed successfully.

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

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The client may retry later. This error generally should be returned when the requested action is valid but not currently possible, probably because another operation is acting upon the logical entity on which the administrative operation is invoked. Such an operation can be another administrative operation. This error code is also returned when the PLM Service was unable to initiate the reset operation due to circumstances for which the management-lost readiness flag was set for the entity.

SA_AIS_ERR_NO_MEMORY - The PLM Service or a library is out of memory and cannot provide the service.

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

SA_AIS_ERR_BAD_OPERATION - The target logical entity for this operation identified by the name to which objectName points could not be reset for various reasons like the presence state of the entity to be reset being inactive.

SA_AIS_ERR_NOT_SUPPORTED - This administrative operation is not supported by the type of entity denoted by the name to which objectName points.

See Also

None

5.4.10 SA_PLM_ADMIN_REPAIRED

Parameters

`operationId` = `SA_PLM_ADMIN_REPAIRED`, as defined in Section 5.2.1.

`objectName` - [`in`] A pointer to the name of the logical entity (EE or HE) to be repaired. The name is expressed as a LDAP DN.

Description

This administrative operation is used to clear the disabled operational state of a PLM entity (HE or EE), after it has been successfully mended to declare it as repaired. The administrator uses this command to indicate that the PLM Service should try to take the entity back to service.

This operation can be used, for instance, to reverse the effect of the call of the function `saPlmEntityReadinessImpact()` by an application to report a failure condition.

If the PLM Service does not detect a pending fault condition, it sets the operational state of the entity to enabled and clears the isolate-pending and the imminent-failure readiness flags. The administrative operation than returns with `SA_AIS_OK`.

If all other preconditions are met, the PLM Service brings the entity in-service and initiates the appropriate actions. The PLM Service invokes the `saPlmReadinessTrackCallback()` callback function of all processes that requested this callback with the `step` parameter set to `SA_PLM_CHANGE_COMPLETED` and `cause` set to `SA_PLM_CAUSE_FAILURE_CLEARED`.
If PLM still detects a pending fault condition, it returns `SA_AIS_ERR_BAD_OPERATION`.

This administrative operation can be issued on any configured entity, even if the entity is inactive.

Return Values

SA_AIS_OK - The function completed successfully.

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

SA_AIS_ERR_TRY_AGAIN - The service cannot be provided at this time. The client may retry later. This error generally should be returned when the requested action is valid but not currently possible, probably because another operation is acting upon the logical entity on which the administrative operation is invoked.

SA_AIS_ERR_NO_MEMORY - The PLM Service or a library is out of memory and cannot provide the service.

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

SA_AIS_ERR_NOT_SUPPORTED - This administrative operation is not supported by the type of entity denoted by the name to which `objectName` points.

SA_AIS_ERR_NO_OP - The invocation of this administrative operation has no effect on the current state of the logical entity, as it is already enabled.

SA_AIS_ERR_BAD_OPERATION - The invocation of this administrative operation was unsuccessful because there is still an error condition. The operational state could not be set to enabled.

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 because the management-lost readiness flag is set for the entity.

See Also

None

5.4.11 SA_PLM_ADMIN_REMOVED

Parameters

`operationId` = `SA_PLM_ADMIN_REMOVED`, as defined in Section 5.2.1.

`objectName` - [`in`] A pointer to the name of the logical entity (HE or EE) that has been removed. The name is expressed as a LDAP DN.

Description

This administrative operation is used when an entity has been removed from the system, while the PLM Service has no management capabilities to detect the removal. It is applicable only to those logical entities (HE or EE) for which the management-lost readiness flag is set.

This operation will set the HE identified by the name to which `objectName` points and its child HEs to out-of-service, not-present and all its child-EEs to out-of-service, uninstantiated.
The operation issued on an EE identified by the name to which `objectName` points will set the EE and all its child-EEs to out-of-service, uninstantiated. Readiness flags of all affected entities are cleared. Users of the track interface are called as appropriate.

Return Values

`SA_AIS_OK` - The function completed successfully.

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

`SA_AIS_ERR_TRY_AGAIN` - The service cannot be provided at this time. The client may retry later. This error generally should be returned when the requested action is valid but not currently possible, probably because another operation is acting upon the logical entity on which the administrative operation is invoked. Such an operation can be another administrative operation.

`SA_AIS_ERR_NO_MEMORY` - The PLM Service or a library is out of memory and cannot provide the service.

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

`SA_AIS_ERR_NOT_SUPPORTED` - This administrative operation is not supported by the type of entity denoted by the name to which `objectName` points.

`SA_AIS_ERR_BAD_OPERATION` - The management-lost readiness flag is not set for the target logical entity that is identified by the name to which `objectName` points.

`SA_AIS_ERR_NO_OP` - The invocation of this administrative operation has no effect on the current state of the logical entity, as it is already in not-present or uninstantiated state.

`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 because the management-lost readiness flag is set for the entity.

See Also

None

6 PLM Service Alarms and Notifications

The Platform Management Service produces alarms and notifications to convey important information regarding the operational and functional state of the objects under its control to an administrator or a management system.

These reports vary in perceived severity and include alarms, which potentially require an operator intervention, and notifications which signify important state or object changes. A management entity should regard notifications, but they do not necessarily require an operator intervention.

The vehicle to be used for producing alarms and notifications is the Notification Service of the Service Availability™ Forum (abbreviated as NTF, see [4]), and hence the various notifications are partitioned into categories, as described in this service.

In some cases, this specification uses the word “Unspecified” for values of attributes that the vendor is at liberty to set to whatever makes sense in the vendor’s context, and the SA Forum has no specific recommendation regarding such values. Such values are generally optional from the CCITT Recommendation X.733 perspective (see [10]).

6.1 Setting Common Attributes

The following attributes of the notifications presented in Section 6.2 are not shown in their description, as the generic description presented here applies to all of them:

- Notification Id - Depending on the Notification Service function used to send the notification, this attribute is either implicitly set by the Notification Service or provided by the caller.
- Notifying Object - DN of the entity generating the notification. This name must conform to the SA Forum AIS naming convention and must contain at least the `safApp` RDN value portion of the DN set to the specified standard RDN value of the SA Forum AIS Service generating the notification, that is, `safPlmService`. For details on the AIS naming convention, refer to [2].
- Event Time - This attribute contains the time when the Platform Management Service detected the event leading to the notification.
- Correlated Notifications - Correlation ids are supplied to correlate notifications that have been generated because of a related cause. The correlated notifications attribute should include
  - in the first position the root notification identifier of the related tree of notifications as described in the Notification Service specification (see [4]),
  - in the second position the parent notification identifier of the same tree,

- in the third position the notification identifier of the sibling notification, if any. This sibling notification is the opening pair of the current notification such as the alarm that is being cleared or the start of an administrative operation or a configuration change that has ended.

If any of these notifications is unknown, the SA_NTF_IDENTIFIER_UNUSED value must be used. This value may be omitted in trailing positions.

The following note applies to all notifications presented in Section 6.2:

- Notification Class Identifier - The vendorId field of the SaNtfClassIdT data structure must be set to SA_NTF_VENDOR_ID_SAF, and the majorId field must be set to SA_SVC_PLM (as defined in the SaServicesT enumeration in [2]) for all notifications that follow the standard formats described in this specification. The minorId field will vary based on the specific notification.

  An implementation of the Platform Management Service may also produce notifications with an implementation-specific format. In particular, HPI event notifications may use implementation-specific data formats. For these implementation-specific notifications, the vendorId portion of the SaNtfClassIdT data structure must be set to a value that identifies the vendor defining the format. The values of majorId and minorId will vary as defined by the vendor.

The PLM Service should provide close-to-source notification suppression, as described in [4]. That is, the PLM Service should save the efforts necessary to construct currently suppressed notifications.

6.2 Platform Management Service Notifications

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

6.2.1 Platform Management Service Alarms

6.2.1.1 Hardware Element Alarm

Description

By using its interface to HPI or by other implementation-specific means, the PLM Service has detected that an alarm condition (for instance, a hardware fault) exists for a hardware entity modeled as a PLM hardware element (HE) object.

Clearing Method

(1) By exchanging faulty hardware, or taking other repair procedure or

(2) manual

- by taking an appropriate administrative action,
- by taking other appropriate action, or

(3) automatic, as a result of subsequent events: issue a subsequent alarm notification with `SA_NTF_SEVERITY_CLEARED` perceived severity to convey that the PLM Service has detected that the alarm condition no longer exists. For instance, the PLM Service has detected that the faulty hardware entity ha been removed.

Table 3 Hardware Element Alarm

NTF Attribute Name	Mandatory/ Optional	Specified Value
Event Type	Mandatory	`SA_NTF_ALARM_EQUIPMENT`
Notification Object	Mandatory	LDAP DN of the HE object associated with the hardware entity
Notification Class Identifier	NTF-Internal	`minorId` = `SA_PLM_NTFID_HE_ALARM`
Additional Text	Optional	"Hardware element <LDAP DN of HE> alarm"

Additional Information	Optional	`infoId` = `SA_PLM_AI_ENTITY_PATH`
		`infoType` = `SA_NTF_VALUE_STRING`
		`infoValue` = HPI entity path, that is, the contents of HE object's `saPlmHECurrEntityPath` attribute

Probable Cause	Mandatory	Applicable value from enum `SaNtfProbableCauseT` in [4]
Specific Problems	Optional	Unspecified
Perceived Severity	Mandatory	Applicable value from enum `SaNtfSeverityT` in [4]
Trend Indication	Optional	Unspecified
Threshold Information	Optional	Unspecified
Monitored Attributes	Optional	Unspecified
Proposed Repair Actions	Optional	Unspecified

6.2.1.2 Execution Environment Alarm

Description

By using implementation-specific means, the PLM Service has detected that an alarm condition (for instance, a fault condition) exists for an execution environment (EE) object.

