Availability Service

Programmer's Reference

6806800C44B

September 2007

Trademarks

Motorola and the stylized M logo are trademarks registered in the U.S. Patent and Trademark Office. All other product or service names are the property of their respective owners.

Intel® is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Java™ and all other Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Microsoft®, Windows® and Windows Me® are registered trademarks of Microsoft Corporation; and Windows XP™ is a trademark of Microsoft Corporation.

PICMG®, CompactPCI®, AdvancedTCA™ and the PICMG, CompactPCI and AdvancedTCA logos are registered trademarks of the PCI Industrial Computer Manufacturers Group.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

Notice

While reasonable efforts have been made to assure the accuracy of this document, Motorola assumes no liability resulting from any omissions in this document, or from the use of the information obtained therein. Motorola reserves the right to revise this document and to make changes from time to time in the content hereof without obligation of Motorola to notify any person of such revision or changes.

Electronic versions of this material may be read online, downloaded for personal use, or referenced in another document as a URL to a Motorola website. The text itself may not be published commercially in print or electronic form, edited, translated, or otherwise altered without the permission of Motorola,

It is possible that this publication may contain reference to or information about Motorola products (machines and programs), programming, or services that are not available in your country. Such references or information must not be construed to mean that Motorola intends to announce such Motorola products, programming, or services in your country.

Limited and Restricted Rights Legend

If the documentation contained herein is supplied, directly or indirectly, to the U.S. Government, the following notice shall apply unless otherwise agreed to in writing by Motorola.

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (b)(3) of the Rights in Technical Data clause at DFARS 252.227-7013 (Nov. 1995) and of the Rights in Noncommercial Computer Software and Documentation clause at DFARS 252.227-7014 (Jun. 1995).

Contact Address

Motorola GmbH

ECC Embedded Communications Computing

Lilienthalstr. 15

85579 Neubiberg-Munich/Germany

Contents

List of Tables

List of Figures

About this Manual

Overview of Contents

This manual is divided into the following chapters and appendices.

- Chapter 1: Introduction
  Describes the functionality and main features of the Availability Service
- Chapter 2: Management Interface
  Describes how to configure the functionality of the Availability Service
- Appendix A: Sample Application
  Describes the sample application which illustrates the functionality of the Availability Service. The sample application is delivered with the Avantellis software.
- Appendix B: Related Documentation
  Provides references to further documentation and specifications that are related to NCS and the Availability Service.

Abbreviations

This document uses the following abbreviations:

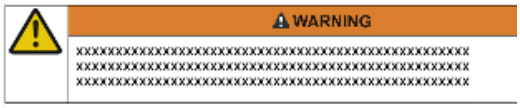
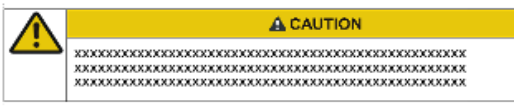
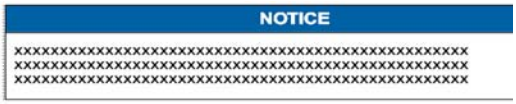
Abbreviation	Definition
AMC	Alarm Management Controller
AMF	Availability Management Framework
API	Application Programming Interface
ARP	Address Resolution Protocol
AvA	Availability Agent
AvD	Availability Director
AvM	Availability Manager
AvSv	Availability Service
BOM	Bill of Material
CLI	Command Line Interface
Cluster Membership Agent	CLA
CSI	Component Service Interface
DTSv	Distributed Tracing Service
Event Distribution Service	EDSv
FRU	Field Replaceable Unit
HISv	Hardware Interface Service

Abbreviation	Definition
HPI	Hardware Platform Interface
LEAP	Layered Enhanced Accelerated Portability
MASv	Management Access Service
MBCSv	Message Based Checkpoint Service
MDS	Message Distribution Service
MIB	Management Information Base
NCS	Netplane Core Services
PCAP	Payload Construction and Availability Service
SAF	Service Availability Forum
SCAP	System Construction and Availability Process
SG	Service Group
SI	Service Instance
SNMP	System Network Management Protocol
SRMSv	System Resource Monitoring Service
SU	Service Unit
XML	Extended Markup Language

Conventions

The following table describes the conventions used throughout this manual.

Notation	Description
0x00000000	Typical notation for hexadecimal numbers (digits are 0 through F), for example used for addresses and offsets
0b0000	Same for binary numbers (digits are 0 and 1)
bold	Used to emphasize a word
`Screen`	Used for on-screen output and code related elements or commands in body text
`Courier + Bold`	Used to characterize user input and to separate it from system output
Reference	Used for references and for table and figure descriptions
File > Exit	Notation for selecting a submenu
<text>	Notation for variables and keys
[text]	Notation for software buttons to click on the screen and parameter description
...	Repeated item for example node 1, node 2, ..., node 12

Notation	Description
. . .	Omission of information from example/command that is necessary at the time being
..	Ranges, for example: 0..4 means one of the integers 0,1,2,3, and 4 (used in registers)
|	Logical OR
	Indicates a hazardous situation which, if not avoided, could result in death or serious injury
	Indicates a hazardous situation which, if not avoided, may result in minor or moderate injury
	Indicates a property damage message
	No danger encountered. Pay attention to important information

Summary of Changes

This manual has been revised and replaces all prior editions.

Part Number	Publication Date	Description
6806800A44A	February 2007	First edition
6806800A44B	September 2007	Minor text updates for Avantellis Release 3.0.2

Comments and Suggestions

We welcome and appreciate your comments on our documentation. We want to know what you think about our manuals and how we can make them better.

Mail comments to:

- Motorola GmbH
  Embedded Communications Computing
  Lilienthalstrasse 15
  85579 Neubiberg

Germany

- eccrc@motorola.com

In all your correspondence, please list your name, position, and company. Be sure to include the title, part number, and revision of the manual and tell how you used it.

Introduction

1

1.1 Overview

The NCS Availability Service (AvSv) is the core service of the NetPlane software. It provides service availability to applications by coordinating the redundant resources in a cluster to provide a system with no single point of failure. It provides high-availability mechanisms to the application software it manages. These include life-cycle management of application software, fault detection, fault isolation, escalation, recovery, and repair.

The AvSv functionality is a highly compliant implementation of Service Availability Forum's Application Interface Specification of Availability Management Framework (SAI-AIS-AMF-B.01.01) and Cluster Membership Service (SAI-AIS-CLM-B.01.01).

The Availability Service (AvSv) provides the following functionality:

- Leverage the SAF "System Description and Conceptual Model"
- Honour the Availability Management Framework" API
- Honour the SA Cluster membership Service API
- House the MIB tables corresponding to the hardware portion of the deployment system description which includes entity containment and fault domain hierarchy information
- House the MIB tables corresponding to the software portion of the deployment system description which include configuration of AMF-defined logical entities and their relationship
- Perform blade validation on receipt of HPI hot swap insertion events
- Handle fault events such as HPI hot swap extraction events, threshold crossing events etc.

