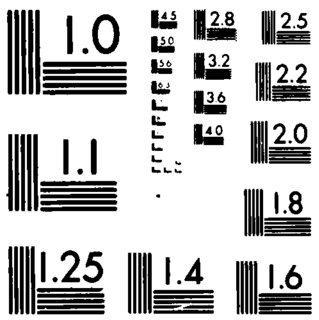
AD-A109 980   COMPUTER SCIENCES CORP FALLS CHURCH VA   F/G 9/2
ADA INTEGRATED ENVIRONMENT II COMPUTER PROGRAM DEVELOPMENT SPEC--ETC(U)
DEC 81   F30602-80-C-0292
UNCLASSIFIED   RADC-TR-81-364-PT-1   NL

1 of 4
AD
A109980

A109 98



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

PHOTOGRAPH THIS SHEET

AD A109980

DTIC ACCESSION NUMBER

LEVEL

INVENTORY

I

Computer Sciences Corp.
Falls Church, VA
ADA Integrated Environment II Computer Program Development Specification. Interim Rept. 15 Sep. 80-15 Mar 81 Dec. 81

DOCUMENT IDENTIFICATION

Contract F30602-80-C-0292 RADC-TR-81-364, Part 1

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DISTRIBUTION STATEMENT

ACCESSION FOR		
NTIS	GRA&I	☒
DTIC	TAB	□
UNANNOUNCED		□
JUSTIFICATION		
BY		
DISTRIBUTION /		
AVAILABILITY CODES		
DIST	AVAIL AND/OR SPECIAL	
A		

DISTRIBUTION STAMP

DATE ACCESSIONED

82 01 12 003

DATE RECEIVED IN DTIC

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-DDA-2

DOCUMENT PROCESSING SHEET

RADC-TR-81-364, Part I
Interim Report
December 1981

ADA INTEGRATED ENVIRONMENT II COMPUTER PROGRAM DEVELOPMENT SPECIFICATION

Computer Sciences Corporation

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-364, Part 1 has been reviewed and is approved for publication.

APPROVED: Donald F Roberts

DONALD F. ROBERTS
Project Engineer

APPROVED:

JOHN J. MARCINIAK, Colonel, USAF
Chief, Command and Control Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
RADC-TR-81-364, Part 1

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
ADA INTEGRATED ENVIRONMENT II COMPUTER PROGRAM DEVELOPMENT SPECIFICATION

5. TYPE OF REPORT & PERIOD COVERED
Interim Report
15 Sep 80 - 15 Mar 81

6. PERFORMING ORG. REPORT NUMBER
N/A

7. AUTHOR(s)

8. CONTRACT OR GRANT NUMBER(s)
F30602-80-C-0292

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Computer Sciences Corporation
803 Broad Street
Falls Church VA 22046

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
62204F/62702F/33126F
55811918

11. CONTROLLING OFFICE NAME AND ADDRESS
Rome Air Development Center (COES)
Griffiss AFB NY 13441

12. REPORT DATE
December 1981

13. NUMBER OF PAGES
310

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)
Same

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES
RADC Project Engineer: Donald F. Roberts (COES)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Ada
Compiler
Database
KAPSE
MAPSE
Kernel
Debugger
APSE
AIE
Integrated environment
Editor

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Ada Integrated Environment (AIE) consists of a set of software tools intended to support design, development and maintenance of embedded computer software. A significant portion of an AIE includes software systems and tools residing and executing on a host computer (or set of computers). This set is known as an Ada Programming Support Environment (APSE). This B-5 Specification describes, in detail, the design for a minimal APSE, called a MAPSE. The MAPSE is the foundation upon which an

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

APSE is built and will provide comprehensive support throughout the design, development and maintenance of Ada software. The MAPSE tools described in this specification include an Ada compiler, linker/loader, debugger, editor, and configuration management tools. The kernel (KAPSE) will provide the interfaces (user, host, tool), database support, and facilities for executing Ada programs (runtime support system).

INTRODUCTION

This document presents the Computer Program Development Specifications (Type B5) for the Computer Program Configuration Items (CPCIs) for the CSC/SEA design of the Ada Integrated Environment (AIE) under Rome Air Development Center (RADC) Contract Number F30602-80-C-0292. These specifications are comprised of the following volumes:

PART I:

Volume 1, Computer Program Development Specification for CPCI KAPSE Framework.

Volume 2, Computer Program Development Specification for CPCI KAPSE Data Base System.

PART II:

Volume 3, Computer Program Development Specification for CPCI APSE Command Language Interpreter.

Volume 4, Computer Program Development Specification for CPCI MAPSE Configuration Management-System.

Volume 5, Computer Program Development Specification for CPCI Ada Compiler.

Volume 6, Computer Program Development Specification for CPCI MAPSE Linker.

Volume 7, Computer Program Development Specification for CPCI MAPSE Editor.

Volume 8, Computer Program Development Specification for CPCI MAPSE Debugger.

Accompanying this document is an Interim Technical Report (ITR), which describes the principles influencing the preliminary design and provides the rationale for the decisions made, and the System Specification (Type A), which presents the functional requirements for the AIE.

Table 1 provides a cross-reference between the AIE Statement of Work (SOW) and the specifications.