Clearing Method

(1) By using a problem-specific repair procedure,

(2) manual,

- by taking an appropriate administrative action,
- by taking other appropriate action, or

(3) automatic, as a result of subsequent events: issue a subsequent alarm notification with `SA_NTF_SEVERITY_CLEARED` perceived severity to convey that the PLM Service has detected that the alarm condition no longer exists.

Table 4 Execution Environment Alarm

NTF Attribute Name	Mandatory/ Optional	Specified Value
Event Type	Mandatory	`SA_NTF_ALARM_ENVIRONMENT`
Notification Object	Mandatory	LDAP DN of the EE object
Notification Class Identifier	NTF-Internal	`minorId` = `SA_PLM_NTFID_EE_ALARM`
Additional Text	Optional	"Execution environment <LDAP DN of EE> alarm"
Additional Information	Optional	Unspecified
Probable Cause	Mandatory	Applicable value from enum `SaNtfProbableCauseT` in [4]
Specific Problems	Optional	Unspecified
Perceived Severity	Mandatory	Applicable value from enum `SaNtfSeverityT` in [4]
Trend Indication	Optional	Unspecified
Threshold Information	Optional	Unspecified
Monitored Attributes	Optional	Unspecified
Proposed Repair Actions	Optional	Unspecified

6.2.1.3 Hardware Element Security Alarm

Description

By using its interface to HPI or by other implementation-specific means, the PLM Service has detected that a security alarm condition exists for a hardware entity modeled as a PLM hardware element (HE) object.

Clearing Method

(1) Manual, after taking the appropriate administrative action or

(2) issue a subsequent security alarm notification with `SA_NTF_SEVERITY_CLEARED` perceived severity to convey that the PLM Service has detected that the security alarm condition no longer exists.

Table 5 Hardware Element Security Alarm

NTF Attribute Name	Mandatory/ Optional	Specified Value
Event Type	Mandatory	`SA_NTF_PHYSICAL_VIOLATION`
Notification Object	Mandatory	LDAP DN of the HE object associated with the hardware entity
Notification Class Identifier	NTF-Internal	`minorId` = `SA_PLM_NTFID_HE_SEC_ALARM`
Additional Text	Optional	"Hardware element <LDAP DN of HE> security alarm"

Additional Information	Optional	`infoId` = `SA_PLM_AI_ENTITY_PATH`
		`infoType` = `SA_NTF_VALUE_STRING`
		`infoValue` = HPI entity path, that is, the contents of HE object's `saPlmHECurrEntityPath` attribute.

Cause	Mandatory	Applicable value from enum `SaNtfProbableCauseT` in [4]
Severity	Mandatory	Applicable value from enum `SaNtfSeverityT` in [4]
Detector	Mandatory	This field should uniquely indicate how the violation was detected. It can contain, for example, a string identifying an HPI sensor.
Service User	Mandatory	This field must represent the identity of the SA Forum Service user. When the identity of the user cannot be determined, the value 'Unidentified', represented as a string, must be used.
Service Provider	Mandatory	Typically, HPI detected the violation: "`safApp=safHpiService`" In other cases, PLM reports the detecting service or: "`safApp=safPlmService`".

6.2.1.4 Execution Environment Security Alarm

Description

By using implementation-specific means, the PLM Service has detected that a security alarm condition exists for an operating environment modeled as a PLM execution environment (EE) object.

Clearing Method

(1) Manual, after taking the appropriate administrative action or

(2) issue a subsequent security alarm notification with `SA_NTF_SEVERITY_CLEARED` perceived severity to convey that the PLM Service has detected that the security alarm condition no longer exists.

Table 6 Execution Environment Security Alarm

NTF Attribute Name	Mandatory/ Optional	Specified Value
Event Type	Mandatory	`SA_NTF_OPERATION_VIOLATION`
Notification Object	Mandatory	LDAP DN of the EE object.
Notification Class Identifier	NTF-Internal	`minorId` = `SA_PLM_NTFID_EE_SEC_ALARM`
Additional Text	Optional	"Execution environment <LDAP DN of EE> security alarm"
Additional Informa-tion	Optional	Unspecified
Cause	Mandatory	Applicable value from enum `SaNtfProbableCauseT` in [4]
Severity	Mandatory	Applicable value from enum `SaNtfSeverityT` in [4]
Detector	Optional	This field should uniquely indicate how the violation was detected. It can contain, for example, a string identifying the program that detected the violation.
Service User	Optional	This field must represent the identity of the SA Forum Service user. When the identity of the user cannot be determined, the value 'Unidentified', represented as a string, must be used.
Service Provider	Optional	Typically, PLM detects the violation: "`safApp=safPlmService`". In other cases, PLM reports the DN of the detecting service.

6.2.1.5 Unmapped Hardware Entity Alarm

Description

The PLM Service, through its interface with HPI or other implementation-specific means, has discovered the presence of a hardware entity that is not mappable to any configured hardware element (HE).

It is possible that a group of HPI entities such as a computing blade plus contained components on that blade, are all mapped to a single HE. Therefore, it is not required for PLM to generate an “unmapped hardware entity” alarm for every discovered HPI entity path that does not map directly to an entity path in an HE. This notification should be issued only when a hardware entity is discovered that is not associated with any HE in the PLM configuration.

Similarly, when this notification is issued for a particular hardware entity, it is not required, nor recommended, that it be issued for all contained entities. For example, if a notification is issued reporting that a discovered computing blade is unmapped, it is not required to issue notifications reporting that individual components on that blade are also unmapped.

The probable cause and perceived severity attributes in the notification may be set to appropriate values determined by the PLM implementation. In most cases, it is expected that the probable cause will be `SA_NTF_CONFIGURATION_OR_CUSTOMIZATION_ERROR` and the perceived severity will be `SA_NTF_SEVERITY_INDETERMINATE`.

Clearing Method

(1) By removing hardware of a wrong types or by changing the configuration such that the hardware can be mapped, or

(2) manual, by taking other appropriate action, or

(3) automatic, as a result of subsequent events: issue a subsequent alarm notification with `SA_NTF_SEVERITY_CLEARED` perceived severity to convey that the PLM Service has resolved the mapping issue. For example, the PLM Service detected that the unmapped hardware entity has been removed from the system or that a reconfiguration of HE objects has been made such that the hardware entity now can be mapped to an HE.

Table 7 Unmapped Hardware Entity Alarm

NTF Attribute Name	Mandatory/ Optional	Specified Value
Event Type	Mandatory	`SA_NTF_ALARM_EQUIPMENT`
Notification Object	Mandatory	LDAP DN of the nearest HE object that is a candidate parent of the unmapped entity, or if there is no candidate parent, the LDAP DN of the PLM Domain object. An HE object is a candidate parent if the `saPlmHECurrEntityPath` attribute of the HE object matches the tail of the entity path of the unmapped hardware entity.
Notification Class Identifier	NTF-Internal	`minorId` = `SA_PLM_NTFID_UNMAPPED_HE_ALARM`
Additional Text	Optional	"Unmapped hardware entity"

Additional Information	Optional	`infoId` = `SA_PLM_AI_ENTITY_PATH`
		`infoType` = `SA_NTF_VALUE_STRING`
		`infoValue` = HPI entity path for unmapped hardware entity

Probable Cause	Mandatory	Applicable value from enum `SaNtfProbableCauseT` in [4]
Specific Problems	Optional	Unspecified
Perceived Severity	Mandatory	Applicable value from enum `SaNtfSeverityT` in [4]
Trend Indication	Optional	Unspecified
Threshold Information	Optional	Unspecified
Monitored Attributes	Optional	Unspecified
Proposed Repair Actions	Optional	Unspecified

6.2.2 Platform Management Service State Change Notifications

6.2.2.1 PLM Entity State Change Notification

Description

The administrative, HA, operational, presence, or readiness state or flags of an HE or EE object has changed. If multiple states of an object change together, a single notification including all associated state changes shall be generated for that object. For example, if the operational state for an HE object that has a readiness state of `SA_PLM_READINESS_IN_SERVICE` changes to `SA_PLM_OPERATIONAL_DISABLED`, the readiness state changes to `SA_PLM_READINESS_OUT_OF_SERVICE` as part of the same operation. These two state changes should be included in a single state change notification.

An object changes state either as the result of an administrative operation, a detected or reported status change, or as a result of a change in state of another PLM entity object, either an ancestor PLM entity or a PLM entity with a configured dependency relationship. A change of state of an object due to an ancestor or dependency object changing state is indicated in the notification with a specific notification class identifier (NCI) and with the inclusion of the LDAP name of the 'root' state change object in the additional information field. The 'root' state change object is not necessarily the direct parent or dependency PLM entity for the object changing state; it is the 'initial' PLM entity that changed state resulting in this state change, that is, the object that changed state for some reason other than the fact that an ancestor or a dependency object changed state.

