MOTOROLA

Checkpoint Service

Programmer's Reference

6806800C47B

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 The text itself may not be published commercially in print or electronic form, edited, translated, or otherwise altered without the permission of Motorola,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, on page 13*
  Provides an overview of the Cechpoint service functionilty and provides references to standard SAF documents.
- Chapter 2, *API Description, on page 17*
  Provides information that is required when writing applications that make use of the Checkpoint service. It also explains non-standard extensions that were added to the service.
- Appendix A, *Sample Application,* on page 23
  Describes the sample application that is available for the Checkpoint service
- Appendix B, *Related Documentation, on page 25*
  Provides references to related user documentation and standard specifications.

Abbreviations

This document uses the following abbreviations:

Abbreviation	Definition
AIS	Application Interface Specification
AMF	Availability Management Framework
API	Application Programming Interface
AvSv	Availability Service
CLI	Command Line Interface
CLM	Cluster membership Service
CPA	Checkpoint Agent
CPD	Checkpoint Director
CPND	Checkpoint Node Director
CPSv	Checkpoint service
DTSv	Distributed Tracing Service
HPI	Hardware Platform Interface
LEAP	Layered Environment for Accelerated Portability
MBCSv	Message-Based Checkpoint Service

Abbreviation	Definition
MDS	Message Distribution Service
MIB	Management Information Base
NCS	Netplane Core Services
SAF	Service Availability Forum

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
. . .	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
	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
6806800C47A	February 2007	First edition
6806800C47B	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 Checkpoint Service provides a facility for processes to record checkpoint data incrementally, which can be used to protect an application against failures. When recovering from fail-over or switch-over situations, or restart situations, the checkpoint data can be retrieved, and execution can be resumed from the state recorded before the failure.

Checkpoints are cluster-wide entities that are designated by unique names. A copy of the data stored in a checkpoint is called a checkpoint replica, which is stored in the main memory rather than on disk for performance reasons. A given checkpoint may have several checkpoint replicas stored on different nodes in the cluster to protect it against node failures.To avoid accumulation of unused checkpoints in the system, checkpoints have a retention time. When a checkpoint has not been opened by any process for the duration of the retention time, the Checkpoint Service automatically deletes the checkpoint.

The CPSv service supports the following two types of update options:

- Asynchronous update option
- Synchronous update option

In the case of asynchronous update option, one of the replicas is designated as the active replica. Data is always read from the active replica and there is no guarantee that all the other replicas contain identical data. A write call returns after updating the active replica.

In the case of synchronous update options the call invoked to write to the replicas returns only when all replicas have been updated, i.e. either all replicas are updated or the call fails and no changes are made to the replicas.

The CPSv supports both collocated and non-collocated checkpoints. In case of checkpoints opened with collocated and asynchronous update option, it is up to the application to set a checkpoint to the active state. In all other cases the CPSv itself handles which checkpoint is currently active.

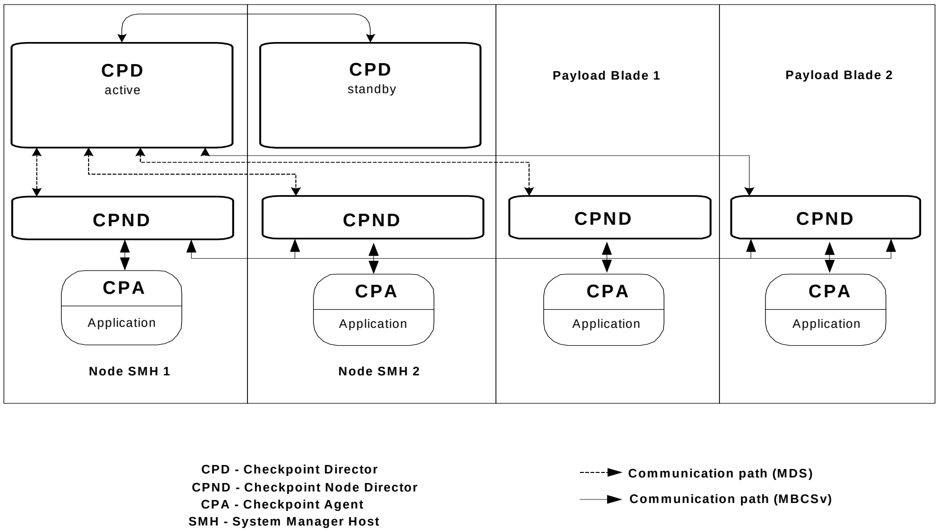
The CPSv defined by SAF does not support hot-standby. This means that the currently stand-by component is not notified of any changes made to the checkpoint. When the stand-by component gets active, it has to iterate through the respective checkpoint sections to get up-to-date. To overcome this drawback, the CPSv provides additional, non-SAF APIs which help to notify the stand-by component of changes and thus facilitate the implementation of a hot-stand-by.