PHASE I SOW REQUIREMENTS	A - SPEC.	B5 - SPEC.
4.1 Phase I Design		
4.1.1 General Requirements	3.1.1	
4.1.1.1 Data Base Support, Interfaces to host facilities (H.W. & S.W.), user interfaces, tool interfaces	3.1.1.1 3.1.1.2 3.1.4 3.7.1	KDBS - 3.2.5 3.3 ACLI - 3.2.5 3.3
4.1.1.2 Portable to maximum extent possible, External interfaces should be clearly isolated, clearly identified	3.1.1.1 3.1.1.2 3.1.2 3.1.4 3.1.5 3.1.5.2	KFW - 3.1.1 KDBS - 3.1.1 ACLI - 3.1.1 CMS - 3.1.1 Compiler - 3.1.1 Linker - 3.1.1 Editor - 3.1.1 Debugger - 3.1.1
4.1.1.3 Specify uniform protocol conventions between user, tools and MAPSE/KAPSE, formats for invoking KAPSE/MAPSE facilities should be uniform or identical	3.1.5.1 3.1.5.2	KDBS - 3.3 KFW - 3.2.5 ACLI - Command Utilities
4.1.1.4 Shall include features to protect itself from user and system errors	3.2.3 3.2.5 3.3.7	KDBS - 3.2.5.7 3.2.5.8 3.3.6 3.3.7
4.1.1.5 Software should be modular and reusable	3.7	KFW - 3.1.1 KDBS - 3.1.1 ACLI - 3.1.1 CMS - 3.1.1 Compiler - 3.1.1 Editor - 3.1.1 Linker - 3.3.1 Debugger - 3.1.1

PHASE I SOW REQUIREMENTS	A - SPEC	B5 - SPEC
4.1.2 KAPSE DATA BASE REQUIREMENTS		
4.1.2.1 Capability to create, delete, modify, store, retrieve, input, and output data base objects	3.7.1.2	KDBS - 3.2.5.6 3.3.4
4.1.2.2 Shall provide for all forms of data necessary to fulfill all SOW Requirements	3.7.1.2	KDBS - 3.2.5.3
4.1.2.3 Shall not be dependent on external software systems not part of the host operating system	3.7.1.2	KDBS - 3.2.5
4.1.2.4 Support creation and storage of Ada libraries in source form	3.7.1.2 3.7.2	KDBS - 3.2.5 3.3.1 3.3.4
4.1.2.5 Capability to define new categories of objects without imposing restrictions on computer information stored in objects	3.7.1.2	KDBS - 3.2.5.3 3.3.1
4.1.2.6 Provide flexible storage facilities to all MAPSE tools. Capability to read and write data base objects from within any MAPSE tool using data transfer and control functions and high level I/O function	3.7.1.2	KDBS - 3.2.5.6 3.3.4
4.1.2.7 Capability to create partitions	3.7.1.2	KDBS - 3.2.5.1 3.3.2

PHASE I SOW REQUIREMENTS	A - SPEC	B5 - SPEC
4.1.2.8 Capability to assign version qualifiers to objects or groups of objects. Time/Date and serial number. Capability to designate and use default version	3.7.1.2	KDBS - 3.2.5.4 3.3.1
4.1.2.9 Capability to create object attributes: History, Category and Access.	3.7.1.2	KDBS - 3.2.5 3.2.5.1 3.2.5.2 3.2.5.4 3.2.5.3 3.2.5.5 3.3.1 3.3.3
4.1.2.10 Capability to control access to data base objects using version qualifier, attributes, and partitions. "Programmable" access controls; provision for privileged user.	3.7.1.2	KDBS - 3.2.5.1 3.2.5.2 3.2.5.3 3.2.5.4 3.2.5.5 3.2.5.6 3.3.1 3.3.3
4.1.2.11 Capability to archive data base objects	3.7.1.2	KDBS - 3.2.5.7 3.2.5.8 3.3.6 3.3.7
4.1.2.12 Data base resources and operations as a result of this effort shall be available to Ada programmers	3.7.1.2	KDBS - 3.2.5 3.3
4.1.3 KAPSE INTERFACE REQUIREMENTS		
4.1.3.1 Specifiy virtual interface for KAPSE /MAPSE communication	3.1.5.2 3.7.2.1	KDBS - 3.2.4 3.2.5 3.3 3.2.4.1

PHASE I SOW REQUIREMENTS	A - SPEC	B5 - SPEC	
4.1.3.2 Virtual interface will provide user capability to invoke MAPSE tools, interact and exercise control over invoked tool	3.1.5.2 3.7.2.1	KFW KDBS	- 3.2.4.1 - 3.2.4
4.1.3.3 Virtual interface will have the capability to invoke any MAPSE tool from other MAPSE tool	3.1.5.2	KFW ACLI	- 3.2.4.1 - 3.1.3 3.2.4.2
4.1.3.4 Virtual interface will provide the capability for user LOGON/LOGOFF INITIATE/TERMINATE functions	3.1.5.3 3.7.2.1	KFW	- 3.3.2 3.2.5.2
4.1.3.5 Virtual interface will provide the capability to execute Ada programs	3.1.5.1 3.7.2.1	ACLI	- 3.1.1 3.2.4.1
4.1.3.6 User commands for job control and invoking tools shall have a uniform format	3.1.5.1 3.7.2.1	ACLI	- 3.2.5
4.1.3.7 User communication at command level will be possible in standard Ada character set	3.1.5.1	ACLI	- 3.1.1
4.1.3.8 Provide standard terminal interface specifications and functions to facilitate batch and interactive terminals. Specification will include protocols for synchronous user interactions and standards for implementing simple editing of the command line	3.1.5.3	KFW	- 3.3.1 3.3.9 3.4.1.2.8

PHASE I SOW REQUIREMENTS	A - SPEC	B5 - SPEC	
4.1.3.9 Specify host interfaces to support low-level I/O function and high level I/O package	3.7.1.1	KFW	- 3.2.5.8 3.2.5.9 3.3.5.10 3.3.9 3.3.10
4.1.3.10 Specify data identified as shared data and provide as standard interfaces	3.1.5.2 3.7.2	KDBS	- 3.2.5.6 3.3.4
4.1.4 KAPSE FUNCTIONS			
4.1.4.1 Provide basic Run-time support facilities	3.1.1.1	KDBS	- 3.2.5.6 3.3.4
4.1.4.2 Provide basic data transfer and control functions to support high level I/O package	3.1.1.1 3.7.1.1	KDBS	- 3.2.5.6 3.3.4
4.1.5 GENERAL MAPSE REQUIREMENTS			
4.1.5.1 Tools written in Ada and conform to standard interface specifications	3.7.2	Compiler Editor Debugger CM ACLI Linker	- 3.1.1 - 3.1.1 - 3.1.1 - 3.1.1 - 3.1.1 - 3.1.1
4.1.5.2 Inter-tool communication via virtual interface facilities	3.1.5.2 3.7.2	KDBS KFW ACLI	- 3.2.4 - 3.2.4.1 - 3.1.3