Table 8 PLM Entity State Change Notification

NTF Attribute Name	Mandatory/Optional	Specified Value
Event Type	Mandatory	`SA_NTF_OBJECT_STATE_CHANGE`
Notification Object	Mandatory	LDAP DN of the PLM entity object whose state changed
Notification Class Identifier	NTF-Internal	`minorId` = `SA_PLM_NTFID_STATE_CHANGE_ROOT` for root state change notification, or `minorId` = `SA_PLM_NTFID_STATE_CHANGE_DEP` for child or dependent state change notification
Additional Text	Optional	Unspecified

Table 8 PLM Entity State Change Notification (Continued)

NTF Attribute Name	Mandatory/Optional	Specified Value

Additional Information	Mandatory when NCI `minorId` = `SA_PLM_NTFID_STATE_CHANGE_DEP`	`infoId` = `SA_PLM_AI_ROOT_OBJECT`
		`infoType` = `SA_NTF_VALUE_LDAP_NAME`
		`infoValue` = LDAP DN of root state change PLM entity object

Source Indicator	Mandatory	Applicable value from enum `SaNtfSourceIndicatorT` in [4]. When the state change is due to an ancestor or dependency object changing state, the Source Indicator is the same as the value in the notification generated for the root object for the state change.
`numStateChanges` field of `SaNtfStateChangeNotificationT` in [4]	Mandatory	Number of state values that changed together in this object. For example, the value 2 is used if both the operational and the readiness states changed as the result of a failure.
`changedStates` array of `SaNtfStateChangeNotificationT` in [4]	Mandatory	Array of `SaNtfStateChangeT` structures (see [4]) for each state in the object that has changed, each entry including the four next attributes
Changed State Attribute Identifier (`stateId` field of `SaNtfStateChangeT`)	Mandatory	Applicable value from enum `SaPlmStateT` (see Section 3.3.15.3) to identify which state has changed

Table 8 PLM Entity State Change Notification (Continued)

NTF Attribute Name	Mandatory/Optional	Specified Value
Old Attribute Value Present (`oldStatePresent` field of `SaNtfStateChangeT`)	Mandatory	Boolean value indicating whether or not the old attribute value is included in the `SaNtfStateChangeT` structure
Old Attribute Value (`oldState` field of `SaNtfStateChangeT`)	Optional	Applicable value from enum `SaPlmHEAdminStateT` (Section 3.3.2), `SaPlmEEAdminStateT` (Section 3.3.3), `SaPlmOperationalStateT` (Section 3.3.4), `SaPlmHEPresenceStateT` (Section 3.3.5), `SaPlmEEPresenceStateT` (Section 3.3.6), `SaPlmReadinessStateT` (Section 3.3.7), or `SaPlmReadinessFlagsT` (Section 3.3.8), depending on the value of the `stateId` field. When reporting the value of readiness flags, all readiness flags which were originally set are included.
New Attribute Value (`newState` field of `SaNtfStateChangeT`)	Mandatory	Applicable value from enum `SaPlmHEAdminStateT` (Section 3.3.2), `SaPlmEEAdminStateT` (Section 3.3.3), `SaPlmOperationalStateT` (Section 3.3.4), `SaPlmHEPresenceStateT` (Section 3.3.5), `SaPlmEEPresenceStateT` (Section 3.3.6), `SaPlmReadinessStateT` (Section 3.3.7), or `SaPlmReadinessFlagsT` (Section 3.3.8), depending on the value of the `stateId` field. When reporting the value of readiness flags, all readiness flags which were originally set are included.

6.2.3 HPI Events Notifications

In addition to the notifications described above, a PLM implementation may produce HPI event notifications for events received from HPI implementations. These notifications are intended to provide only an audit-trail of the detail of HPI events received by PLM, and should not be expected to be a replacement for any of the required notifications described in the previous sections.

For example, when a HPI Hot Swap event is received by a PLM implementation, an HPI event notification may be generated to record the reception of that event. However, the processing of that event will normally result in the associated PLM HE object changing its hot swap state. A byproduct of this state change is the generation of a state change notification. This state change notification is the "primary" notification that other processes should use to track hot swap activity. Similarly, processes wanting to be informed of hardware alarm conditions should receive PLM alarm notifications rather than HPI event notifications for sensor changes.

HPI event notifications are formatted as described next.

Table 9 HPI Event Notification

NTF Attribute Name	Mandatory/ Optional	Specified Value
Event Type	Mandatory	Defined below
Notification Object	Mandatory	LDAP DN for HPI implementation. See below for more details.

Notification Class Identifier	NTF-Internal	If using a standard data format:
		`vendorId` = `SA_NTF_VENDOR_ID_SAF`
		`majorId` = `SA_SVC_PLM`
		`minorId` = applicable value defined below for data format.
		
		If using an implementation-specific data format:
		`vendorId` = <identifier for vendor defining data format>
		`majorId` = <defined by vendor>
		`minorId` = <defined by vendor>.

Additional Text	Optional	Unspecified
Additional Information	Mandatory + Optional	Two to four fields, as defined below, for domain Id, event data, RDR data, and RPT data, formatted as identified via NCI

Event Type

The event type parameter in HPI event notifications is derived from the HPI event type in the received event. HPI event types are mapped to notification event types, so that various subsets of HPI event notifications may be identified for filtering or suppression. HPI event types are mapped to notification event types as defined in Table 10.

Table 10 Mapping HPI Event Type to Notification Event Type

HPI Event Type	Notification Event Type
SAHPI_ET_RESOURCE SAHPI_ET_HOTSWAP	SA_NTF_HPI_EVENT_RESOURCE
SAHPI_ET_SENSOR SAHPI_ET_SENSOR_ENABLE_CHANGE	SA_NTF_HPI_EVENT_SENSOR
SAHPI_ET_WATCHDOG	SA_NTF_HPI_EVENT_WATCHDOG
SAHPI_ET_DIMI	SA_NTF_HPI_EVENT_DIMI
SAHPI_ET_FUMI	SA_NTF_HPI_EVENT_FUMI
All Other HPI Event Types	SA_NTF_HPI_EVENT_OTHER

Notification Object

The notification object parameter in HPI event notifications identifies the HPI implementation that generated the event. That is, the LDAP DN will be:

```
"safApp=safHpiService[:<varAppName>]"
```

The optional ":<varAppName>" string can be used to distinguish between multiple instances of HPI implementations.

Notification Class Identifier

The notification class identifier parameter in HPI event notifications defines the format of the HPI event and related data included in the additional information field. Three formats are defined, any of these may be used. These formats are identified by use of the appropriate notification class identifier defined below. Other formats may be defined by specific implementations. When an implementation-specific format is used, the notification class identifier should include an appropriate `vendorId` for the implementation.

Additional Information

The additional information parameter in HPI event notifications is used to hold the data received from HPI for the event.
Two mandatory data fields and two optional data fields may be included as additional information.

One mandatory additional information field holds the domain Id of the domain from which the event was received. This is the domain with which the HPI session that retrieved the event was opened. If the session was opened using `SAHPI_UNSPECIFIED_DOMAIN_ID`, the actual domain Id of the domain accessed by the session must be retrieved by calling `saHpiDomainInfoGet()`. For more details, see [3]. The domain Id additional information field is defined with an `SaNtfAdditionalInfoT` structure with:

```
infoType   = SA_NTF_VALUE_UINT32
infoId     = SA_PLM_AI_HPI_DOMAIN_ID
```

The other mandatory additional information field holds the data returned in the `SaHpiEventT` structure passed to the `saHpiEventGet()` function, and the two optional additional information fields hold the data returned in the optional `SaHpiRdrT` and `SaHpiRptEntryT` structures passed to the `saHpiEventGet()` function. Each of these fields is defined with an `SaNtfAddtionalInfoT` structure with:

```
infoType        = SA_NTF_VALUE_BINARY
infoId          = SA_PLM_AI_HPI_EVENT_DATA,
                  SA_PLM_AI_HPI_RDR_DATA, or
                  SA_PLM_AI_HPI_RPT_DATA
```

The format of the binary data block in each of these three fields is defined by the notification class identifier. Three preferred formats are defined by this specification. An implementation-specific format may be used with an appropriate notification class identifier defined by the implementation.

It is expected that the data structures returned by HPI in most implementations will follow one of the first two formats described below. If this is the case, the implementation may copy the HPI data structure directly into the additional data field and use the appropriate notification class identifier. If the data in the HPI structures does not follow one of these formats, the implementation may define an implementation-specific notification class identifier to describe the format for the data, or it may translate the data into one of the three standard formats.

The standard formats are:

Normal Binary, MSB First
Notification Class Identifier:

```
vendorId   = SA_NTF_VENDOR_ID_SAF
majorId    = SA_SVC_PLM
minorId    = SA_PLM_NTFID_HPI_NORMAL_MSB
```

Data is binary, and the fields are ordered as listed in the HPI structure definition in SaHpi.h. Each field takes the number of bytes recommended in that header, preceded by pad bytes as required to achieve the recommended alignment. That is,

- `SaHpiUint8T` and `SaHpiInt8T` types use 1 byte and have 1-byte alignment.
- `SaHpiUint16T` and `SaHpiInt16T` types use 2 bytes and have 2-byte alignment.
- `SaHpiUint32T` and `SaHpiInt32T` types use 4 bytes and have 4-byte alignment.
- `SaHpiUint64T` and `SaHpiInt64T` types use 8 bytes and have 8-byte alignment.
- `SaHpiFloat64T` types use 8 bytes and have 8-byte alignment.
- enum types use 4 bytes and have 4-byte alignment.