The AvSv maintains a software system model database which captures SAF-described logical entities and their relationships to each other. The software system model database is initially configured from data contained in the System Description file. Through time the system model will modify due to changing system realities and administrative actions.

The SAF logical entities related in the system model include components which normalize the view of physical resources such as processes, drivers or devices. Components are grouped into Service Units according to fault dependencies that exist among them. A Service Unit is also scoped to one or more (physical) fault domains. Service Units of the same type are grouped into Service Groups (SG) which exhibit particular redundancy modelling characteristics. Service Units within a SG are assigned to Service Instances (SI) and given a High Availability state of active and standby.

The hardware database maintained by AvSv includes hardware entity containment information and the hardware fault domain hierarchy. All hardware entities are represented by their HPI entity paths. The hardware entity containment tree only includes managed FRUs which may or may not include processor environments., and non-FRU resources which include processor environments. The fault domain data includes dependency relationships between parent-child

entities as well as non-parent child entities. The hardware system model also includes validation data for managed FRUs and the linkages between entities and AMF logical nodes. All the processor environment entities in the hardware entity containment tree, which correspond to AMF nodes, contain the node name of the associated node. The node name provides the linkage between the hardware and the software system models.

Further functionality provided by AvSv includes:

- Automatic and administrative means to instantiate, terminate and restart resources
- Automatic and administrative means to manage or reflect Service Group, Service Unit, Service Instance and Resource state
- Administrative means to perform switch-over
- Administrative means to reset (but not power cycle) nodes
- Heartbeat and event subscription schemes for fault detection, isolation and identification
- Health-check services to probe and prevent system trauma that lead to faults
- Fault recovery mechanisms to fail-over SIs which maintain service availability in case of system trauma
- Fault repair mechanisms to restore failed components
- Validation of hardware resources (managed FRUs) entering the system

The AsVs itself cannot be a single-point of failure. It provides its own internal scheme and mechanisms to protect itself from its own failure.

1.2 Models and Concepts

This chapter provides information on:

- Service Structure and architecture
- Compliancy to SAF standard
- Service Dependencies
- References to SAF documents which provide details about the service functionality
- Service Extensions
- Implementation Notes
- Configuration

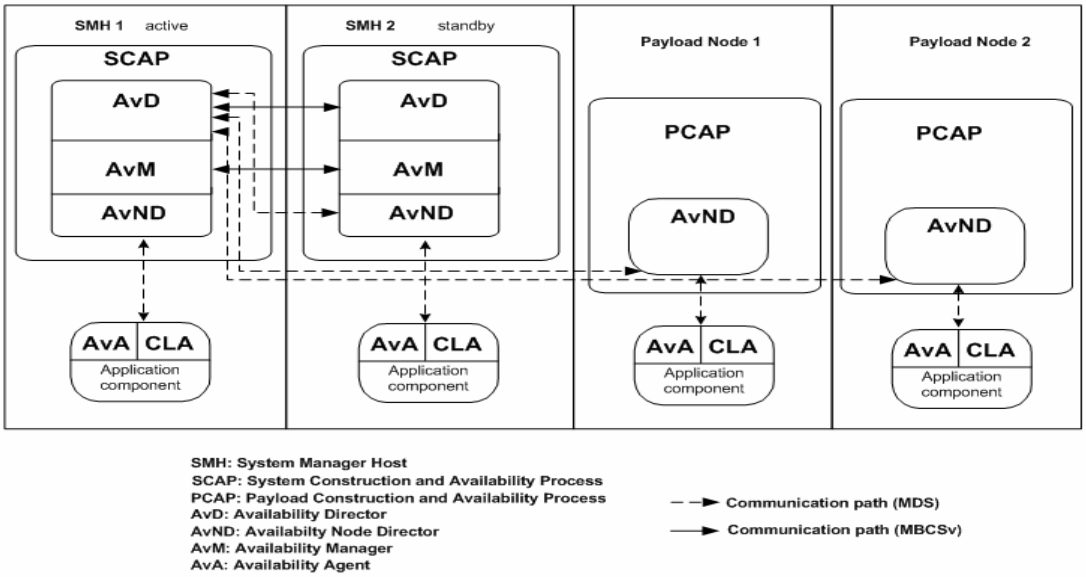
1.2.1 Service Structure Overview

Availability Service is made up of the following distributed sub-parts:

- Availability Manager
- Availability Director
- Availability Node Director

- Availability Agent
- Cluster Membership Agent

Figure 1-1 Availability Service - Subparts



1.2.1.1 Availability Manager

NetPlane Core Services' Availability Manager (AvM) maintains the hardware model of the system above. It acts as a bridge between the Availability Management Framework (AMF) and the Hardware Platform Interface (HPI). It supports activation and deactivation of field-replaceable units (FRUs), Reset Management, Lock Management, and Fault Management. AvM interacts with internal role distribution and fault management mechanisms to capture the role of system manager hosts and propagate it to the AMF. It is also used to trigger administrative switchovers of system manager hosts. AvM resides on both the active and standby system manager hosts.

1.2.1.2 Availability Director

The Availability Director (AvD) maintains the entire system model, consisting of nodes, the Service Groups (SG), their constituent Service Units (SUs), their constituent components, and their corresponding component service instance (CSI) and service instances (SIs) that are in the system. There is an active and a standby instance of the AvD in a system. The AvD runs as part of a System Construction and Availability Process (SCAP) on the system manager host.

Its main tasks include fault detection, isolation and recovery procedures as defined in the SAF AMF. Any problems and failures on a component that cannot be handled locally, are prompted to the Availability Director which controls and triggers the isolation of the affected component and, if possible, the activation of a stand-by component.

1.2.1.3 Availability Node Director

The Availability Node Director (AvND) resides on each system node and its main task is to maintain the node-scoped part of the software system model described above.

The AvND coordinates local fault identification and repair of components and furthermore facilitates any wishes it receives from the Availability Director.

The AvND watches for components arriving or leaving the system and summarizes this information in a Service Unit (SU) presence state, and keeps the AvD informed about the current status and changes. The AvND is capable of disengaging, restarting and destroying any component within its scope. This may occur according to AvD instructions or as a result of an administrative action or automatically triggered by policies.

1.2.1.4 Availability Agent

The Availability Agent (AvA) is the linkable library that provides a means for the AvSv to exchange information with system components overseen by the process in which this library is planted. It does not run as a separate thread.

The AvA implements the SAF Availability Management Framework API and provides the entry-point for accessing AMF functionality.

1.2.1.5 Cluster Membership Agent

The Cluster Membership Agent (CLA) is a linkable library that enables AvSv to provide information about nodes in the cluster to the process in which it is linked. It does not run as a separate thread.

The CLA implements the SAF Cluster Membership Service Library functionality and provides an entry point to the SAF CLM functionality.