PHASE I SOW REQUIREMENTS	A - SPEC	B5 - SPEC
4.1.5.3 Formats for similar user commands shall be uniform and consistent across all tools	3.1.5.1	ACLI - 3.1.1
4.1.5.4 Data produced by one MAPSE tool needed or useful to another tool shalled be saved. Identify such data and provide interface specifications	3.1.5.2 3.7.2	ACLI - Appendix ACL Compiler - Appendix A Appendix C Appendix D Linker - 3.3.2.3 Appendix C
4.1.6 MAPSE Editor, includes the following capabilities: find, alter insert, delete, input, output, move copy, and substitute	3.7.2.5	Editor - 3.2.5 3.3
4.1.7 MAPSE Debugger		Debugger
4.1.7.1 Shall function at the Ada level	3.7.2.6	Debugger - 3.2.5
4.1.7.2 Shall support debugging of all Ada language features including concurrent programs	3.7.2.6	Debugger - 3.2.5 3.3.2 3.3.15
4.1.7.3 Shall provide linkage between executing program in binary form and corresponding source program	3.7.2.6	Compiler - Appendix C Debugger - 3.2.5 3.2.6 Linker - 3.2.5.3 Editor - 3.3

PHASE I SOW REQUIREMENTS	A - SPEC	B5 - SPEC
4.1.7.4 As a minimum shall provide: Breakpoints Display Values Modify Values Display and modifications of variables shal be machine or scalar type representations at the users option Display Subprogram arguments Modify flow of program Tracking Dumps	3.7.2.6	Debugger - 3.2.4
4.1.8 Compiler Requirements		Compiler
4.1.8.1 Operate in a modular fashion; minimize resource utilization	3.7.2.3	Compiler - 3.3
4.1.8.2 Operate in batch, remote batch, and on-line modes	3.7.2.3	Compiler - 3.2.5
4.1.8.3 Shall be easily rehosted and retargeted	3.7.2.3	Compiler - 3.2.5
4.1.8.4 Process Ada source and produce an efficient, equivalent program	3.7.2.3	Compiler - 3.2.5
4.1.8.4.1 Process the complete Ada language	3.7.2.3	Compiler - 3.2.5

PHASE I SOW REQUIREMENTS	A - SPEC	B5 - SPEC
4.1.8.4.2 Design pragmas to support requirements, design language pragmas	3.7.2.3	Compiler - 3.2.4
4.1.8.5 Produce all necessary outputs required to implement separate compilation and linking and execu tion produce output listings any or all of which can be user suppressed.	3.7.2.3 3.7.2.4	Compiler - 3.2.5 Appendix C Appendix D Appendix E
4.1.8.5.1 Produce symbol attribute listing	3.7.2.3	Compiler - 3.2.5 3.3.14 Appendix E
4.1.8.5.2 Produce symbol cross reference listing		Compiler - 3.2.5 3.3.14 Appendix E
4.1.8.5.3 Produce source listings		Compiler - 3.2.5 3.3.2 Appendix E
4.1.8.5.4 Produce object program listing	3.7.2.3	Compiler - 3.2.5 3.3.13 Appendix E
4.1.8.5.5 Collect, store, and output source program and compilation statistics	3.7.2.3	Compiler - 3.2.5 3.3.4 Appendix E

PHASE I SOW REQUIREMENTS	A - SPEC	B5 - SPEC
4.1.8.5.6 Produce environment listing	3.7.2.3	Compiler - 3.2.5 3.3.13 Appendix E
4.1.8.5.7 Produce system management listings	3.7.2.3	Compiler - 3.2.5 Appendix E
4.1.8.6 Shall perform extensive error checking. Errors shall be associated with the source line number	3.7.2.3	Compiler - 3.3 3.3.2 3.3.4 Appendix E
4.1.8.6.1 Severities of compiler errors shall include	3.7.2.3	Compiler - Appendix E
4.1.8.6.2 Error messages shall contain an error identifier, severity code, and a descriptive text	3.7.2.3	Compiler - Appendix E
4.1.8.6.3 The compilers shall detect 100% of syntax errors and all semantic errors, any capacity requirement that has been exceeded; list of all error messages generated shall appear in the Users Manual.	3.7.2.3	Compiler - 3.2.5 Appendix E
4.1.8.7 Optimization shall occur at the user's option via language pragmas. Optimization with respect to memory usage and execution speed shall be provided.	3.7.2.3	Compiler - 3.2.4 3.2.7 3.2.8 3.2.9 3.2.10 3.2.11 3.2.12

PHASE I SOW REQUIREMENTS	A - SPEC	B5 - SPEC
4.1.8.8 Shall process Ada source at a rate of 1000 statements per minute or faster	3.2.1	Compiler - 3.2.5
4.1.8.9 Goal shall be no arbitrary limitations; clearly identify any limitations on internal capacities	3.7.2.3	Compiler - 3.5
4.1.9 LINKING and LOADING REQUIREMENTS facilities shall adhere to rules and specifications contained in language manuals	3.7.2.4	Linker - 3.2.5
4.1.10 Ada Program Library as specified in language manuals	3.7.2.4	Compiler - Appendix D
4.1.11 Project/Configuration Management facilities	3.7.2.2	CM - 3.2.5 KDBS - 3.2.5 3.3.1 3.3.5
4.1.11.1 Must provide the following reports: Configuration Composition Report Attribute Report Partition Report Attribute Select Report	3.7.2.1 3.7.2.2	KDBS - 3.2.5 3.3.1 3.3.2 CM - 3.2.5 ACLI - Command Utilities
4.1.11.2 Summary reports based on combinations of attribute, partition, configuration, or version qualifier	3.7.2.1 3.7.2.2	KDBS - 3.2.5 - 3.3.1 - 3.3.2 - 3.3.5 CM - 3.2.5 ACLI - Command Utilities