Floating point fields are represented as IEEE 754 double-precision values with high-order sign bit, followed by 11 bits of exponent, and 52 bit mantissa.

All multi-byte fields are stored most-significant-byte first (that is, big-endian).

Structures and unions are preceded by pad bytes to achieve the same alignment required for their most restrictive (that is, largest) element. Each structure or union is followed by pad bytes as required such that the total length of the structure or union is a multiple of its required alignment.

Normal Binary, LSB First
Notification Class Identifier:

```
vendorId   = SA_NTF_VENDOR_ID_SAF
majorId    = SA_SVC_PLM
minorId    = SA_PLM_NTFID_HPI_NORMAL_LSB
```

Data format is the same as in the previous description, except all multi-byte fields are stored least-significant-byte first (that is, little-endian).

XDR
Notification Class Identifier:

```
vendorId = SA_NTF_VENDOR_ID_SAF
majorId = SA_SVC_PLM
minorId = SA_PLM_NTFID_HPI_XDR
```

Data format is based on XDR (see [14]).

Appendix A Mapping of PLM State Model to CCITT X.731

If a system needs to provide state management as defined in CCITT Recommendation X.731 (see [11]) to the outside world, the PLM state model can be mapped as follows:

⇒ **Administrative state**
The values locked, unlocked, and shutting-down map directly on X.731.
Locked-inactive of hardware elements should be shown as locked plus availability status power-off.
Locked-instantiation of execution environments in X.731 should be shown as locked plus the availability status being not-installed.

⇒ **Readiness state** can be mapped to the X.731 operational state:

- In-service => enabled
- Out-of-service => disabled
- There is a semantic difference between out-of-service and disabled: An administrative LOCK operation affects the readiness state, but not the X.731 operational state. If an object in PLM is locked/out-of-service/enabled, and the readiness flag dependency is not set, the X.731 should show this as locked/enabled. If the dependency flag is set, or the PLM operational state is disabled, X.731 should show locked/disabled.
- Stopping cannot be directly mapped to X.731; it could be shown through the procedural status terminating.

⇒ **Operational state** of the SA Forum maps on a flag in X.731 availability status:

- Disabled => Failed is set
- Enabled => Failed is not set

⇒ **Presence State** for HEs should be shown in X.731 availability status, procedural status, and lifecycle status (see [12]):

- not-present => life cycle status planned and availability status not-installed
- inactive => life cycle status installed, availability status power-off, and procedural status not-initialized
- activating => life cycle status installed and procedural status initializing
- active => life cycle status installed and procedural status empty
- deactivating => life cycle status installed and procedural status terminating

⇒ **Presence State** for EEs maps similarly on flags in X.731 availability status or procedural status and lifecycle status:

- instantiated => life cycle status installed
- uninstantiated => life cycle status planned and availability status not-installed
- terminating => procedural status terminating
- instantiating => procedural status initializing
- instantiation-failed => procedural status initializing and availability status failed
- termination-failed => procedural status terminating and availability status failed

⇒ **Readiness Flags**

- management-lost => unknown status attribute
- dependency => availability status dependency
- admin-operation-pending, isolate-pending, imminent-failure and dependency-imminent-failure cannot be shown in X.731

Appendix B Basic Operational Scenarios

The following basic operational scenarios explain in principle the necessary actions. These basic operational scenarios do not mandate a certain sequence of actions or specific function calls. They are only examples of how an implementation could behave, and they intend only to explain the main principles.

B.1 Extraction of a Computing Blade

There are many different use cases for hardware extraction, depending on the type of hardware and the way it is extracted. The scenarios presented here show the typical extraction of a computing blade. They illustrate the interworking of the PLM Service with HPI and other AIS Services. The provided sequences are just examples of how an implementation could behave. For simplification, no other dependencies are assumed.

The first scenario applies when the blade supports the managed hot swap model. In the second scenario, the hardware supports the unmanaged hot swap model. In the latter case, the hardware does not allow graceful termination of all services on the blade, since it cannot remain in the extraction-pending state.

B.1.1 Extraction of a Computing Blade with Managed Hot Swap

As typical for bladed architectures, extraction is done in two steps: the operator opens some latches, HPI recognizes this extraction request and sends events. The PLM Service has subscribed with HPI and starts all necessary actions for a graceful deactivation. When the deactivation is completed, LEDs are switched to indicate a successful deactivation to the operator. When the operator now physically removes the blade from the chassis, again appropriate actions are triggered in the system.

The blade in this example supports the managed hot swap model and is modeled as an HE. It runs one single operating system, modeled as an EE, on which a CLM node is running. Some applications controlled by the Availability Management Framework run on the AMF node that is mapped to this CLM node. In the beginning of the sequence, all entities are in-service; no entities are administratively locked.

The sequence of actions in this scenario is described next. For the reader's convenience, the steps shown in Section 3.1.3.1.1 in the description of the deactivating presence state (graceful case, on page 30) are reproduced here in a shortened way. To each of these steps (represented by numbers enclosed in curly brackets), the corresponding actions in the scenario (represented by numbers enclosed in parentheses) are given.

{1} PLM sets the presence state of the HE to deactivating.
See steps (1) through (7).

{2} PLM checks the readiness state of the HE being deactivated and all dependent PLM entities.
See step (8).

{3a} If the deactivation policy is `SA_PLM_DP_REJECT_NOT_OOS`, the deactivation is rejected.
See step (9).

{3b} If the deactivation policy is `SA_PLM_DP_VALIDATE`, track callbacks are invoked for the `SA_PLM_CHANGE_VALIDATE` step.
See steps (9) through (15).
A sequence diagram for these first steps is in FIGURE 11.

{3c} If the deactivation policy is `SA_PLM_DP_UNCONDITIONAL`, or after all processes receiving VALIDATE callbacks accept the deactivation, track callbacks are invoked for the `SA_PLM_CHANGE_START` step.
See steps (16) through (27) and FIGURE 12.

{4} PLM initiates the necessary actions using HPI interfaces to actually deactivate the hardware.
See step (28).

{5} When the deactivation is complete, PLM changes the presence state to inactive and the readiness state to out-of-service, and track callbacks are invoked for the `SA_PLM_CHANGE_COMPLETED` step.
See steps (29) through (33).

A sequence diagram for these steps is in FIGURE 13.

Additionally to the deactivation steps, the sequence shows:

{6} The operator actually extracts the blade
See steps (34) through (42) and FIGURE 14

(1) Operator opens latches.

(2) Detection by HPI, PLM receives HPI hot swap-event.
To do this, PLM must have an open session to the domain, managing the entity, and be subscribed for the events.
The HPI event for this case is:

```
EventType            = SAHPI_ET_HOTSWAP
HotSwapState         = SAHPI_HS_STATE_EXTRACTION_PENDING
PreviousHotSwapState= SAHPI_HS_STATE_ACTIVE
CauseOfStateChange   = SAHPI_HS_CAUSE_OPERATOR_INIT
```

(3) PLM generates a notification for the HPI event. The notification id is used as root correlation id in all further notifications.

(4) PLM maps HPI event to HE in the IMMS object model.
HPI entity path is included in HE attributes, search is possible by following the containment.

(5) PLM calls `saHpiHotSwapPolicyCancel()` to stop HPI from taking automatic actions.
Note that PLM will not call this function if the event does not map to a PLM managed object. In such a case, HPI would act according to its policies.

(6) PLM changes the presence state of the HE of the blade to `SA_PLM_HE_PRESENCE_DEACTIVATING`.

(7) PLM generates a state change notification for the blade HE presence state changing to `SA_PLM_HE_PRESENCE_DEACTIVATING`.

(8) PLM checks containment of the affected HE and dependencies. PLM generates a list of all PLM objects (HEs and EEs) that need to terminate their services.
In this example, only the blade HE and the EE are affected.

(9) PLM checks the configured deactivation policy.

- If this attribute is `SA_PLM_DP_REJECT_NOT_OOS`, and there are entities that are not out-of-service, the extraction processing is stopped.
- If this attribute is `SA_PLM_DP_UNCONDITIONAL`, steps (10) to (15) are skipped.

The following sequence shows the case of `SA_PLM_DP_VALIDATE`.

(10) PLM starts to inform subscribed users for the track callbacks:
PLM calls for every subscriber of an affected HE or EE:

```
saPlmReadinessTrackCallback(   my_objGrpHandle,
                               my_trackCookie,
                               my_invocation,
                               SA_PLM_CAUSE_HE_DEACTIVATION,
                               DN_extracted_blade,
                               rootCorrelationId,
                               List_affected_objects,
                               SA_PLM_CHANGE_VALIDATE,
                               return_value);
```

CLM should be subscribed for track callbacks on all EEs on which CLM nodes run. So in this example, CLM is called for the affected EE and will now evaluate whether it is safe to terminate that CLM node.
Other services may be subscribed on EEs or HEs additionally and can take separate validation actions.