1.2.2 Compliance Report

Availability Service conforms to the Application Interface specifications mentioned in the following SAF documents:

- SAI-AIS Volume 1: Overview and Models (SAI-AIS-B.01.01)
- SAI-AIS Volume 2: Availability Management Framework (SAI-AIS-AMF-B.01.01)
- SAI-AIS Volume 3: Cluster Membership Service (SAI-AIS-CLM-B.01.01)

Table 1-1 Compliance Table - Availability Service, SAI-AIS Volume 1: Overview and Models

Section	Description	Supported	Notes
1	Document Introduction	NA	Informational
2	Overview of the AIS	NA	Informational
3	Programming Model and Naming Conventions	Yes	

Table 1-1 Compliance Table - Availability Service, SAI-AIS Volume 1: Overview and Models

Section	Description	Supported	Notes
4	System Description and Conceptual Model	Yes	
4.1	Physical Entities	Yes	AvSv supports management of hardware resources only when they are modeled using proxy components
4.2	Logical Entities	Yes	
5	AIS Abbreviations, Concepts, and Terminology	NA	Informational

Table 1-2 Compliance Table - Availability Service, SAI-AIS Volume 2: Availability Management Framework

Section	Description	Support	Notes
1	Document Introduction	NA	Informational
2	Overview	NA	Informational
3	System Description and System Model	Yes	
3.1	Logical Entities	Yes	Constituent sub-sections that are only partially supported or not supported are mentioned below. Note that the rest of the sub-sections are fully supported.
3.1.2	Components	Yes	AvSv currently does not support external components
3.1.2.3	Proxy and Proxied Components	Yes	AvSv currently supports only the model in which proxied and their proxy components are on the the same Node.
3.1.4	Service Units	Yes	AvSv currently does not support External Service Units.
3.2	State Models	Yes	
3.3	Fail-over and Switch-over	Yes	AvSv supports switchover of service instances caused due to administrative operations specified in SA-AIS-AMF-B.01.01 and some proprietary switchover mechanisms not specified in the document.
3.4	Administrative Operations	Yes	
3.5	Possible Combination of States for Service Units	Yes	

Table 1-2 Compliance Table - Availability Service, SAI-AIS Volume 2: Availability Management Framework (continued)

Section	Description	Support	Notes
3.6	Component Capability Model	Yes	
3.7	Service Group Redundancy Model	Yes	Constituent sub-sections that are only partially supported or not supported are mentioned below. Note that the rest of the sub-sections are fully supported.
3.7.1	Common Characteristics	Yes	AvSv currently does not support the failback option.
3.7.2	2N Redundancy Model	Yes	
3.7.3	N+M Redundancy Model	Yes	SIs based on `saAmfSIRank` are assigned to SUs based on `saAmfSURank` till each SU is assigned `saAmfSGMaxActiveSIsperSU` number of SIs. If there are SIs still left without assignments than the SUs will be assigned with SIs up to their capability level. The standby assignments of all the SIs assigned to an active SU will be assigned to the same standby SU. The `saAmfSGMaxActiveSIsperSU` can be leveraged by the configurator to get the required spread of SI assignments to the SUs.

Table 1-2 Compliance Table - Availability Service, SAI-AIS Volume 2: Availability Management Framework (continued)

Section	Description	Support	Notes
3.7.4	N-Way Redundancy Model	Yes	SIs based on `saAmfSIRank` are assigned to SU based on `saAmfSUsperSIRankTable` till all the SIs are assigned an active assignment. Then each SI based on `saAmfSIRank` is assigned fully to SUs as standby before the next SI is assigned as standby. When an SU fails and the system is in a degraded state, AvSv may ignore the `saAmfCompNumMaxActiveCsi` attribute of a component and assign more active assignments than the permitted number. This policy has been adopted to maintain service continuity for as many SIs as possible even when the system is in a degraded state.
3.7.5	N-Way Active Redundancy Model	Yes	AvSv assigns an SI to SU only after the SI with better `saAmfSIRank` is fully assigned.
3.7.6	No Redundancy Model	Yes	
3.7.7	The Effect of Administrative Operations on Service Instance Assignments	Yes	
3.8	Component Capability Model and Service Group Redundancy Model	Yes	
3.9	Dependencies Among SIs, Component Service Instances, and Components	Yes	Constituent sub-sections that are only partially supported or not supported are mentioned below. Note that the rest of the sub-sections are fully supported.
3.9.1.1	Dependencies Between SIs when Assigning a Service Unit Active for a Service Instance	No	
3.9.1.2	Impact of Disabling a Service Instance on the Dependent Service Instances	No	

Table 1-2 Compliance Table - Availability Service, SAI-AIS Volume 2: Availability Management Framework (continued)

Section	Description	Support	Notes
3.10	Component Monitoring	Yes	
3.11	Error Detection, Recovery, Repair, and Escalation Policy	Yes	Constituent sub-sections that are only partially supported or not supported are mentioned below. Note that the rest of the sub-sections are fully supported.
3.11.1.3	Recovery	Yes	AvSv currently does both the component failover and SU failover after doing quiesced of the enabled active components in the SU. Node failover and switch over are done after doing quiesced of all the enabled active components in the node. In all the above cases the components that did not fail in the SUs are not terminated.
3.11.1.4	Repair	Yes	Only automatic repair by restart is supported.
3.11.1.5	Recovery Escalation	Yes	
3.11.2.1	Recommended Recovery Action	Yes	Cluster Reset is not supported.
3.11.2.2	Escalations of Levels 1 and 2	Yes	
3.11.2.3	Escalations of Levels 3	Yes	
4	Local Component Life Cycle Management Interfaces	Yes	
5	Availability Management Framework API	Yes	
6	Basic Operational Scenarios	Yes	
Appendix A	Implementation of CLC Interfaces	Yes	

Table 1-3 Compliance Table - Availability Service, SAI-AIS Volume 3: Cluster Membership Service

Section	Description	Support	Notes
1	Document Introduction	N/A	Informational
2	Overview	Yes	
3	SA Cluster Membership Service API	Yes	

1.2.3 Dependencies

This section describes dependencies between the AvS and other services and between the Avs and libraries.

1.2.3.1 Service Dependencies

The following table lists other NCS services and how the Availability Service depends on them.

Table 1-4 Availability Service - Dependencies

Service	Dependency
Layered Enhanced Accelerated Portability (LEAP)	Availability Service uses LEAP for portability. It uses the memory manager, timers, encode-decode utility, and handle manager services provided by LEAP.
Message Distribution Service (MDS)	Interaction between the subparts of AvSv takes place using MDS messaging.
Distributed Tracing Service (DTSv)	Availability Service uses DTSv to log debug messages and to report informational events.
System Resource Monitoring Service (SRMSv)	Availability Service uses SRMSv for passive health monitoring of components.
Event Distribution Service (EDSv)	Availability Service uses EDSv to receive all fault events related to resources it manages.
Hardware Interface Service (HISv)	AvM uses HISv to issue hardware platform (HPL) commands for managing nodes.
Management Access Service (MASv)	AvD and AvM use MASv to manage the MIB objects defined in the AvSv MIBs.
Message Based Checkpoint Service (MBCSv)	The active AvD uses MBCSv to checkpoint the state information with a standby AvD.