PHASE I SOW REQUIREMENTS	A - SPEC	B5 - SPEC
4.1.11.3 MAPSE shall include a mechanism for automatic stub generation. MAPSE shall store source code and maintain pertinent information for the stub	3.7.2	Compiler - 3.3.15 Linker - 3.2.5 3.3.2.3
4.1.12 High level I/O will be an extension of or alternative to package specified in the Ada Reference Manual	3.7.1.2 3.7.1.1 3.7.1.1 3.5.1.3	KDBS - 3.2.5.6 - 3.3.4
4.1.13 Specify and include in design terminal interface routines for batch and online keyboard terminals required for Phase II	3.1.5.3	KFW - 3.3.1 - 3.3.9 - 3.4.1.2.8
4.1.14 Identify, specify and design any additional host dependent programs necessary to implement MAPSE on IBM and Interdata computers	3.7.1.1 3.1.5.3	KFW - 3.2.5 - 3.3

Volume 1

COMPUTER PROGRAM DEVELOPMENT SPECIFICATION

(TYPE B5)

COMPUTER PROGRAM CONFIGURATION ITEM

KAPSE Framework

Prepared for

Rome Air Development Center
Griffiss Air Force Base, NY 13441

Contract No. F30602-80-C-0292

14

TABLE OF CONTENTS

Vol 1
Page

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

17

SECTION 1 - SCOPE

1.1 IDENTIFICATION

This document presents the Computer Program Development Specification (Type B5) for the Computer Program Configuration Item (CPCI) calleo the Kernel Ada Programming Support Environment (KAPSE) Framework (KFW). This CPCI provides the Minimal Ada Programming Support Environment (MAPSE) interface to the host system.

The purpose of this specification is to define the KFW being designed as part of the Ada Integrated Environment contract for Rome Air Development Center (RADC). This document will serve to communicate the functional design decisions that have been adopted and to provide a basis for the detailed design and implementation phase.

This specification provides the performance, design, and testing requirements for the KFW. Section 3 presents the performance and design requirements. Section 4 presents the testing and quality assurance requirements. This specification, after approval by RADC, will serve as the development baseline for the KFW.

1.2 FUNCTIONAL SUMMARY

The KFW is designed to provide machine-independent process management and resource management functions to the MAPSE and to translate machine-oriented requests into machine-dependent calls.

The KFW provides the administration and control services that are necessary for the MAPSE to support the execution of multiple programs interacting with a shared data base. These services are presented as a canonical interface to a virtual operating system that uses the system facilities of a host execution domain.

The services are visible to the other MAPSE components through the KAPSE virtual interface, which enables the other components to be designed with a minimum knowledge of the host environment. The interface is designed to specify the functionality that is usually contained within an operating system. Therefore, when the host execution domain includes an operating system, the KFW services are derived from existing facilities to avoid duplication of or interference with those systems facilities.

The KFW interface is designed to comply with the requirements of the Ada language (such as tasks). In those instances where the language semantics are to be defined by implementation considerations, the KFW functionality is designed so that minimal constraints are imposed in exploiting the host execution domain. This results in the designed functionality being restricted when the host execution domain does not supply the underlying facility (such as multiprocessing).

SECTION 2 - APPLICABLE DOCUMENTS

2.1 PROGRAM DEFINITION DOCUMENTS

1. Requirements for Ada Programming Support Environment - STONEMAN, United States Department of Defense, February 1980.
2. Reference Manual for the Ada Programming Language, United States Department of Defense, July 1980.
3. Revised Statements of Work for Ada Integrated Environment, Rome Air Development Center, 26 March 1980.

2.2 INTER-SUBSYSTEM SPECIFICATIONS

4. Specification for the Ada Integrated Environment.
5. Volume 2, Computer Program Development Specification for CPCI KAPSE Data Base System.
6. Volume 3, Computer Program Development Specification for CPCI APSE Command Language Interpreter.
7. Volume 4, Computer Program Development Specification for CPCI MAPSE Configuration Management System.
8. Volume 5, Computer Program Development Specification for CPCI Ada Compiler.
9. Volume 6, Computer Program Development Specification for CPCI MAPSE Linker.
10. Volume 7, Computer Program Development Specification for CPCI MAPSE Editor.
11. Volume 8, Computer Program Development Specification for CPCI MAPSE Debugger.

2.3 MILITARY SPECIFICATIONS AND STANDARDS

12. MIL-STD-483, Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs, 1 June 1971.
13. MIL-STD-490, Specification Practices, 30 October 1968.

2.4 MISCELLANEOUS DOCUMENTS

14. Ada Support System Study (for the United Kingdom Ministry of Defence), Systems Designers Limited, Software Sciences Limited, 1979-1980.

15. Fisher, David A., Design Issues for Ada Program Support Environments, Science Applications Inc., SAI-81-289-WA, October 1980.

16. Ritchie, D. M., and K. Thompson, The UNIX Time-Sharing System, The Bell System Technical Journal, Vol. 57, No. 6, Part 2, July-August 1978.

17. Thompson, K., UNIX Implementation, The Bell System Technical Journal, Vol. 57, No. 6, Part 2, July-August 1978.

SECTION 3 - REQUIREMENTS

3.1 INTRODUCTION

This section presents the design and performance requirements of the KFW. The visible specifications for the KFW available to all MAPSE components are incorporated in the KAPSE virtual interface and are presented as an appendix to this specification. The MAPSE environment support to meet machine-independent portability design requirements, as specified in the SOW and STONEMAN have been restated in the System Specification (Type A) and are included here by reference.

3.1.1 General Description

The KFW presents the facilities through which the user accesses the host operating system. These facilities are embodied in a virtualization of operating system services that provides for resource management, process scheduling, and servicing of user requests. The view of the KFW presented to users and to the MAPSE Tool Set will be consistent from implementation to implementation. The KFW will also provide the translation of the user requests from the virtual system to the host system. The KFW may execute on a bare machine or under an existing operating system, depending upon the implementation. In each instance, the KFW must interface directly with the host to provide the support for the canonical interface that is visible to the portable MAPSE components through the KAPSE virtual interface.