(11) In this example, a CLM node in the membership is affected. CLM invokes similarly the `saClmClusterTrackCallback_4()` callbacks of its consumers in the validate step to evaluate the extraction request.
The Availability Management Framework should be subscribed for CLM track callbacks. So the Availability Management Framework will now evaluate whether all active HA assignments can be moved to other nodes, that is, to nodes not affected by the list of affected objects.

(12) The Availability Management Framework checks the redundancy configuration. If it detects that active HA assignments cannot be moved to redundant entities on other nodes, it will reject the extraction by responding to CLM with `SA_CLM_CALLBACK_RESPONSE_REJECTED`.

(13) The Availability Management Framework calls `saClmResponse_4()` and returns `SA_CLM_CALLBACK_RESPONSE_OK` to CLM to indicate that it is safe to extract the entity.

(14) CLM calls `saPlmReadinessTrackResponse()` and returns `SA_PLM_CALLBACK_RESPONSE_OK` to PLM to indicate that it is safe to extract the entity.

(15) After PLM receives a positive response from all track callbacks for all affected objects, PLM decides to allow the extraction.

(16) PLM changes the presence state of the affected EE to `SA_PLM_EE_PRESENCE_TERMINATING`.

(17) PLM now requests all subscribed users to terminate their services, that is, PLM calls for every subscriber of an affected HE or EE:

```
saPlmReadinessTrackCallback(   my_objGrpHandle,
                               my_trackCookie,
                               my_invocation,
                               SA_PLM_CAUSE_HE_DEACTIVATION,
                               DN_extracted_blade,
                               rootCorrelationId,
                               List_affected_objects,
                               SA_PLM_CHANGE_START,
                               return_value);
```

CLM should be subscribed for the EE and will now evict the affected node. Other services that are subscribed can take separate actions.

(18) CLM invokes the `saClmClusterTrackCallback_4()` callbacks of its consumers in the start step, and passes all needed information in the call.

(19) The Availability Management Framework should be subscribed for CLM track callbacks as above. So if an AMF node has running services, and the corresponding CLM node terminates, the Availability Management Framework will now change HA assignments according to its redundancy configuration.

(20) The Availability Management Framework terminates all components and service units on the affected nodes.

(21) The Availability Management Framework calls `saClmResponse_4()` and returns `SA_CLM_CALLBACK_RESPONSE_OK` to CLM to indicate that the node may now leave the membership.

(22) As soon as all track callbacks have returned, CLM removes the affected nodes from the membership.

(23) CLM invokes the `saClmClusterTrackCallback_4()` callbacks of its consumers in the completed step, and passes all needed information in the call.

(24) CLM calls `saPlmReadinessTrackResponse()` and returns `SA_PLM_CALLBACK_RESPONSE_OK` to PLM.

(25) After PLM receives a positive response from all track callbacks, PLM terminates that EE (OS shutdown or similar operation).

(26) PLM changes the presence state of the EE to `SA_PLM_EE_PRESENCE_UNINSTANTIATED` and its readiness state to `SA_PLM_READINESS_OUT_OF_SERVICE` and generates a state change notification using the correlation id.

(27) PLM may informs the subscribers of the track interface that subscribed only for the EE already now with the completed step.
Note that from HPI's perspective, the process is not yet completed; however, these users are only interested in the EE readiness state.
PLM calls for every subscriber of the affected EE:

```
saPlmReadinessTrackCallback(   my_objGrpHandle,
                               my_trackCookie,
                               my_invocation,
                               SA_PLM_CAUSE_HE_DEACTIVATION,
                               DN_extracted_blade,
                               rootCorrelationId,
                               List_affected_objects,
                               SA_PLM_CHANGE_COMPLETED,
                               return_value);
```

(28) PLM allows HPI to execute the hot swap extraction policy by calling `saHpiResourceInactiveSet()` for the blade. This function will initiate all necessary hardware actions to deactivate the entity.

(29) PLM receives HPI events for the completion of the hot swap requests.
The HPI event for this case is:

```
EventType            = SAHPI_ET_HOTSWAP
HotSwapState         = SAHPI_HS_STATE_INACTIVE
PreviousHotSwapState= SAHPI_HS_STATE_EXTRACTION_PENDING
CauseOfStateChange   = SAHPI_HS_CAUSE_EXT_SOFTWARE
```

(30) PLM detects that the event is related to the running extraction processing.

(31) PLM generates a notification for the HPI event using the correlation id.

(32) PLM changes the presence state of the related HE to `SA_PLM_HE_PRESENCE_INACTIVE` and its readiness state to `SA_PLM_READINESS_OUT_OF_SERVICE` and generates a state change notifications using the correlation id.

(33) PLM informs the subscribers of the track interface for the HE with the completed step.
PLM calls for every subscriber of the affected HE (CLM should not be subscribed here):

```
saPlmReadinessTrackCallback(   my_objGrpHandle,
                               my_trackCookie,
                               my_invocation,
                               SA_PLM_CAUSE_HE_DEACTIVATION,
                               DN_extracted_blade,
                               rootCorrelationId,
                               List_affected_objects,
                               SA_PLM_CHANGE_COMPLETED,
                               return_value);
```

(34) The Operator extracts the blade.

(35) HPI notifies about the extraction.
The HPI event for this case is:

```
EventType            = SAHPI_ET_HOTSWAP
HotSwapState         = SAHPI_HS_STATE_NOT_PRESENT
PreviousHotSwapState= SAHPI_HS_STATE_INACTIVE
CauseOfStateChange   = SAHPI_HS_CAUSE_OPERATOR_INIT
```

(36) PLM generates a notification for the HPI event. This event is a new root cause. The new notification id is used in all further calls to allow correlation.

(37) PLM maps the HPI event to HE in the IMMS object model and checks the states.

(38) PLM changes the presence state of the affected HE object to `SA_PLM_HE_PRESENCE_NOT_PRESENT`.

(39) PLM generates a state change notification for the presence state of the HE changing to `SA_PLM_HE_PRESENCE_NOT_PRESENT`.

(40) PLM checks containment and dependencies of the affected HE. All contained HEs are extracted together and should already be out-of-service. All contained and dependent EEs should also already be out-of-service.

(41) PLM changes the presence state of all contained HEs to `SA_PLM_HE_PRESENCE_NOT_PRESENT` and generates state change notification (using the correlation identifier).
Note that no track callbacks are called in this case. All affected entities are already out-of-service. Any software that needs to know about the physical extraction will need to listen to notifications.

(42) PLM clears active alarms of the HE and all contained HEs.
Alarms in the domain alarm table are cleared by HPI itself.
However, PLM has to send a notification with severity "cleared" for hardware-related alarms that it had issued for these HEs.

The next figures show the actions taken when a computing blade is extracted.

FIGURE 11 Extraction of a Computing Blade (Deactivation Part 1)

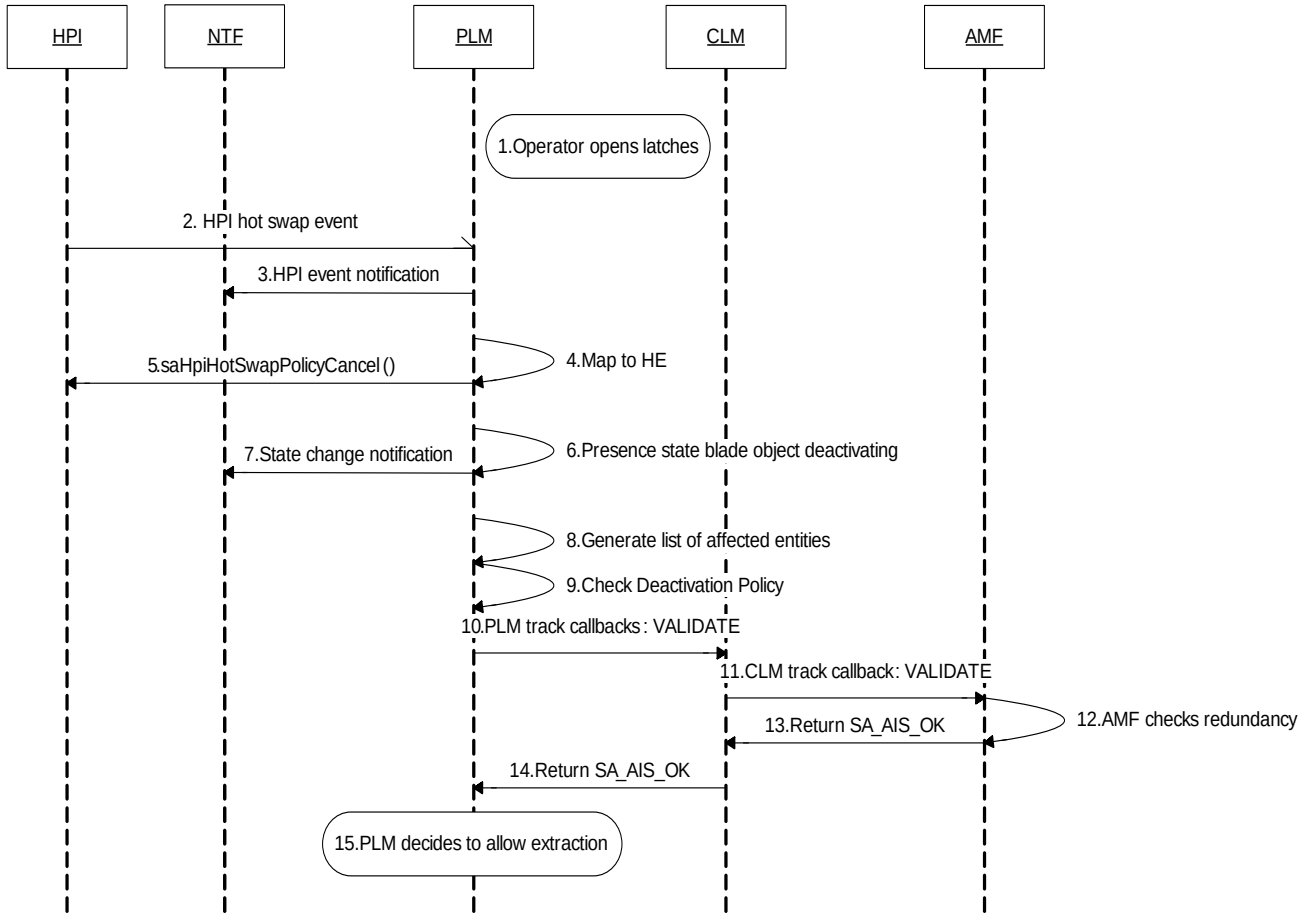


FIGURE 12 Extraction of a Computing Blade (Deactivation Part 2)