1.2.3.2 Library Dependencies

The AvSv library, `libSaAmf.so`, and Cluster Membership library, `libSaClm.so`, depend on:

- libncs_core.so
- libavsv_common.so
- libsaf_common.so

1.2.4 Service Definition Documents

The documents available at the following links are SAF-standard documents. They provide the service definition for the Availability Service.

- http://www.saforum.org/apps/org/workgroup/twg/ais/download.php/1451/aisOverview.B0101.pdf
- http://www.saforum.org/apps/org/workgroup/twg/ais/download.php/1449/aisAmf.B0101.pdf
- http://www.saforum.org/apps/org/workgroup/twg/ais/download.php/1446/aisClm.B0101.pdf

The following information can be found in the referenced document:

- Service concept definitions and descriptions
- Functional behaviors and relationships
- A complete set of service data types exposed to the service user
- The set of Service APIs available to the service user

An application managed by AvSv is provided with APIs to perform the following:

- Component registration and unregistration
- Passive monitoring of processes of a component
- Component health monitoring
- Component service instance management
- Component life cycle management
- Protection group management
- Error reporting

An application also provides certain "Component Life Cycle" commands that are used by AvSv to instantiate, terminate, and clean it up. Applications can also use the APIs specified in Section 3 of the SAI-AIS-CLM-B.01.01 document to track and get information related to the nodes in the cluster.

1.2.5 Service Extensions

AvSv's AvM service is proprietary to Motorola. All functionality and associated behaviors identified in this documentation are supported, as explained in this document.

1.2.6 Implementation Notes

The MIB rows whose "row status" is "not active" are lost on failover or switchover of the active system management host. The configuration being performed when the failover/switchover occurs should be redone.

1.2.7 Configuration

An application managed by AvSv to provide high service availability must be modeled in terms of logical entities in accordance with the structure of the "System Model". Further, the application must implement the state models and callback interfaces according to the AMF specification.

Availability Service provides a management interface to configure the System Model and to perform runtime administrative operations. The System Model configuration can be done either using SNMP or XML, while the runtime administrative operations can be performed using SNMP or CLI.

Management Interface

2

2.1 Overview

The Availability Service is configured using SNMP or XML. Runtime administrative operations are performed using SNMP or CLI.

2.2 Management Information Base (MIB)

This section provides information about two standard MIBs (SAF-AMF-MIB and SAF-CLM-MIB) and two enterprise MIBs (NCS-AVSV-MIB and NCS-AVM-MIB) that AvSv supports.

2.2.1 NCS-AVSV-MIB

This proprietary MIB contains managed object definitions for NCS Availability Service. This MIB contains NCS enhancements beyond the SAF AMF and CLM MIBs. The MIB is in the development tar installation directory.

The following table describes the objects and traps supported by this MIB.

Table 2-1 NCS-AVSV-MIB

MIB Table ID\Trap ID	Description
Scalar objects	`ncsAvDHeartbeatSendInt` and `ncsAvDHeartbeatDownInt` are configuration objects used for heartbeating between the AvSv's distributed entities. The `ncsCLAdminState` object is not currently supported.
ncsNDTable	This table contains the proprietary configuration required for the node.The `ncsNDNodeId` in this table must be filled for each of the nodes configured in `saAmfNodeTable`.
ncsSGTable	This table contains the proprietary configuration required for the SGs specific to the NCS services that are managed by AvSv.
ncsSUTable	This table contains the MIB objects through which proprietary admin operations can be performed on the SU.
ncsSITable	This table contains the MIB objects through which proprietary admin operations can be performed on the SI.
ncsSNDTable	This table contains status information on the cluster nodes.
ncsSSUTable	This table contains status information of the SUs.

Table 2-1 NCS-AVSV-MIB (continued)

MIB Table ID\Trap ID	Description
ncsSCompTable	This table contains status information of the components in the cluster.
ncsInitSuccessOnNode	This trap will be generated when NCS initialization is successful on a particular node.
ncsAlarmCompFailOnNode	This trap notifies the system administrator about a component failure as well as reason of component failure.
ncsAlarmStateChgStartSUHaState	This trap will be generated whenever SUSI HA state starts changing.

2.2.2 NCS-AVM-MIB

This proprietary MIB is used to manage hardware deployment system configuration. It is in the development tar installation directory.

The following table describes the objects and traps supported by this MIB.

Table 2-2 NCS-AVM-MIB

MIB Table ID/Object ID	Description
ncsAvmEntDepolyTable	This table contains the hardware deployment configuration.
ncsAvmAdmSwitch	This is the scalar used to perform switchover of system manager hosts.
ncsAvmEntFaultDomainTable	Not supported

2.2.2.1 Example

To issue a lock on a node in physical slot 9 on chassis 2, the SNMP-SET would be:

```
snmpset -v2c  rwcommunity   ip-address
1.3.6.1.4.1.161.10.3.1.13.1.1.1.1.12.\"\{\{7,9\},\{23,2\},\{65535,0\}
\}\"  i  2
```

In this SNMP-SET:

- {{7,9},{23,2},{65535,0}} refers to the index.
- {7,9} refers to the blade in physical slot 9 (the entity type of the blade is 7 and its entity instance is 9).
- {23,2} refers to the chassis in location 2 (the entity type of the chassis is 23 and its entity instance is 2).
- {65535,0} is the default root entity in the system.

2.2.3 SAF-AMF-MIB

NCS 06A Availability Service supports an intermediate draft version of the SAF AMF MIB. This MIB for the Availability Service is compliant with the SAI-AIS-AMF-B.01.01 specification.The following table describes the standard AMF MIB tables and objects that are not implemented by Availability Service or have an anomaly in implementation.

Table 2-3 SAF-AMF-MIB

MIB Table ID/Object ID	Description
saAmfServiceState	This AMF scalar is optional. Not supported
saAmfCompDelayBetweenInstantiateAttempts mib object	Not supported
saAmfCompNodeRebootCleanupFail mib object	Not supported
saAmfCompNumMaxInstantiateWithDelay mib object	Not supported
saAmfCompNumMaxAmStopAttempts mib object	Not supported
saAmfCompPresenceState Mib object	Orphaned presence State(8) will returned in case of a orphaned component which is not there in the SAF-AMF-MIB.
saAmfCompInstantiateTimeout	Default value will be 2000000000 nano seconds.
saAmfCompTerminateTimeout	Default value will be 2000000000 nano seconds.
saAmfCompCleanupTimeout	Default value will be 2000000000 nano seconds.
saAmfCompAmStartTimeout	Default value will be 2000000000 nano seconds.
saAmfCompAmStopTimeout	Default value will be 2000000000 nano seconds.
saAmfSGFailbackOption mib object	This MIB object value can not be set to TRUE, it will be always FALSE.
saAmfSUFailOver mib object	Not supported
saAmfSISIDepTable Table	Not supported
saAmfAlarmServiceImpaired	Not supported
saAmfAlarmClusterReset	Not supported
saAmfCompQuiescingCompleteTimeout	Not supported

2.2.4 SAF-CLM-MIB

NCS 2.0 Availability Service supports an intermediate draft version of the SAF Cluster Member Service MIB. This MIB for the Availability Service is compliant with the SAI-AIS-CLA-B.01.01 specification.