A principal objective of the KFW design is to optimize the coexistence and integration of the MAPSE and the underlying operating system. The MAPSE user's awareness of the host environment should be minimal or nonexistent, but the MAPSE, through the KFW, should exploit existing facilities, where appropriate, to maintain the required efficiency.

3.1.2 Peripheral Equipment Identification

Standard terminal interface specifications and functions are provided through the KFW to facilitate the use of a variety of batch and interactive terminals and to ensure that machine-dependent interfaces do not affect the user. The KFW also provides the host interfaces required to support low-

22

level I/O functions and basic data transfer and control functions. All host dependent computer programs necessary to implement the MAPSE system on the IBM and Interdata computers specified for delivery of the system will be specified and implemented as part of the KFW. Although these initial hosts are both uniprocessors, considerable attention has been given to the design of the KFW control functions so as to permit efficient implementation on multiprocessors.

3.1.3 Interface Identification

The KFW interfaces directly with the KDBS, with the MAPSE tools and user programs through the KAPSE Interface Package and the Ada Tasking Package, and with the host machine.

3.1.4 Functional Identification

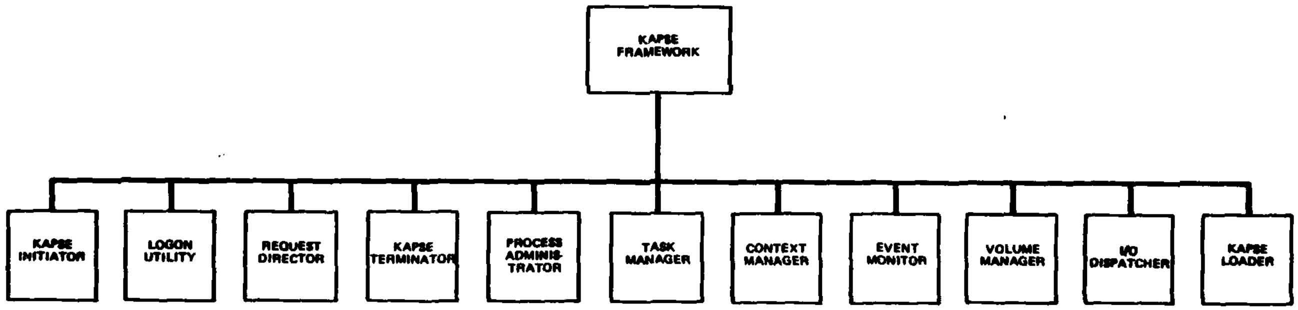
The major functional areas of the KFW are:

1. KAPSE Initiator
2. Logon Utility
3. Request Director
4. KAPSE Terminator
5. Process Administrator
6. Task Manager
7. Context Manager
8. Event Monitor
9. Volume Manager
10. Input/Output (I/O) Dispatcher
11. KAPSE Loader

The 11 functional domains are depicted in Figure 3-1.

3.2 FUNCTIONAL DESCRIPTION

This section describes the functions of the KFW, the program and equipment relationships and interfaces and the I/O utilization of the KFW.



TP No. 031-3009-B

Figure 3-1. KFW Functional Domains

The KFW provides the administration and control services that are necessary for the KAPSE to support the execution of Ada programs interacting with a shared data base. These services are presented as a virtual operating system interfacing with a host system.

The services are visible to the other MAPSE components through the KAPSE virtual interface. The KFW is designed to provide the functionality that is usually contained within an operating system.

The KFW is designed to provide a maximally machine-independent interface to host systems. Where host operating system features provide the functionality required by the KFW, the interface to those operating systems are minimal.

3.2.1 Equipment Description

The host systems with which the KFW must interface are the IBM VM/370 system and the Interdata 8/32 under the OS/32 operating system.

3.2.2 Computer Input/Output Utilization

The KFW design provides those facilities required by the MAPSE to communicate interactively with terminal and storage devices that are configured in the host hardware suite. See Figure 3-2.

The host hardware suite includes physical storage devices on which data may be recorded and subsequently retrieved. The KFW provides an interface to these devices as required to support those data base objects that have been designated as devices for manipulation by an Ada program. The KFW relies on the availability of device handlers in the host system so that the correspondence between a data base object and a device may be established and maintained in a manner consistent with that of a data base object and a file.

When console or terminal communication devices are configured, the host system facilities for handling communication devices are used by the KFW to implement an interface that is responsive to the needs of all MAPSE tools that may establish a dialogue with a user.

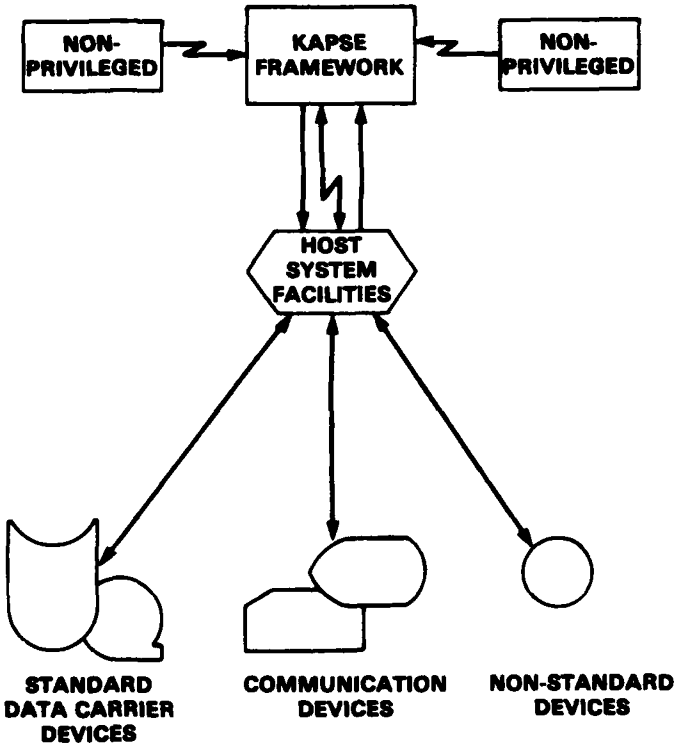


Figure 3-2. Computer Input/Output Utilization

A consistent user communication interface to the MAPSE requires that the KFW incorporate in its design a stanuard line-editing protocol for console or terminal input. Host system facilities, while providing services for reading and writing characters to communication devices, are unlikely to conform to this protocol. Therefore the system facilities must permit the KFW to implement the necessary functionality to support the editing of input characters without interference. A critical requirement is that the defined MAPSE breakin or attention signal be discernible by the KFW so that a user may be connected initially to the Logon Utility or may terminate a current execution state in the MAPSE.