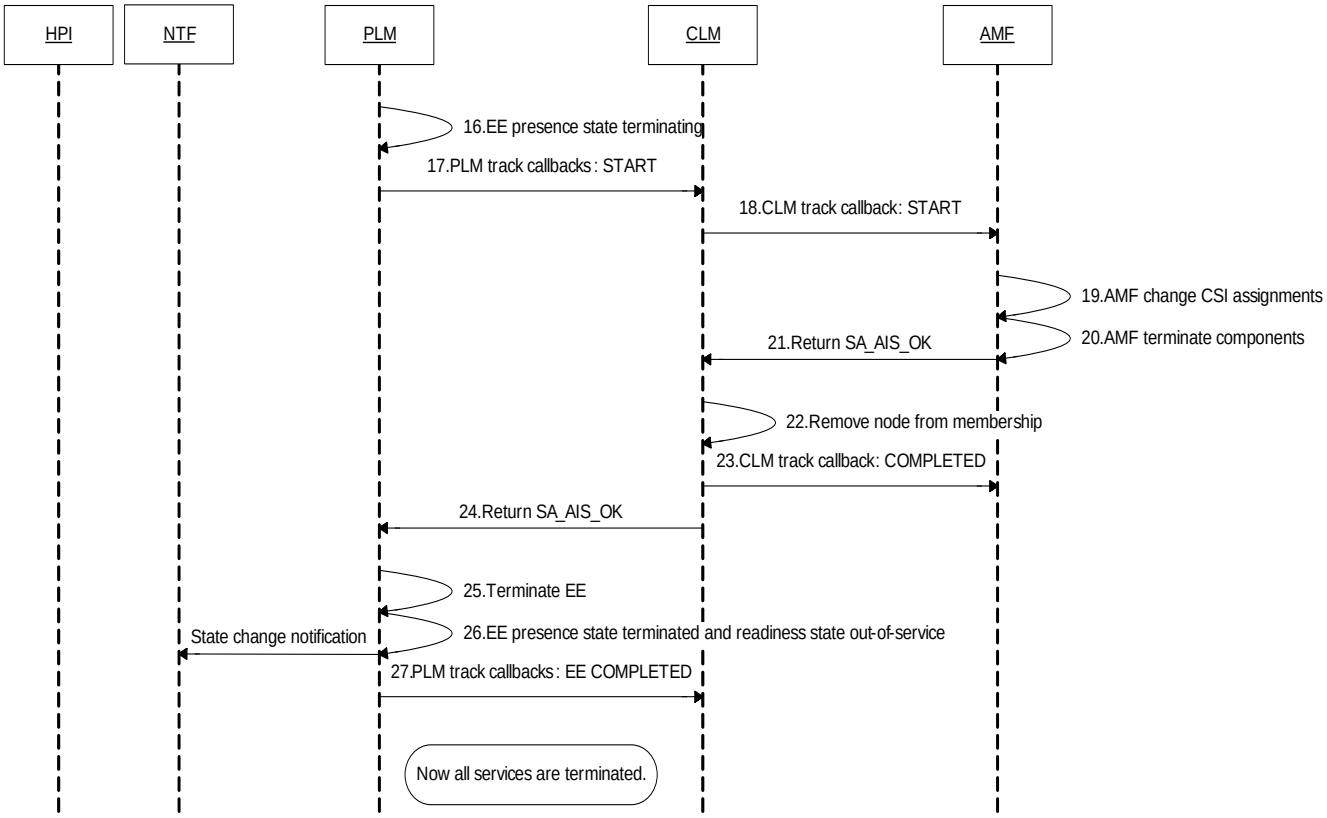


FIGURE 13 Extraction of a Computing Blade (Deactivation Part 3)

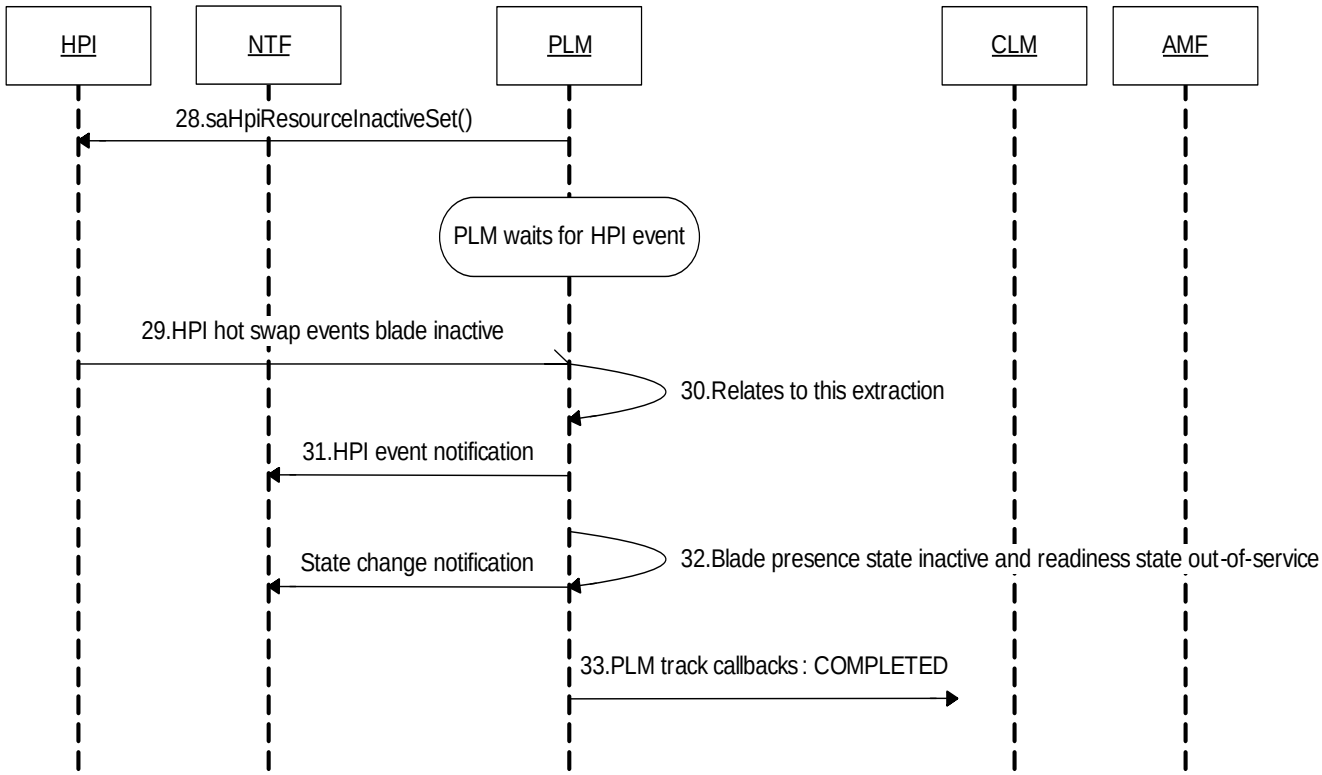
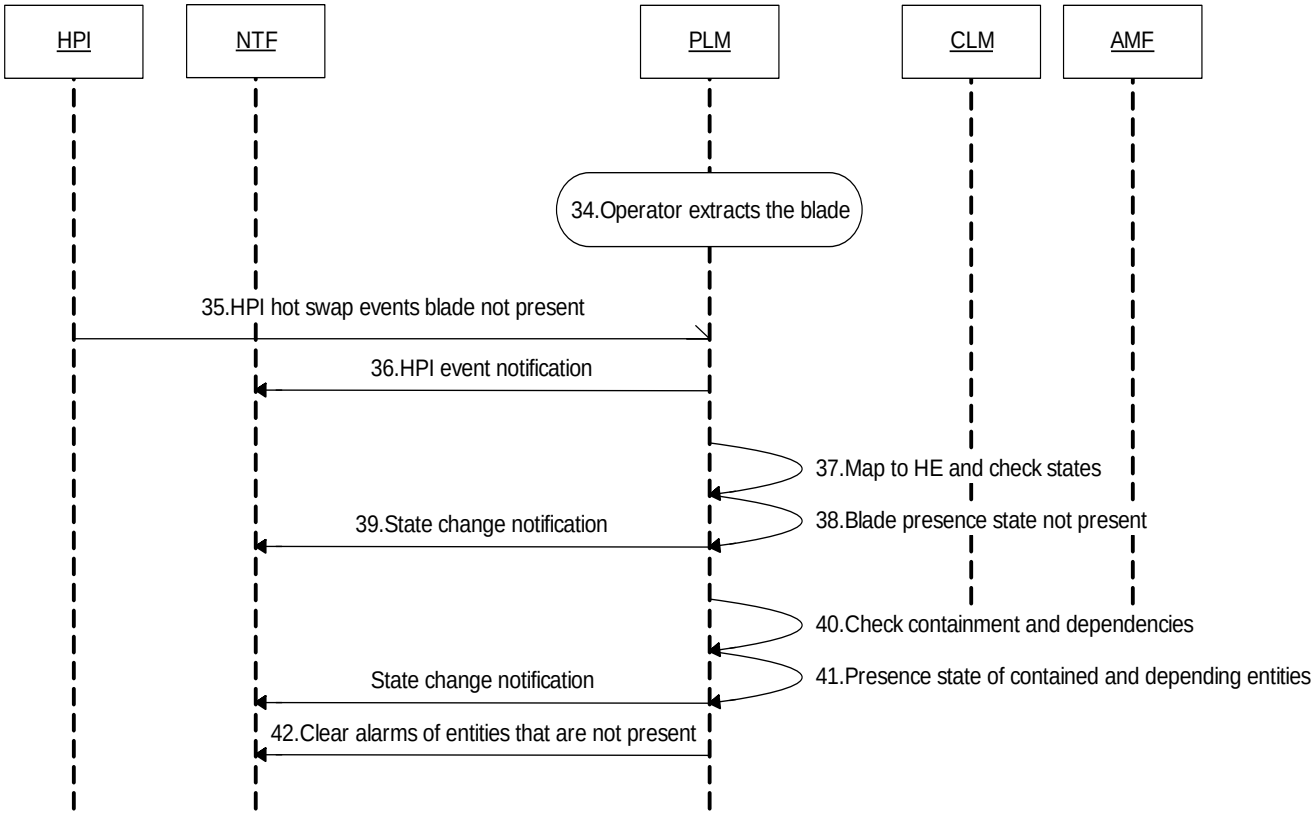