The following table describes the standard CLM MIB tables and objects that are not implemented by Availability Service.

Table 2-4 SAF-CLM-MIB

MIB Table ID/Object ID	Description
saClmServiceState	This CLM scalar is not supported
saClmNodeAddressType mib object	Not Supported
saClmNodeAddress mib object	Not Supported
saClmNodeHPIEntityPath mib object	Not Supported
saClmAlarmServiceImpaired	Not Supported
saClmStateChgClusterNodeReconfigured	Not Supported

2.2.5 Example MIB Operations

This section describes the steps required to completely uninstall an application component on a sample node and then install it again.

Uninstalling an Application Component on a Sample Node

To uninstall all application SUs from a node, take the steps below.

Assume there is only one application SU (`safSu=Su_app,safNode=PL_2_2`) and only one application component on the node (`safComp=Comp_app, safSu=Su_app,safNode=PL_2_2`).

1. Perform an SNMP Set of SAF-AMF-MIB table saAmfSUTable index= ”safSu=Su_app,safNode=PL_2_2”, object saAmfSUAdminState value= locked(1).
2. Perform an SNMP Set of NCS-AVSV-MIB table ncsSUTable index= ”safSu=Su_app,safNode=PL_2_2”, object ncsSUTermState value= True(1).
3. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompCSTypeSupportedTable index1= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, index2=”CSI_TYPE_1”object saAmfCompRowStatus value= RowDestroy(6).
4. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompRowStatus value= RowDestroy(6).
5. Perform an SNMP Set of SAF-AMF-MIB table saAmfSUTable index= ” safSu=Su_app,safNode=PL_2_2”, object saAmfSURowStatus value= RowDestroy(6).

After this, the application can be uninstalled from the node.

Install an Application Component on a Sample Node

Assume there is only one application SU (safSu=Su_app,safNode=PL_2_2) and only one application component (safComp=Comp_app, safSu=Su_app,safNode=PL_2_2) to be installed on the node. Assume that this SU should be part of the SG “safSg=SG_app”.

1. Install the component on the node.
2. Perform an SNMP Set of SAF-AMF-MIB table saAmfSUTable index= ”safSu=Su_app,safNode=PL_2_2”, object saAmfSURank value= 3.
3. Perform an SNMP Set of SAF-AMF-MIB table saAmfSUTable index= ”safSu=Su_app,safNode=PL_2_2”, object saAmfSUNumComponents value= 1.
4. Perform an SNMP Set of SAF-AMF-MIB table saAmfSUTable index= ”safSu=Su_app,safNode=PL_2_2”, object saAmfSUParentSGName value= “safSg=SG_app”.
5. Perform an SNMP Set of SAF-AMF-MIB table saAmfSUTable index= ”safSu=Su_app,safNode=PL_2_2”, object saAmfSURowStatus value= RowActive(1).
6. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompCapability value= oneactiveoronestandby (4).
7. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompCategory value= saAware(0).
8. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompInstantiateCmd value= “The command to start the component”.
9. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompCleanupCmd value= “The command to cleanup the component”.
10. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompInstantiationLevel value= 1.
11. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompInstantiateTimeout value= 2000000000(2 seconds)
12. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompCleanupTimeout value= 2000000000(2 seconds).

13. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompTerminateCallbackTimeOut value= 2000000000(2 seconds).

14. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompRecoveryOnError value= componentrestart(2).

15. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompNumMaxInstantiate value=2.

16. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompDisableRestart value= False(2).

17. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompCSTypeSupportedTable index1= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, index2=”CSI_TYPE_1”object saAmfCompRowStatus value= RowActive(1).

18. Perform an SNMP Set of SAF-AMF-MIB table saAmfCompTable index= ” safComp=Comp_app, safSu=Su_app,safNode=PL_2_2”, object saAmfCompRowStatus value= RowActive(1).

2.2.6 AvSv Traps

Traps are defined in SAF-AMF-MIB, SAF-CLM-MIB and NCS-AVSV-MIB files. NCS defines a SAF Event Service channel called the NCS_TRAP channel. NCS services publish TRAPs as events on the NCS_TRAP channel. AvSv defines various Event filters that allow applications to subscribe to

- AMF TRAP events
- CLM TRAP events
- NCS-proprietary AvSv TRAP events

or any combination of the above.

Users can subscribe for all AvSv traps and partially subscribe for AMF traps, CLM traps and NCS traps. The following table describes the filters for AvSv traps.

Table 2-5 AvSv Traps

Filter	Description
AVSV_TRAPS	Common filter of all AvSv traps
SAF_AMF_MIB_TRAPS	Filter for AMF specific traps
SAF_CLM_MIB_TRAPS	Filter for CLM specific traps
NCS_AVSV_MIB_TRAPS	Filter for NCS Specific traps

The subscriber should use a combination of these filters, based on the requirements. For example, if a subscriber wants to receive all AvSv traps, AVSV_TRAPS filter should be used. If a subscriber wants to receive only AMF traps, the array of AVSV_TRAPS and SAF_AMF_MIB_TRAPS filters should be used.

2.2.7 XML

Motorola uses a proprietary XML schema for configuring entities in the AMF system model. The XML syntax is described in the XSD (the schema file) and the NCS System Description Programer's Reference. This schema has been developed to support the four MIBs described in the section *Management Information Base (MIB)* on page 23. The files in XML format the system description file, and the application config file is used only once during the initial system boot. The system description file also contains the hardware deployment configuration that AvM uses.

2.3 Command Line Interface

The Availability Service supports CLI commands for admin operations on Service Groups, Service Units, Service instances, and nodes.

To issue AvSv CLI commands the file cli_cefslib_conf should be updated with the entry:
`libavsv_clicef.so ncsavsv_cef_load_lib_req`

Only users with Admin permissions can access these commands. (Refer to the *Command Line Interface Programmer's Reference* for further intormation.)

2.3.1 set

Description

This command sets the admin state of the Service Group (SG), Service Unit (SU), Service instance (SI), or node to locked, unlocked, or shutting down.

Synopsis

```
set index (SG Name | SU Name |
SI Name | Node Name) adminstate value
(locked | unlocked | shuttingdown)
```

Parameters

`index`

Name of SG, SU, SI or node

`value`

Possible values are: locked|unlocked|shuttingdown

Example

Example Commands:

`set safSg=SG_NCS_DIRECTORS adminstate locked`

The previous command will lock the Service Group NCS DIRECTORS.