1.2 Models and Concepts

The Checkpoint service comprises three distributed subparts that maintain the cluster-wide checkpoint database.

- Checkpoint Director
- Checkpoint Node Director
- Checkpoint Agent

Figure 1-1 Checkpoint Service -Subparts



1.2.1 Checkpoint Director

Checkpoint Director (CPD) runs as a process on a system manager node. CPD maintains the centralized repository of control information for all checkpoints created in the cluster. The CPD also maintains the location information of active replicas for all the checkpoints opened in the cluster. In case of non-collocated checkpoint, the CPD designates a particular node to manage an active replica for that checkpoint and also decides on the number or replicas to be created which depends on the policy (See section 8.1.5.1 Usage of Non-Collocated Checkpoints, for policies). Two instances of CPD are configured, one on each system manager node, in order to achieve high-availability. The two instances are configured to be part of a service group having a 2N redundancy model.

1.2.2 Checkpoint Node Director

There is one instance of the Checkpoint Node Director (CPND) on each system manager and payload nodes. It is modeled as a separate process. CPND maintains the detailed information of the Checkpoints referred from that node and the corresponding updates and retrievals that operate on those checkpoints. CPND also handles the requests issued by the CPA instances on behalf of its client applications on the same node. In case of checkpoints that have been created with the collocated attribute and the asynchronous update option, the application will choose the CPND that oversees the active replica of a particular checkpoint via the invocation of the `saCkptActiveReplicaSet()` API. In all other cases, the CPD will designate the CPND that oversees the active replica. The CPND that oversees the active replica of a particular checkpoint will control all the operations on that checkpoint and it is not constrained to be present on the same node where the application resides. The CPND that manages the active replica of a particular checkpoint serializes all the requests to that checkpoint from all the applications present at different nodes in the cluster.

1.2.3 Checkpoint Agent

The Checkpoint Agent (CPA) is a linkable library, which conforms to the SAF APIs described in the document SAF-AIS-CKPT-B.01.01. The CPA library runs in the context of the application processes that initialize the CPA library. The SAF APIs are part of this library through which different checkpoint requests can be issued by the application processes.

1.3 Compliance Report

Checkpoint Service conforms to the Checkpoint specification mentioned in SAF-AIS-CKPT-B.01.01. The table given below provides the specification conformance report specific to this release.

Table 1-1 Compliance Table - Checkpoint Service

Section	Description	Supported
3.1.1	Checkpoints	Yes
3.1.2	Sections	Yes
3.1.3	Checkpoint Replica	Yes
3.1.4	Checkpoint Data Access	Yes
3.1.5	Synchronous Update	Yes
3.1.6	Asynchronous Update	Yes
3.1.7	Collocated and Non-Collocated Checkpoint	Yes
3.1.8	Active Replica	Yes
3.1.9	Persistence of Checkpoints	Yes
3.2	Include File and Library Names	Yes
3.3	Type Definitions	Yes
3.4	Library Life Cycle	Yes
3.5	Checkpoint Management	Yes
3.6	Section Management	Yes
3.7	Data Access	Yes

1.4 Related SAF Standard Documents

The document SAF-AIS-CKPT-B.01.01 is an SAF standard document. It provides the service definition of the Checkpoint service and can be found at the following location: *http://www.saforum.org/apps/org/workgroup/twg/ais/download.php/1445/aisCkpt.B0101.pdf*

The following information can be found in the 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

API Description

2

2.1 Service Extensions

The current release of NCS Checkpoint Service provides one API and a callback function in addition to the APIs defined in the SAF-AIS Checkpoint Service document SAF-AIS-CKPT-B.01.01. These APIs are defined as 'stand-alone' APIs so that other SAF-defined APIs are not disturbed, and compliance to SAF is not compromised. These extensions are defined to provide the hot-standby support to the Checkpoint Service user applications.