When noninteractive communication devices are configured, such as a card or paper tape device, the KFW is designed to provide conventional batch operation by directing the device to the APSE Commanu Language Interpreter. Again the host system facilities for handling these devices are useu by the KFW.

The KFW is designeu to support a variety of nonstandard input and output requirements. These requirements result directly from an Ada program anu from the KFW itself.

Through the KFW, an Ada program is provided the functionality to connect to a device that is not in the prescribed host haraware suite. In this instance, the host system facilities must enable the KFW to have control of the I/O channel for the device so that the KFW may receive and send instructions or data from the Aua program to the device or device controller.

Other nonstanuard inputs required by an Aua program are specific entry interrupts and clock data. The KFW is designed to field the interrupts and read the clock through the host system facilities. Similar interrupt and timer services are required by the KFW in order to detect the termination of asynchronous events that it may have initiated in the host environment. For example, the completion of a MAPSE I/O operation is recognized by its termination interrupt being made available to the KFW through the host system facilities.

3.2.3 Computer Interface Block Diagram

Figure 3-3 identifies the interface points between the KFW, the MAPSE components, and the host system.

3.2.4 Program Interfaces

This paragraph identifies the KFW interfaces and their purposes. The KFW interfaces through the KAPSE virtual interface to the MAPSE tool set and Ada user programs, to the KDBS at the kernel level, and to the host system. Figure 3-4 represents these interfaces.

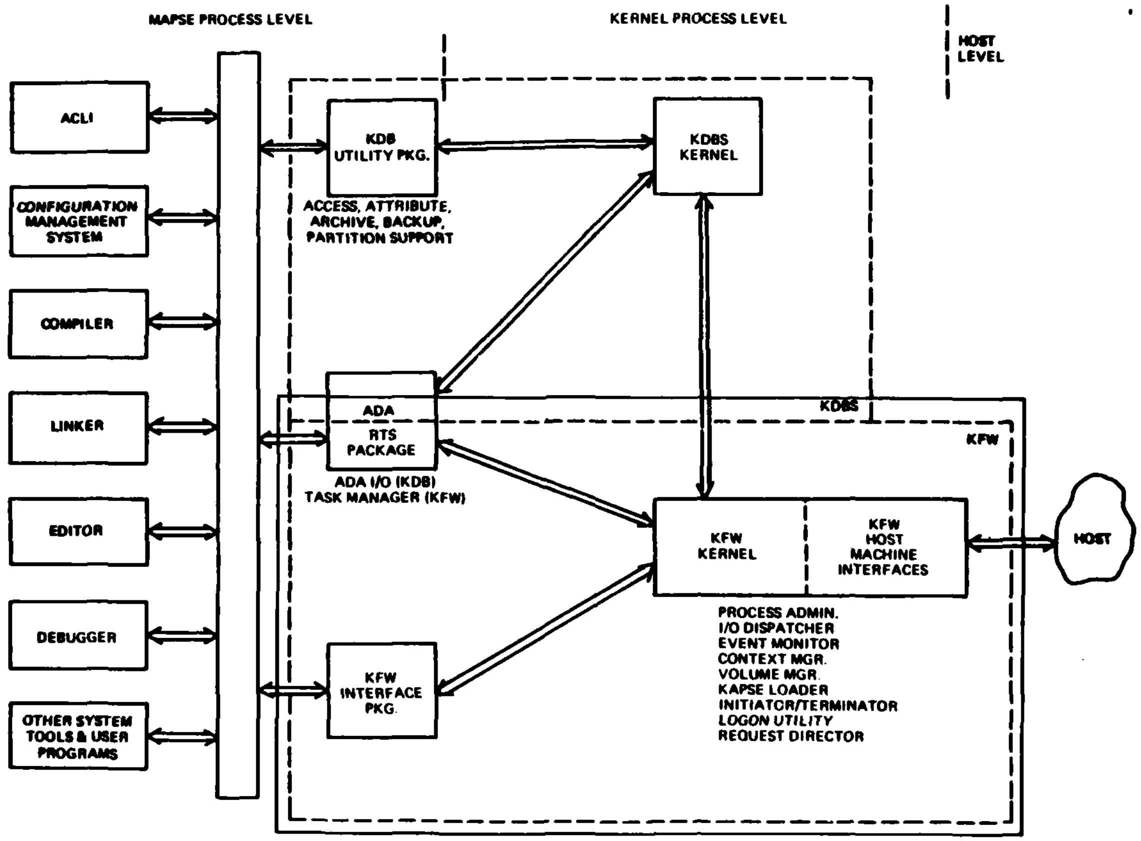
Figure 3-5 depicts the six primary interfaces provided by the KFW in the KVI. For each interface its calling and called states are identified, such as, Process-to-Kernel, Kernel-to-Kernel and Process-to-Process. The latter two modes do not require the use of the Request Director interface and may be performed through Ada subprogram calls. In those instances where an interface has multiple modes, as does the Process Administrator, the interface is provided to accommodate each mode through multiple packages with identical visible specifications.

Appendix A contains the Ada package specifications for the MAPSE to KFW interfaces. In addition, where appropriate, Ada package specifications for the major data types used by the interfaces are included.

3.2.4.1 KAPSE Virtual Interface

Specifications of the services the KFW provides to the MAPSE tool set and Ada user programs are encapsulated into the KFW Interface Package and the Ada Run-time Support Package.