`set safNode=PL_2_6 adminstate unlocked`

The previous command will unlock the node PL_2_6.

`set safSu=SuT_NCS_Payload adminstate shuttingdown`

The previous command will shut down the Service Unit NCS_Payload

2.3.2 admin reset

Description

This command resets nodes at the location mentioned within.

AvSv supports two types of resets: soft and hard. With soft resets, applications on a node are first failed over, then gracefully shut down, and then an HPI command to gracefully reset the processor is issued.

Soft resets are not allowed on active system manager hosts.

With hard resets, an HPI command to abruptly reset the processor is issued.

Synopsis

```
reset /Shelf-id/Slot-id/Subslot-id/
operation softreset | hardreset
 /Shelf-Type/Blade-Type/Blade-Type
```

Parameters

`Shelf-id`

ID of the chassis.

`Slot-id`

ID of the physical slot.

`Subslot-id`

ID of the subslot. This field is optional. Users have to provide the ID only if there is any entity at level 3.

`softreset`

To reset a node gracefully.

`hardreset`

To reset a node abruptly.

`Shelf-Type`

The shelf type of the chassis in Shelf-Id. Default value: 23

`Blade-Type`

The type of blade in Slot-Id. Default value: 7.

`Blade-Type`

The type of blade in Subslot-Id.

Example

The following example commands illustrate the usage of this command.

```
[reset /2/9/ operation softreset
```

This command resets the node on chassis 2, physical slot 9.

Users can also issue this command as:

```
reset /2/9 operation softreset /23/7
```

This would reset the node on chassis 2, physical slot 9, shelf type 23, and blade type 7.

Note that the default entity types are supported only for shelf-type and first blade-type, i.e., 23 for shelf and 7 for blade. If there is a valid entry in the Subslot-Id, then users have to provide all the three entity types at the end.

2.3.3 admin lock

Description

This command locks or unlocks the node at the location mentioned within.

Three operations are supported: shutdown, lock, and unlock.

In the case of shutdown, applications are failed over and then shut down gracefully. The AvM deactivates the relevant blade gracefully using an HPI command and sets the state of the blade to "locked" The blade can then be brought up only when an administrator performs an explicit "unlock" operation. Shutdown cannot be performed on active system manager hosts.

In the case of a lock operation, the AvM deactivates the blade abruptly using an HPI command and sets the state of the blade to "locked" The blade cannot then be brought up unless an administrator performs an "unlock" operation.

An "unlock" operation is performed to bring up a blade that has been locked because of a shutdown or lock operation.

Synopsis

```
admreq /Shelf-id/Slot-id/Subslot-id/ operation
shutdown | lock | unlock
/Shelf-Type/Blade-Type/Blade-Type
```

Parameters

`Shelf-id`

ID of the chassis.

`Slot-id`

ID of the physical slot.

`Subslot-id`

ID of the subslot. This field is optional. Users have to provide the ID only if there is an entity at level 3. Default value: 0.

`shutdown`

To deactivate a node gracefully.

`lock`

To deactivate a node abruptly.

`unlock`

To unlock a node (if it has been shut down or locked).

`Shelf-Type`

The type of chassis corresponding to the chassis in Shelf-Id. Default value: 23.

`Blade-Type`

The type of blade corresponding to the blade in Slot-id. Default value: 7.

`Blade-Type`

The type of blade at the level corresponding to the blade in Subslot-id.

Example

Example Commands

`admreq /2/9/ operation shutdown`

This command deactivates the node on chassis 2, physical slot 9.

Users can also issue this command as:

admreq /2/9 operation shutdown /23/7

This command deactivates the node on chassis 2, physical slot 9, Shelf-Type 23, and Blade-Type 7.

Note that the default values are supported only for entities up to two levels of hierarchy, starting from Shelf, i.e., 23 for Shelf-Type and 7 for First Blade-Type. If there is a valid entry in Subslot-id, users should provide all the three entity types at the end.

2.3.4 admswitch

Description

This command performs a switchover of two system manager hosts. If the switchover is successful, the active host becomes a standby while the standby host becomes active.

Synopsis

```
admswitch
```

Sample Application

A

A.1 Overview

The sample AvSv application is a 'counter' application that is run in a 2N-redundancy model. The active entity counts periodically. When it fails, the standby entity becomes active and resumes counting from where the previous active entity failed.

The sample application shows you how to use some APIs defined in the SAI-AIS AMF service. It also demonstrates the following features:

- Passive monitoring
- Protection Group tracking
- Component failover (triggered by a component-generated error report) followed by component restart repair.
- AMF-invoked health check

A.1.1 Sequence of Events in the Sample Application

When the demo is started, AvSv instantiates two instances of the sample application per the configuration in the BOM. The sequence of events in both the applications is described below

Create 3 threads (one each for the counter application, AMF-INTF, and CKPT-INTF)

In the AMF-INTF thread:

1. Initialize with AMF
2. Call the AMF selection object
3. Call the API to get the component name
4. Register the component
5. Wait on the AMF selection object for callback events.

In the CKPT-INTF thread:

- Open the local checkpoint
- Initialize with CKPT
- Register the arrival callback
- Call the CKPT selection object
- Wait on the CKPT selection object for callback events

AMF dispatches CsiSetCallback with active/standby HA state.

In the active application:

1. Invoke the HA state handling callback function
2. Increment the counter and write it to the local checkpoint
3. Start the AMF initiated health check (as a result, the health check callbacks are dispatched by AMF periodically)
4. Stop responding to the health check after certain number of health checks
5. Send an error report with "component failover" as the recommended recovery

In the standby application:

1. Read the local checkpoint and update the counter value when standby assignment happens
2. Each update to the local checkpoint by the active results in a callback to the standby
3. Start tracking the protection group associated with the assigned CSI
4. Start and stop passive monitoring of the component

When the active application sends an error report, the standby application receives the active assignment

The new active application resumes incrementing the counter value

The new active application receives the protection group callback and stops tracking this protection group.

The previous active application is terminated

A new application is instantiated (as a part of repair)

The new active component then unregisters and finalizes with AMF

A.2 Configuration for the Sample Application

The configuration for the sample application is captured in the System Description File. It comprises of the following entities:

- A service group (SG) that comprises 2 service units (SU) in a 2N-redundancy model Each SU contains a single component.
- A single service instance (SI) is configured to be assigned to the SG
- The two SUs come up in the payload nodes (safNode=PL_2_3 and safNode=PL_2_4 respectively).

The sample application also provides scripts (available in `/opt/motorola/ncs/dev/source/avsv` directory on the development host) to control the component life cycle. These are:

- comp_inst.sh script (to instantiate the sample application)
- comp_term.sh script (to terminate the sample application).