2.1.1 ncsCkptRegisterCkptArrivalCallback()

Prototype

```
SaAisErrorTncsCkptRegisterCkptArrivalCallback(
    SaCkptHandleT     ckptHandle,
    ncsCkptCkptArrivalCallbackT    ckptArrivalCallback
);
```

Parameters

The following table describes the possible parameters.

Table 2-1 ncsCkptRegisterCkptArrivalCallback() Parameters

Parameter	Description
ckptHandle - [in]	The handle obtained through the `saCkptInitialize()` function, designating this particular initialization of the Checkpoint Service.
ckptArrivalCallback - [in]	The function pointer that the CKPT service shall invoke whenever an opened checkpoint scoped to `ckptHandle` is updated.

Description

This call registers the function callback that will be invoked whenever a opened checkpoint scoped to ckptHandle is updated. Though it can be invoked any time, the most likely time to invoke is just after `saCkptInitialize()` has been invoked. A client will not invoke this call at all if it does not wish to be notified in real-time about checkpoint updates.

Return Values

The following table lists possible return values of this call.

Table 2-2 ncsCkptRegisterCkptArrivalCallback() Return Values

Return Value	Description
SA_AIS_OK	The function completed successfully
SA_AIS_ERR_LIBRARY	An unexpected problem
SA_AIS_ERR_BAD_HANDLE	the handle ckptHandle is invalid
SA_AIS_ERR_INVALID_PARAM	the callback function pointer is wrong
SA_AIS_ERR_NO_MEMORY	out of memory

2.1.2 (*ncsCkptCkptArrivalCallback)()

Prototype

```
typedef void(*ncsCkptCkptArrivalCallbackT)(
   Const SaCkptCheckpointHandleT  checkpointHandle,
   SaCkptIOVectorElementT   *ioVector,
   SaUnit32T    numberOfElements
  );
```

Parameters

The following table lists possible parameters.

*Table 2-3 (*ncsCkptCkptArrivalCallback)() Parameters*

Parameter	Description
checkpointHandle - [in]	Handle to the checkpoint that is available for reading.
ioVector - [in]	Pointer to a vector that contains elements ioVector[0],…,ioVector[numberOfElements - 1]. Each element is of the type saCktptIOVectorElementT, defined in Section 3.3.4.1 of the document SAF-AIS-CKPT-B.01.01, which contains the following fields: ● sectionId - [in] the identifier of the section available for reading. ● dataBuffer - [in] Always set to NULL. ● dataSize - [in] size of data available for reading. ● dataOffset - [in] offset in the section that marks the start of the data that is available for reading. ● readSize - [in] Always set to 0.
numberOfElements - [in]	the size of the ioVector.

Description

If a callback of this form has been registered with the Checkpoint service via `ncsCkptRegisterCkptArrivalCallback()`, then it will be invoked whenever new or updated checkpoint replica data arrives for the checkpoint identified by checkpointHandle. The checkpoint writer is never called back. Also, applications that have not opened the checkpoint with the SA_CKPT_CHECKPOINT_READ flag are not called back. This callback is invoked in the context of a thread issuing `saCkptDispatch()` call.

The expected behavior for the client application is to take these very same arguments and use them as-is to invoke `saCkptCheckpointRead()`, thus fetching the section data that has been modified in the checkpoint.

For the NCS implementation, this callback function shall report that the data available for reading is exactly the same set of data that was described and written by the checkpoint writer that invoked one of `saCkptCheckpointWrite()`, `saCkptSectionOverwrite()` or `saCkptSectionCreate()`. This means/implies that our NCS implementation shall deliver checkpoint data in exactly the same units as was written. However, note that this callback is not invoked when a section is deleted by a writer using the `saCkptSectionDelete()` API. Therefore this service extension can only be used if sections created are expected to exist through the lifetime of the distributed application, i.e. sections that are created by the service are never deleted.

This function does not conflict or affect the behavior of any other SAF Checkpoint function.

Return Values

n.a.

2.2 Implementation Notes

This section summarizes important information that should be kept in mind when writing applications that make use of the Checkpoint service.

2.2.1 Usage of Non-Collocated Checkpoints

Checkpoints created without the collocated attribute are called non-collocated checkpoints. The management of replicas of non-collocated checkpoints and whether they are active or not is the responsibility of the Checkpoint Service.