FIGURE 14 Actual Extraction of a Computing Blade



B.1.2 Extraction of a Computing Blade with Unmanaged Hot Swap

In the case of unmanaged hot swap, the hardware cannot stay in the extraction-pending state, that is, the auto-extraction-policy cannot be stopped. The blade in this example is again modeled as an HE that runs a single operating system, which is modeled as an EE, on which in turn a CLM node is running. Some applications controlled by the Availability Management Framework run on the AMF node that is mapped to this CLM node. In the beginning of the sequence, all entities are in-service; no entities are administratively locked.

Note that an operator wanting graceful deactivation can use the administrative LOCK administrative operation to terminate the services gracefully.

The sequence of actions in this scenario is described next. For the reader's convenience, the steps shown in Section 3.1.3.1.1 in the description of the deactivating presence state (abrupt case, on page 31) are reproduced here in a shortened way. To each of these steps (represented by numbers enclosed in curly brackets), the corresponding actions in the scenario (represented by numbers enclosed in parentheses) are given.

{1} PLM sets the presence state to deactivating.
See steps (1) through (6).

{2} PLM monitors the hardware as it deactivates.
Wait for step (7) to happen.

{3} When the deactivation is complete, PLM changes the presence state to inactive and the readiness state to out-of-service, and track callbacks are invoked for the `SA_PLM_CHANGE_COMPLETED` step.
See steps (8) through (16).

FIGURE 15 shows a sequence diagram for these steps.

The actual extraction is done in the same way as in the previous use-case.

(1) Operator opens latches.

(2) Detection by HPI, PLM receives HPI a hot swap event.
To receive this event, PLM must have an open session to the domain managing the entity and be subscribed for the events.
The HPI event for this case is:

```
EventType            = SAHPI_ET_HOTSWAP
HotSwapState         = SAHPI_HS_STATE_EXTRACTION_PENDING
PreviousHotSwapState= SAHPI_HS_STATE_ACTIVE
CauseOfStateChange   = SAHPI_HS_CAUSE_OPERATOR_INIT
```

(3) PLM generates a notification for the HPI event. The notification id is used as root correlation id in all further notifications.

(4) PLM maps the HPI event to an HE in the IMMS object model.
The HPI entity path is included in the HE attributes, search is possible by following the containment.
PLM knows hardware capabilities, so it will not call `saHpiHotSwapPolicyCancel()`. HPI will continue to deactivate the hardware.

(5) PLM changes the presence state of the HE of the blade to `SA_PLM_HE_PRESENCE_DEACTIVATING`.

(6) PLM generates a state change notification for the blade HE presence state changing to `SA_PLM_HE_PRESENCE_DEACTIVATING`.

(7) PLM receives an HPI event for the completion of the deactivation.
The HPI event for this case is:

```
EventType            = SAHPI_ET_HOTSWAP
HotSwapState         = SAHPI_HS_STATE_INACTIVE
PreviousHotSwapState= SAHPI_HS_STATE_EXTRACTION_PENDING
CauseOfStateChange   = SAHPI_HS_CAUSE_EXT_SOFTWARE
```

(8) PLM detects that the event is related to the running extraction processing.

(9) PLM generates a notification for the HPI event using the correlation id.

(10) PLM changes the presence state of the related HE to `SA_PLM_HE_PRESENCE_INACTIVE` and its readiness state to `SA_PLM_READINESS_OUT_OF_SERVICE` and generates a state change notifications using the correlation id.

(11) PLM checks the containment of the affected HE and dependencies. PLM generates a list of all PLM objects (HEs and EEs) that need to be informed about the abrupt deactivation.
In this example, only the blade HE and the EE are affected.

(12) PLM changes the presence state of the EE to `SA_PLM_EE_PRESENCE_UNINSTANTIATED` and its readiness state to `SA_PLM_READINESS_OUT_OF_SERVICE` and generates a state change notification using the correlation id.

(13) PLM informs subscribers of the track interface with the completed step. PLM calls for every subscriber of the affected HE and EE:

```
saPlmReadinessTrackCallback(   my_objGrpHandle,
                               my_trackCookie,
                               my_invocation,
                               SA_PLM_CAUSE_HE_DEACTIVATION,
                               DN_extracted_blade,
                               rootCorrelationId,
                               List_affected_objects,
                               SA_PLM_CHANGE_COMPLETED,
                               return_value);
```

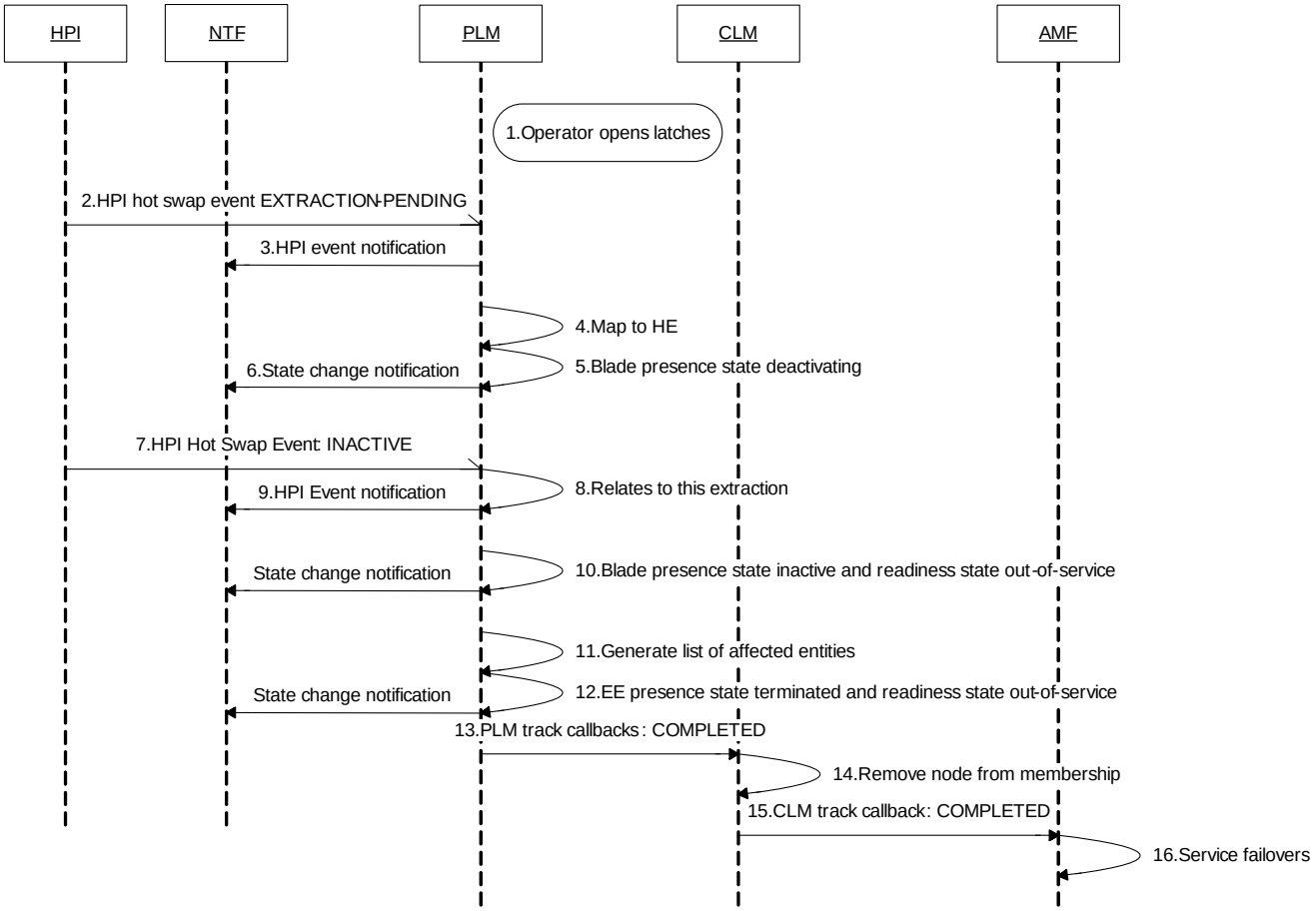
(14) CLM should be subscribed for the EE and will remove the affected node from membership.
Note that CLM may already have detected that the node has unexpectedly left the cluster, so some of the following steps may have already happened.
Other services that are subscribed can take separate actions.