Ada user programs and MAPSE tools execute in a nonprivileged execution state. The KFW considers these execution domains to be MAPSE processes. The functional domains of the KFW and KDBS that execute in a privileged execution state constitute the Kernel. In order to support the logical distinction between a MAPSE process and the Kernel Process, the KFW supplies an interface that enables a MAPSE process to request a service provided by the Kernel Process. Any KDBS or KFW Kernel service that is requested by a MAPSE process is connected to the Request Director in the Kernel for the



TP NO. 021-2002-A

Figure 3-3. MAPSE Interface Block Diagram

29

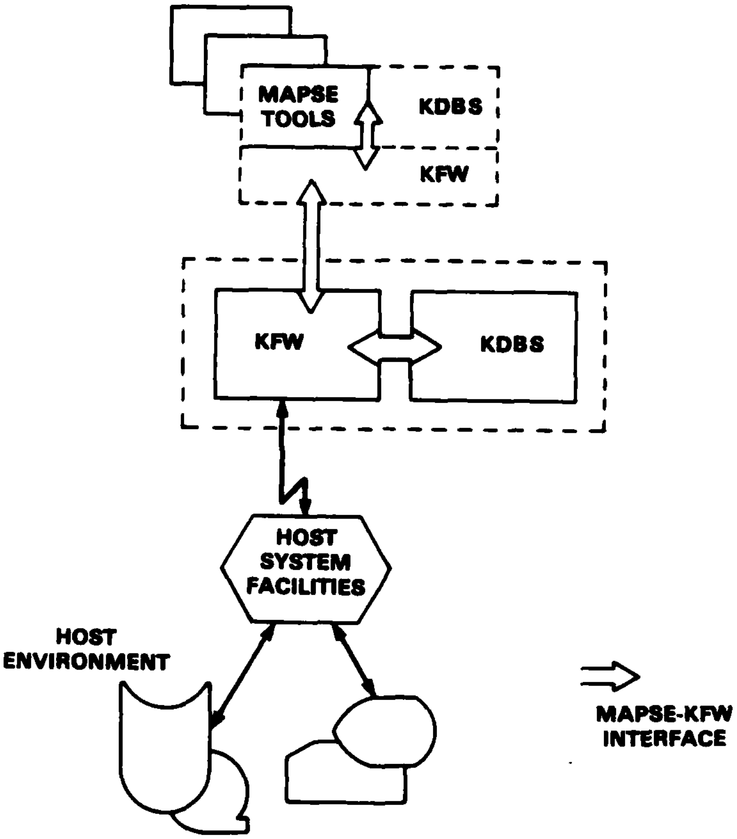


Figure 3-4. KFW Program Interfaces

MAPSE-KFW INTERFACE STATES	INTER-MAPSE	INTRA-PROCESS	INTRA-KERNEL
CONTEXT MANAGER	X		X
EVENT MONITOR	X		X
I/O DISPATCHER	X		X
PROCESS ADMINISTRATOR	X		X
TASK MANAGER		X	
VOLUME MANAGER			X

Figure 3-5. KFW Interface States

service to be recognized and routed to the appropriate logical domain within the Kernel. The nature of the interface to the Request Director depends upon the host system facility available for communication between executing processes.

The interfaces supplied by the KFW are oriented to specific features of the Ada language. Figure 3-6 itemizes these features and shows the functional domains that provide the interfaces used to satisfy them. In the case of the standard I/O feature, the feature also requires interfaces that are supplied by the KDBS.

Certain facilities provided by a KFW logical domain are designated as critical facilities. Critical facilities are those facilities that perform operations which, if misused, may result in unpredictable execution states. A design requirement of the KFW interfaces is to organize its package specifications so that the misuse of these facilities is minimized. A means of accomplishing this requirement is achieved by judicious use of the KDB access control of Ada library objects in conjunction with the separation of critical facilities.

3.2.4.2 KDBS Interface

The portability of the KDBS is achieved through the KDBS interface provided by the KFW in the Kernel. This interface presents to the KDBS a straightforward, convenient abstraction upon which to specify the storage and retrieval of information. The abstraction and its accompanying operations are designed to be compliant with host system facilities that are generally available. The interface is implementable in terms of any underlying host file management system or device handling packages. The interface insulates the KDBS in particular from the nature of the device on which the abstraction is mapped. In the instance of an interactive communication device, the KFW provides the terminal handler to refine the transmission of characters, unless precluded by the host. If this occurs the host terminal handler is enhanced to meet the KDBS interface specification.

MAPSE-KFW INTERFACE SUPPORT OF ADA FEATURES	CONTEXT MANAGER	EVENT MONITOR	I/O DISPATCHER	PROCESS ADMIN.	TASK MANAGER	VOLUME MANAGER
MAIN PROGRAM EXECUTION	X			X		
TASKS	X			X	X	
ENTRY INTERRUPTS		X				
ACCESS ALLOCATORS	X					
STANDARD I/O		X	X			X
LOW LEVEL I/O		X	X			

Figure 3-6. Ada Feature Interface

3.2.4.3 Host System Interface

The host system interfaces of the KFW provide the MAPSE direct communication to the host environment. The nature of these interfaces determines the functional complexity of the KFW.

For the two initial host systems the nature of these interfaces is significantly different. This requires that the KFW design be adaptable to both the low level machine interface of VM/370 and to the conventional multiprogramming interface of OS/32. The low level style of interface facilitates the exploitation of the base computer architecture in realizing the potential of the KFW design. The multiprogramming style of interface requires that the KFW use the services of the system software. As a consequence, host software performance characteristics are projected into the MAPSE. In those instances that result in unacceptable performance, the host system interfaces may be tuned to an improved level of capability.

3.2.5 Function Description

The main function of the KFW is to provide the administrative and control services that are necessary for the KAPSE to support the execution of Ada programs interacting with a shared data base. Thus the KFW provides the services of a logical operating system to map the MAPSE onto various host systems.

The KFW consists of the components identified in Paragraph 3.1.4. A schematic that informally shows the major functional interfaces provided and employed by the KFW is shown in Figure 3-7. The schematic omits the functionality of the Request Director because it is assumed, where required, to be an implicit property of each functional interface.