For the non-collocated Checkpoints, NCS06A Checkpoint Service will specify the location of the checkpoint replicas as per the following policy:

- If a non-collocated checkpoint is opened for the first time by an application residing on a payload blade, the replicas will be created on the local payload blade and both the system manager nodes. In this case, the replica residing on the payload blade is designated as active replica.
- If a non-collocated checkpoint is opened for the first time by an application residing on the system manager nodes, the replica will be created only on the system manager blade. In this case, this replica on a system manager node will act as the active replica.
- If another application opens the same checkpoint from a payload node, the checkpoint service will not create the replica on that node.

Creating extra replicas on the system manager node for non-collated checkpoints is an overhead. The advantage of a non-collocated checkpoint is that replica will be created in two places, no matter from how many nodes it is opened.

2.2.2 Time-out Arguments for Checkpoint Service APIs

For all synchronous API calls, the application will provide the “timeout” argument. The application will consider invocation of the particular API failed in case it did not complete the call by the specified time. CPSv requires that the value passed in the timeout argument is greater than 100000000 nano seconds (100 milliseconds).

2.2.3 Cancellation of Pending Callbacks

According to the SAF-AIS-CKPT-B.01.01 specification, whenever a checkpoint is closed, all the pending callbacks corresponding to this checkpoint should be cancelled. In CPSv, implementation does cancel the pending callbacks related to closed checkpoints. However, the selection object already raised and related to cancelled pending callbacks, will not be cleared or reset. Due to this, `saCkptDispatch` API may return without invoking callback routine.

2.2.4 Maximum Number of Replicas Per Node

CPSv applications can create upto 1,000 replicas per node at a given instance. This includes the replicas created by CPSv for non-collocated checkpoints as per the “replica creation policy.”

In the case of collocated checkpoints, CPSv returns SA_AIS_ERR_NO_RESOURCES if an application attempts to create a new checkpoint and the current number of replicas on the local node is already the maximum that CPSv can support per node.

In the case of non-collocated checkpoints, CPSv returns an SA_AIS_ERR_NO_RESOURCES if the number of checkpoint replicas on the node on which CPSv decides to create a replica is already the maximum that CPSv can support per node. In all other cases, the checkpoint open does not return an error but the replicas will not be created on the backup nodes as decided by the “replica creation policy”.

2.2.5 Handling of SA_AIS_ERR_TRY_AGAIN

If the Checkpoint service API returns SA_AIS_ERR_TRY_AGAIN, the application should attempt the API call only after a couple of milliseconds. The suggested wait time is 3 seconds and the number of retries are 12.

Note that the Checkpoint write,overwrite, and read operations may sometime return SA_AIS_ERR_TRY_AGAIN if called simultaneously. This is to avoid any inconsistencies in the checkpoint database.

2.3 Configuration

This section describes how the Checkpoint service is preconfigured regarding shared memory and the maximum write data size.

2.3.1 Shared Memory Configuration

NCS3.0 Checkpoint service uses the shared memory for storing the checkpoint replicas. Checkpoint service will manage the shared memory segments created by it for storing the checkpoint replicas. The shared memory requirements for storing the checkpoint replica can be derived from the checkpoint creation attributes supplied at the time of saCkptCheckpointOpen( ) or saCkptCheckpointOpenAsync( ) call using the formula. `maxSections * maxSectionSize`

The maximum size of the shared memory segment is limited by the operating system. In most of the cases, the maximum value is 31MB. This can be found by executing the command: `cat /proc/sys/kernel/shmmax`

To increase the shared memory size to the desired value, one can use the following command: **`echo 134217728 >/proc/sys/kernel/shmmax`**

The above example command will set the maximum shared memory segment value to 27MB.

2.3.2 Maximum Data Size Per One write or Overwrite

The maximum data size per one write or over write is 40MB. Applications that try to write more than 40MB data in one `saCkptSectionWrite( )` or `saCkptSectionOverwrite( )` call will get the error SA_AIS_ERR_NO_RESOURCES.