(15) CLM invokes the `saClmClusterTrackCallback_4()` callbacks of its clients in the completed step and passes all needed information in the call.

(16) The Availability Management Framework should be subscribed for CLM track callbacks as above. So if an AMF node has running services, these services need to failover according to its redundancy configuration.

Some hardware may not support the extraction-pending step at all. In this case, steps (2) to (6) do not apply, and the first HPI event reports a direct state change from active to inactive.

FIGURE 15 Extraction of a Computing Blade Supporting Unmanaged Hot Swap



B.2 Fault of a Computing Blade

This use-case shows the processing of the PLM Service when a fault of a computing blade is detected. The PLM Service can detect the fault by analyzing HPI events or if an application reports the error by invoking the `saPlmEntityReadinessImpact()` function.

The blade in this example is again modeled as an HE; the HE runs one single operating system, which is modeled as an EE on which a CLM node is running. Some applications controlled by the Availability Management Framework run on the AMF node that is mapped to this CLM node. In the beginning of the sequence, all entities are in-service, and no entities are administratively locked. It is assumed that no other entities are affected.

The sequence of actions is described next. A sequence diagram is in FIGURE 16.

(1) An application reports the fault:

```
saPlmEntityReadinessImpact(    plmHandle,
                               DN_faulty_blade,
                               SA_PLM_RI_FAILURE,
                               correlationIds);
```

(2) PLM checks containment of the affected HE and dependencies. PLM generates a list of all PLM objects (HEs and EEs) that are affected. It may be necessary to isolate multiple entities.
In this example, only the blade HE and the EE are affected. The EE does not need a separate isolation, it terminates automatically when the HE is isolated.

(3) PLM uses HPI to deactivate the blade by calling `saHpiResourceInactiveSet()` for the blade. This function will initiate all necessary hardware actions to deactivate the entity.

(4) PLM receives HPI events for the completion of the hot swap requests.
The HPI event for this case is:

```
EventType = SAHPI_ET_HOTSWAP
HotSwapState = SAHPI_HS_STATE_INACTIVE
PreviousHotSwapState= SAHPI_HS_STATE_ACTIVE
CauseOfStateChange = SAHPI_HS_CAUSE_EXT_SOFTWARE
```

(5) PLM detects that the event is related to the isolation of the faulty HE.

(6) PLM generates a notification for the HPI event using the correlation ids.

(7) PLM changes the presence state of the HE of the blade to SA_PLM_HE_PRESENCE_INACTIVE, its operational state to SA_PLM_OPERATIONAL_DISABLED, its readiness state to SA_PLM_READINESS_OUT_OF_SERVICE, and generates a state change notification using the correlation id.

(8) PLM changes the presence state of the affected EE to SA_PLM_EE_PRESENCE_UNINSTANTIATED, sets the readiness flag SA_PLM_RF_DEPENDENCY for this EE, sets the readiness state of the EE to SA_PLM_READINESS_OUT_OF_SERVICE, and generates a state change notification using the correlation id.

(9) PLM informs the subscribers of the track interface for the HE and EE for the completed step.
PLM calls for every subscriber of HE or EE:

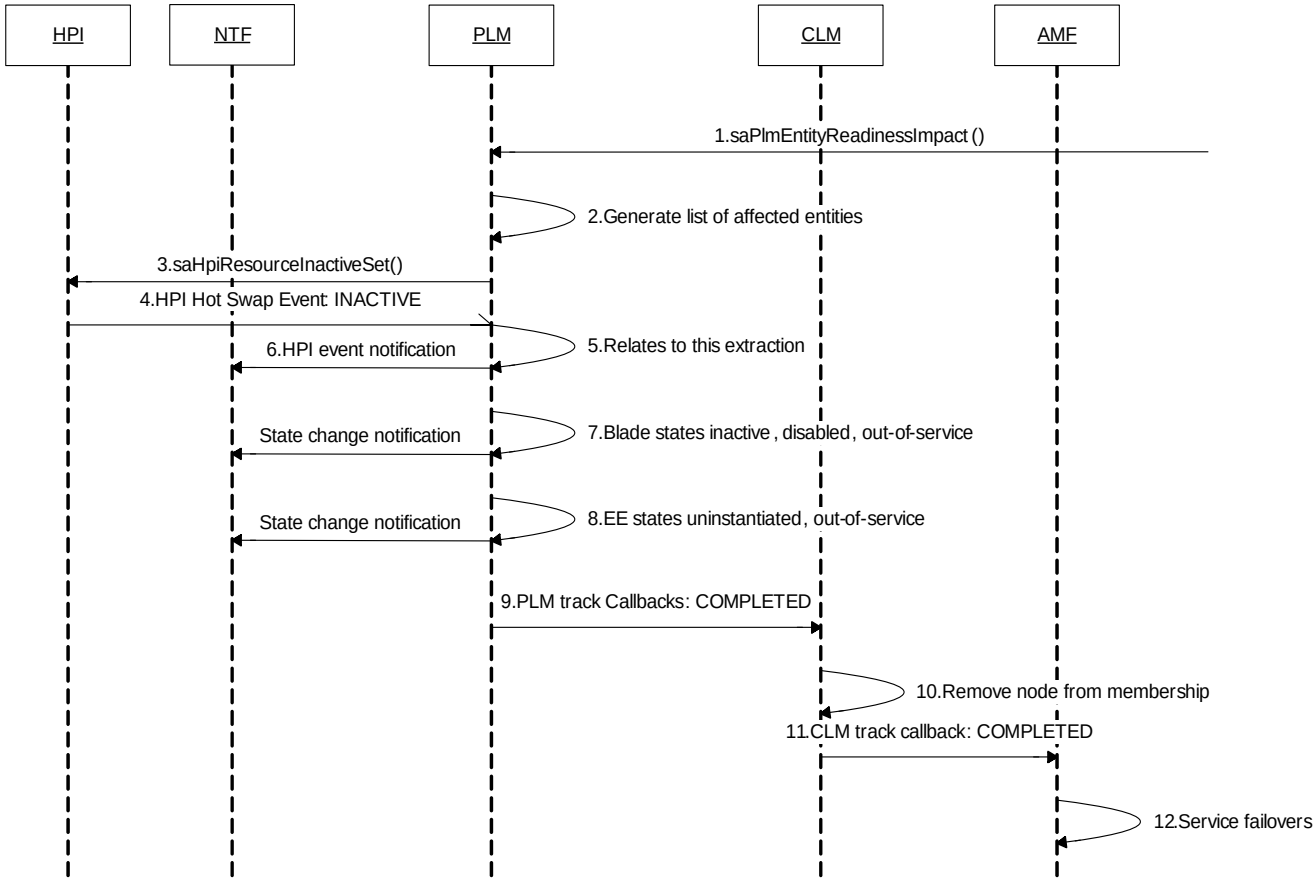
```
saPlmReadinessTrackCallback(   my_objGrpHandle,
                               my_trackCookie,
                               my_invocation,
                               SA_PLM_CAUSE_FAILURE,
                               DN_faulty_blade,
                               rootCorrelationId,
                               list_affected_objects,
                               SA_PLM_CHANGE_COMPLETED,
                               return_value);
```

(10) CLM should be subscribed for the EE, and CLM thus removes the affected node from the cluster membership.
Note that CLM may already have detected that the node has unexpectedly left the cluster, so some of the following steps may already have happened. Other services that are subscribed can take separate actions.

(11) CLM invokes for the completed step the saClmClusterTrackCallback_4() callback functions of its clients and passes all needed information in the corresponding invocations.

(12) The Availability Management Framework should be subscribed for CLM track callbacks as above. So if an AMF node had running services, these services need to failover according to the redundancy configuration.

FIGURE 16 Fault of a Computing Blade



Index of Definitions