A.3 Building the Sample Application

On the development host the sample application should be crosscompiled for the target architecture. To build the AvSv sample application, use the following command:

```
./make_env.sh  <target-architecture>  avsv_demo
```

This will generate a sample executable file `avsv_demo.out` in the `bin/<target-architecture>/ directory`.

A.4 Running the Sample Application

To run the sample application on the target architecture:

Procedure

1. Install NCS on the System Manager Node and two payload nodes (`safNode=PL_2_3` and `safNode=PL_2_4` respectively).

2. Transfer/install the sample application inventory on the target machine as follows.
   Transfer the sample program executable file (`avsv_demo.out`) to the payload nodes. Place it in `/etc/ncs/ folder`.
   Transfer the sample program scripts (`comp_inst.sh` and `comp_term.sh`) to the payload nodes. Place them in `/etc/ncs/` folder.

3. Ensure that the scripts have executable permission. Use the following command:
   `chmod +x comp_inst.sh`
   `chmod +x comp_term.sh`

4. Update the BOM on the system manager host. The configuration for the sample application is captured in `AppConfig.xml` file that is supplied along with the sample applications. They can be found (along with sample program scripts) in the `/opt/motorola/ncs/dev/source/avsv` directory on the development host.

5. Copy this BOM to the `/etc/ncs` folder on the System Manager Node. The sample application specific configuration attributes are commented in the `AppConfig.xml`. Remove the comments (the commented portions begin with a “<!--” and end with a “-->”).

6. Verify if the `/etc/ncs/pssv_spcn_list` file on the System Manager Node contains the string PSS. Replace it with an XML file. This is required to force initial configuration data read from the BOM file.

7. Remove the `avsv_demo.log` file, if any, from the `/ncs/log/stdouts` folder on the payload nodes. This file contains the output of the sample application.

Reboot the chassis. The entire system will come up with the sample application along with the NCS infrastructure elements. The `avsv_demo.log` file captures the output of the sample application on each payload node.

A.5 Sample Application Output

For the active node:

```
##############################################
#                                            #
#   You are about to witness AvSv Demo !!!   #
#                                            #
##############################################



AVSV-APP TASK CREATION SUCCESS !!!




CKPT :: CKPT-INTF TASK CREATION SUCCESS !!!




AMF-INTF TASK CREATION SUCCESS !!!



AMF Initialization Done !!!
AmfHandle: -35651582


CKPT Initialization Done !!!
CkptHandle: 2


CKPT :: Registered Arrival Callback !!!
```

```
CKPT :: Checkpoint Opened !!!

CKPT :: Ckpt Section Create being called ....PASSED


AMF Selection Object Get Successful !!!

CKPT :: Selection Object Get Successful !!!

Component Name Get Successful !!!
CompName: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3

Component Registered !!!

Dispatched 'CSI Set' Callback
Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3
CSIName: safCsi=Csi_AvSvDemo,safSi=Si_AvSvDemo
HAState: Active
CSIFlags: Add One


INVOKING saAmfHAStateGet() API !!!


COUNTER VALUE: 1


CKPT :: Wrote 1 to the CheckPoint
```

```
COUNTER VALUE: 2



CKPT :: Wrote 2 to the CheckPoint

CompName: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3

CSIName: safCsi=Csi_AvSvDemo,safSi=Si_AvSvDemo

HAState: Active



DEMONSTRATING AMF-INITIATED HEALTHCHECK !!!




COUNTER VALUE: 3



CKPT :: Wrote 3 to the CheckPoint



COUNTER VALUE: 4



CKPT :: Wrote 4 to the CheckPoint

Started AMF-Initiated HealthCheck (with Component Failover Recommended
Recovery)

Comp: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3

HealthCheckKey: A9FD64E12C


Dispatched 'HealthCheck' Callback

Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3

HealthCheckKey: A9FD64E12C
```

```
COUNTER VALUE: 5


CKPT :: Wrote 5 to the CheckPoint


COUNTER VALUE: 6


CKPT :: Wrote 6 to the CheckPoint

Dispatched 'HealthCheck' Callback
Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3
HealthCheckKey: A9FD64E12C


COUNTER VALUE: 7


CKPT :: Wrote 7 to the CheckPoint


COUNTER VALUE: 8


CKPT :: Wrote 8 to the CheckPoint


Dispatched 'HealthCheck' Callback
Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3
```

```
HealthCheckKey: A9FD64E12C


COUNTER VALUE: 9


CKPT :: Wrote 9 to the CheckPoint


COUNTER VALUE: 10


CKPT :: Wrote 10 to the CheckPoint

Dispatched 'HealthCheck' Callback
Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3
HealthCheckKey: A9FD64E12C


COUNTER VALUE: 11


CKPT :: Wrote 11 to the CheckPoint


COUNTER VALUE: 12


CKPT :: Wrote 12 to the CheckPoint


Dispatched 'HealthCheck' Callback
```

```
Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3
HealthCheckKey: A9FD64E12C


COUNTER VALUE: 13


CKPT :: Wrote 13 to the CheckPoint


COUNTER VALUE: 14


CKPT :: Wrote 14 to the CheckPoint

Dispatched 'HealthCheck' Callback
Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3
HealthCheckKey: A9FD64E12C


COUNTER VALUE: 15


CKPT :: Wrote 15 to the CheckPoint


COUNTER VALUE: 16


CKPT :: Wrote 16 to the CheckPoint
```

```
Dispatched 'HealthCheck' Callback
Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3
HealthCheckKey: A9FD64E12C


COUNTER VALUE: 17


CKPT :: Wrote 17 to the CheckPoint


COUNTER VALUE: 18


CKPT :: Wrote 18 to the CheckPoint

Dispatched 'HealthCheck' Callback
Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3
HealthCheckKey: A9FD64E12C


COUNTER VALUE: 19


CKPT :: Wrote 19 to the CheckPoint


COUNTER VALUE: 20


CKPT :: Wrote 20 to the CheckPoint
```

```
 Dispatched 'HealthCheck' Callback

 Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3

 HealthCheckKey: A9FD64E12C



 COUNTER VALUE: 21



 CKPT :: Wrote 21 to the CheckPoint



 COUNTER VALUE: 22



 CKPT :: Wrote 22 to the CheckPoint


 Dispatched 'HealthCheck' Callback

 Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3

 HealthCheckKey: A9FD64E12C


 Stopped HealthCheck for Comp:
safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3 with HealthCheckKey:
A9FD64E12C



 DEMONSTRATING COMPONENT FAILOVER THROUGH ERROR REPORT !!!




 COUNTER VALUE: 23
```

```
 CKPT :: Wrote 23 to the CheckPoint



 COUNTER VALUE: 24



 CKPT :: Wrote 24 to the CheckPoint


 Sent Error Report for Comp:
safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3 with CompFailover as
the recommended recovery



 ##############################################
 #                                            #
 #   You are about to witness AvSv Demo !!!   #
 #                                            #
 ##############################################



 AVSV-APP TASK CREATION SUCCESS !!!



 CKPT :: CKPT-INTF TASK CREATION SUCCESS !!!



 AMF-INTF TASK CREATION SUCCESS !!!
```

```
AMF Initialization Done !!!
AmfHandle: -38797310

CKPT Initialization Done !!!
CkptHandle: 3

CKPT :: Registered Arrival Callback !!!

CKPT :: Checkpoint Opened !!!

CKPT :: Ckpt Section Create being called ....PASSED



AMF Selection Object Get Successful !!!

CKPT :: Selection Object Get Successful !!!

Component Name Get Successful !!!
CompName: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3

Component Registered !!!

Dispatched 'CSI Set' Callback
Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3
CSIName: safCsi=Csi_AvSvDemo,safSi=Si_AvSvDemo
HAState: Active
CSIFlags: Add One



INVOKING saAmfHAStateGet() API !!!
```

```
COUNTER VALUE: 1


CKPT :: Wrote 1 to the CheckPoint


COUNTER VALUE: 2


CKPT :: Wrote 2 to the CheckPoint
CompName: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3
CSIName: safCsi=Csi_AvSvDemo,safSi=Si_AvSvDemo
HAState: Active


DEMONSTRATING AMF-INITIATED HEALTHCHECK !!!
```