2.4 Service Dependencies

The internal interfaces of the Checkpoint service are given below:

- Layered Environment for Accelerated Portability (LEAP) - for Shared Memory: Checkpoint Service uses LEAP for portability. The service uses the memory manager, timers, encode-decode utility and handle manager services provided by the LEAP.
- Message Distribution Service (MDS) - for Messaging: All the interaction between the different subparts of the Checkpoint service will take place using MDS messaging. The MDS is also used to register the service up and down events to handle the failure cases.
- Distributed Tracing Service (DTSv) - for Logging messages: Checkpoint service uses DTSv to log debug messages, which are stored in a file and could be used for debugging and to report informational events.
- Availability Service (AvSv) - for High Availability: CPD and CPND are modelled as AMF components.
- Message based Checkpoint Service (MBCSv) - for checkpointing information: CPD uses the MBCSv to checkpoint the state information with the standby CPD.
- Cluster Membership Service (CLM) - for Node names: CPD uses Cluster membership service to get the node name for a given node ID. Node names are required to implement Checkpoint service MIBs.

The Checkpoint library libSaCkpt.so depends on functions found in the following library: `libncs_core.so`

2.5 Management Interface

SAF-CHK-SVC-MIB is defined by SA forum's systems management WG. This MIB provides the manageable objects to access the cluster wide created checkpoint properties, location of the checkpoint replicas, version supported etc. This MIB also defines the traps to notify the errors like no more sections, sections available now etc.

NCS Checkpoint Service implements a draft version of SAF-CKPT-SVC-MIB, which aligns with B.01.01 version of CKPT. Checkpoint Service does not support the Notifications and Traps defined in SAF-CKPT-SVC-MIB.

The following table describes the MIB objects and traps supported by NCS Checkpoint Service:

Table 2-4 SAF-CHK-SVC-v7_5 MIB

MIB table id \ trap id	Description
safSpecVersion	Supported
safAgentVendor	Supported
safAgentVendorProductRev	Supported
safServiceStartEnabled	Supported. Always set to FALSE
saCkptCheckpointTable	Supported
saCkptNodeReplicaLocTable	Supported
saCkptAlarmServiceImpaired	Not Supported
saCkptStateChgNoMoreSections	Not Supported
saCkptStateChgSectionsAvailable	Not Supported

Command Line Interface (CLI) is not supported by Checkpoint Service

Sample Application A

A.1 Overview

The sample application provided here consists of two application processes that use the Checkpoint service APIs to ‘write’ to a checkpoint, and ‘read’ the checkpoint data written by the first application process.

A.2 Run the Checkpoint Service Demo

This sample application assumes that the NCS software is installed and running on the target system. Refer to the *Avantellis 3000 Series Rel. 3.0 User’ s Guide* for information on how to install the NCS software.

Running the demo application:

To run the checkpoint service demo, follow the steps given here:

1. Build the sample program to create the executable file `cpsv_demo.out.` (Refer to section - A.4.2 "Make" Commands of the NetPlane Core Services Overview User’s Guide, Part Number: 6806800C08 for more details)
2. Copy the executable file to the target. (Refer to section - A.4.2 "Make" Commands of the NetPlane Core Services Overview User’s Guide, Part Number: 6806800C08 for more details)

Ensure the `cpsv_demo.out` has executable permission. To give executable permission, use the following command:

```
chmod +x cpsv_demo.out
```

3. Open two terminals, and change to the directory where the executable `cpsv_demo.out` is copied.
4. Execute the following command in the first terminal. This application process will act as “MESSAGE-WRITER”.`./cpsv_demo.out 1`
5. Execute the following command in the second terminal. This application process will act as “MESSAGE-READER”. `./cpsv_demo.out 0.`
   The output will be displayed on both the terminals. Refer "Appendix A, *Sample Application Output*" of this document.

A.3 Sample Application Output

```
MESSAGE_ WRITER

Ckpt Initialising being called ....     PASSED

Ckpt Open being called ....     PASSED

Ckpt Active Replica Set being called ....      PASSED

Ckpt Section Create being called ....   PASSED

Ckpt Write being called with data: The Checkpoint Service provides a
facility for processes to record checkpoint data               ....
PASSED

Ckpt Synchronize being called ....      PASSED

Ckpt Unlink being called ....   PASSED

Ckpt Close being called ....    PASSED

Ckpt Finalize being called .... PASSED


MESSAGE_READER

Ckpt Initialising being called ....     PASSED

Ckpt Open being called ....     PASSED

Ckpt Read being called, data in the read buffer is: The Checkpoint
Service provides a facility for processes to record checkpoint data
....  PASSED

Ckpt Synchronize being called ....      PASSED

Ckpt Close being called ....    PASSED

Ckpt Finalize being called .... PASSED
```

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	6806800B08
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