For the stand-by node:

```
################################################
#                                              #
#   You are about to witness AvSv Demo !!!   #
#                                              #
################################################


AVSV-APP TASK CREATION SUCCESS !!!
```

```
CKPT :: CKPT-INTF TASK CREATION SUCCESS !!!



AMF-INTF TASK CREATION SUCCESS !!!



AMF Initialization Done !!!
AmfHandle: -38797310

CKPT Initialization Done !!!
CkptHandle: 2

CKPT :: Registered Arrival Callback !!!

CKPT :: Checkpoint Opened !!!

CKPT :: Ckpt Section Create being called ....PASSED



AMF Selection Object Get Successful !!!

CKPT :: Selection Object Get Successful !!!

Component Name Get Successful !!!
CompName: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_2

Component Registered !!!

Dispatched 'CSI Set' Callback
```

```
Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_2
CSIName: safCsi=Csi_AvSvDemo,safSi=Si_AvSvDemo
HAState: Standby
CSIFlags: Add One

Started Protection Group Tracking
CSI: safCsi=Csi_AvSvDemo,safSi=Si_AvSvDemo
Track Flags: Changes Only

CKPT :: Read 8 during initial read

CKPT :: Read 9 from the CheckPoint
CKPT :: Read 10 from the CheckPoint
CKPT :: Read 11 from the CheckPoint
CKPT :: Read 12 from the CheckPoint
CKPT :: Read 13 from the CheckPoint
CKPT :: Read 14 from the CheckPoint
CKPT :: Read 15 from the CheckPoint
CKPT :: Read 16 from the CheckPoint
CKPT :: Read 17 from the CheckPoint
CKPT :: Read 18 from the CheckPoint
CKPT :: Read 19 from the CheckPoint
CKPT :: Read 20 from the CheckPoint
CKPT :: Read 21 from the CheckPoint
CKPT :: Read 22 from the CheckPoint
CKPT :: Read 23 from the CheckPoint
CKPT :: Read 24 from the CheckPoint
Dispatched 'Protection Group' Callback
CSI: safCsi=Csi_AvSvDemo,safSi=Si_AvSvDemo
No. of Members: 2
CompName[0]: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_3
```

```
 Rank[0]    : 1

 HAState[0] : Quiesced

 Change[0]  : State Change



 Stopped Protection Group Tracking for CSI:
safCsi=Csi_AvSvDemo,safSi=Si_AvSvDemo


 Started Passive Monitoring for Comp:
safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_2


 Stopped Passive Monitoring for Comp:
safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_2


 Dispatched 'CSI Set' Callback

 Component: safComp=CompT_AvSvDemo,safSu=SuT_AvSvDemo,safNode=PL_2_2

 CSIName:

 HAState: Active

 CSIFlags: Target All



 DEMO OVER (UNREGISTER & FINALIZE THE COMPONENT) !!!




 COUNTER VALUE: 25



 CKPT :: Wrote 25 to the CheckPoint



 COUNTER VALUE: 26
```

```
CKPT :: Wrote 26 to the CheckPoint
Component UnRegistered !!!


COUNTER VALUE: 27


CKPT :: Wrote 27 to the CheckPoint


COUNTER VALUE: 28


CKPT :: Wrote 28 to the CheckPoint
AMF Finalize Done !!!


DEMO OVER !!!
```

Related Documentation

B

B.1 Motorola Embedded Communications Computing Documents

The Motorola publications listed below are referenced in this manual. You can obtain electronic copies of Embedded Communications Computing (ECC) publications by contacting your local Motorola sales office or by visiting ECC's World Wide Web literature site: http://www.motorola.com/computer/literature. This site provides the most up-to-date copies of ECC product documentation.

Table B-1 Motorola Publications

Document Title	Publication Number
Availability Service Programmer's Reference	6806800C44
Avantellis 3000 Series Rel. 3.0 User' s Guide	6806800B91
Checkpoint Service Programmer's Reference	6806800C47
Command Line Interface Programmer's Reference	6806800C11
Distributed Tracing Service Programmer's Reference	6806800B40
Event Distribution Service Programmer's Reference	6806800C48
Global Lock Service Programmer's Reference	6806800C49
HPI Integration Service Programmer's Reference	6806800C51
Interface Service Programmer's Reference	6806800B50
LEAP Programmer's Reference	6806800B56
Management Access Service Programmer's Reference	6806800B55
Message Based Checkpointing Service Programmer's Reference	6806800B41
Message Distribution Service Programmer's Reference	6806800B89
Message Queue Service Programmer's Reference	6806800C50
NetPlane Core Services Overview User's Guide	6806800C08
Persistent Store Restore Service Programmer's Reference	6806800B54
Simple Software Upgrade Programmer's Reference	6806800B19
SMIDUMP Tool Programmer's Reference	6806800B37
SNMP SubAgent Programmer's Reference	6806800B38
System Description Programmer's Reference	6806800B90
System Resource Monitoring Service Programmer's Reference	6806800B39

B.2 Related Specifications

For additional information, refer to the following table for related specifications. As an additional help, a source for the listed document is provided. Please note that, while these sources have been verified, the information is subject to change without notice.

Table B-2 Related Specifications

Document Title	Version/Source
Service Availability Forum Application Interface Specification, Volume 1, Overview and Models	SAF-AIS-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 2, Availability Management Framework	SAF-AIS-AMF-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 3, Cluster Membership Service	SAF-AIS-CLM-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 4, Checkpoint Service	SAF-AIS-CKPT-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 5, Event Service	SAF-AIS-EVT-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 6, Message Service	SAF-AIS-MSG-B.01.01/ http://www.saforum.org
Service Availability Forum Application Interface Specification, Volume 7, Lock Service	SAF-AIS-LCK-B.01.01/ http://www.saforum.org