The components of the KFW that execute at the Kernel process level are depicted in the schematic of Figure 3-7. These components provide the essential facilities for controlling and servicing multiple MAPSE processes and for sending and receiving requests to and from the host environment. In addition, facilities are included to startup and shutdown the execution of the KAPSE.

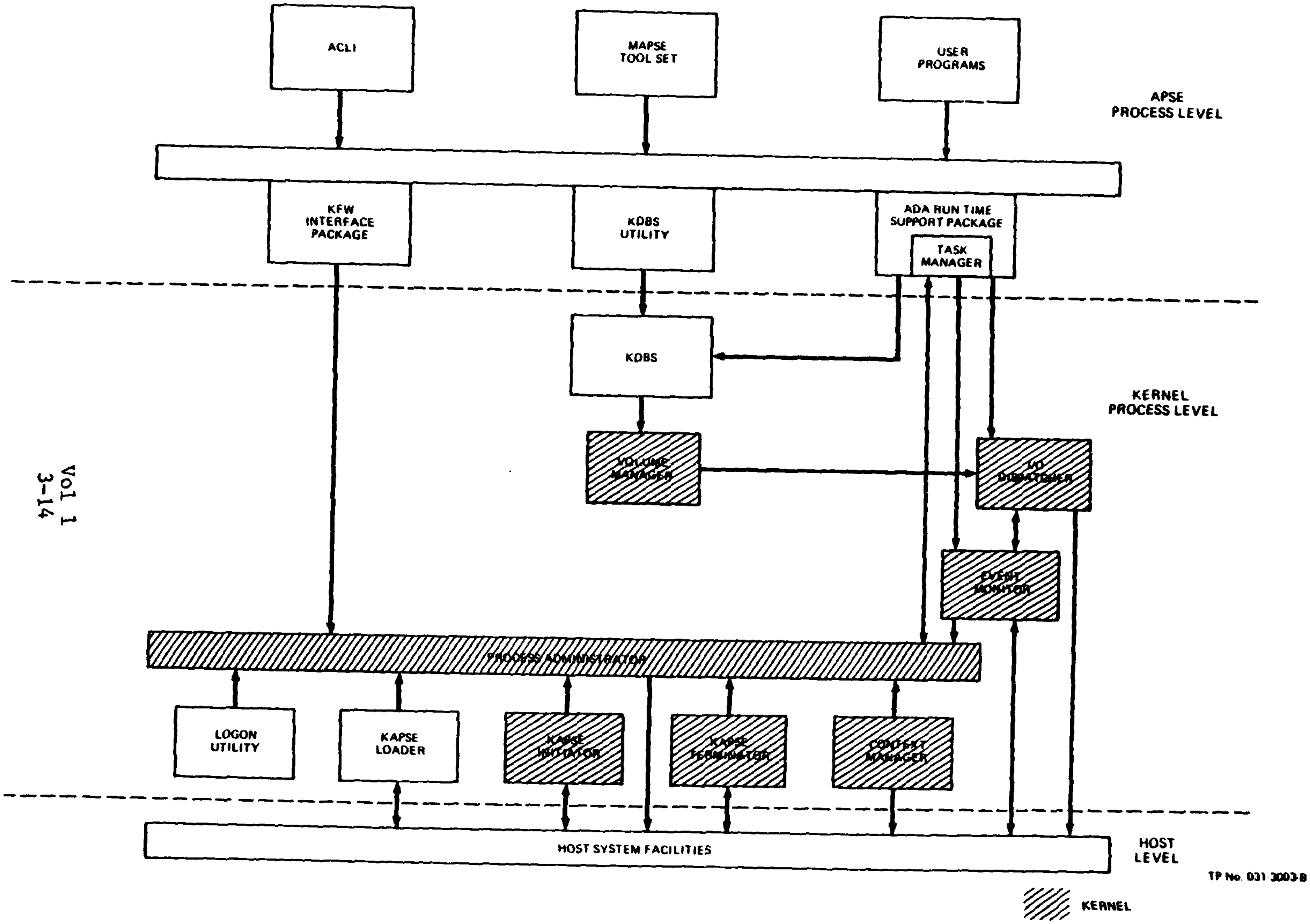


Figure 3-7. KFW Kernel Interfaces

In those instances where a KFW component presents an interface through the virtual interface, the KFW Kernel Process maps the portable virtual interface functionality into one or more host dependent system facilities. The schematic of Figure 3-8 identifies the components that provide this mapping. The bold arrowed lines entering the hatched KFW Kernel component denote the portable KFW interfaces in the virtual interface, and the arrowed lines exiting denote the results of the functional mapping to the host facilities. When the host system facility is a bare machine, the mapping is isomorphic and the KFW Kernel process becomes the host operating system.

All MAPSE processes are created through an instantiation of the KAPSE Loader executing as a process under the control of the host system facilities.

Initially, the KAPSE Loader is instantiated to load the Logon Utility. The Logon Utility is then executed as a MAPSE process. The schematic in Figure 3-9 shows the three instantiations of the KAPSE Loader required to establish the execution domains for the Logon Utility, the APSE Command Language Interpreter (ACLI) and the MAPSE tool to run as MAPSE processes. A consequence of the Logon Utility executing as a MAPSE process is that by definition, it becomes the parent of all MAPSE processes and relies primarily on the portable interfaces of the KFW Kernel Process. The KAPSE Loader, however, is dependent upon the direct use of host system facilities.

The Task Manager is the only KFW functional domain that resides in the Shared Execution Domain of the MAPSE. This functional domain is used by any MAPSE process enclosing Ada tasks and executes as a part of the MAPSE process. The schematic in Figure 3-10 shows two MAPSE processes that have enclosed task objects.

Only the portable interfaces of the KFW Kernel process are used by this functional domain which is thereby insulated from the host system facilities. Through the Kernel process facilities, different executions of a MAPSE process are initiated in the host environment for each enclosed task object.

The next 11 paragraphs describe the individual components of the KFW.

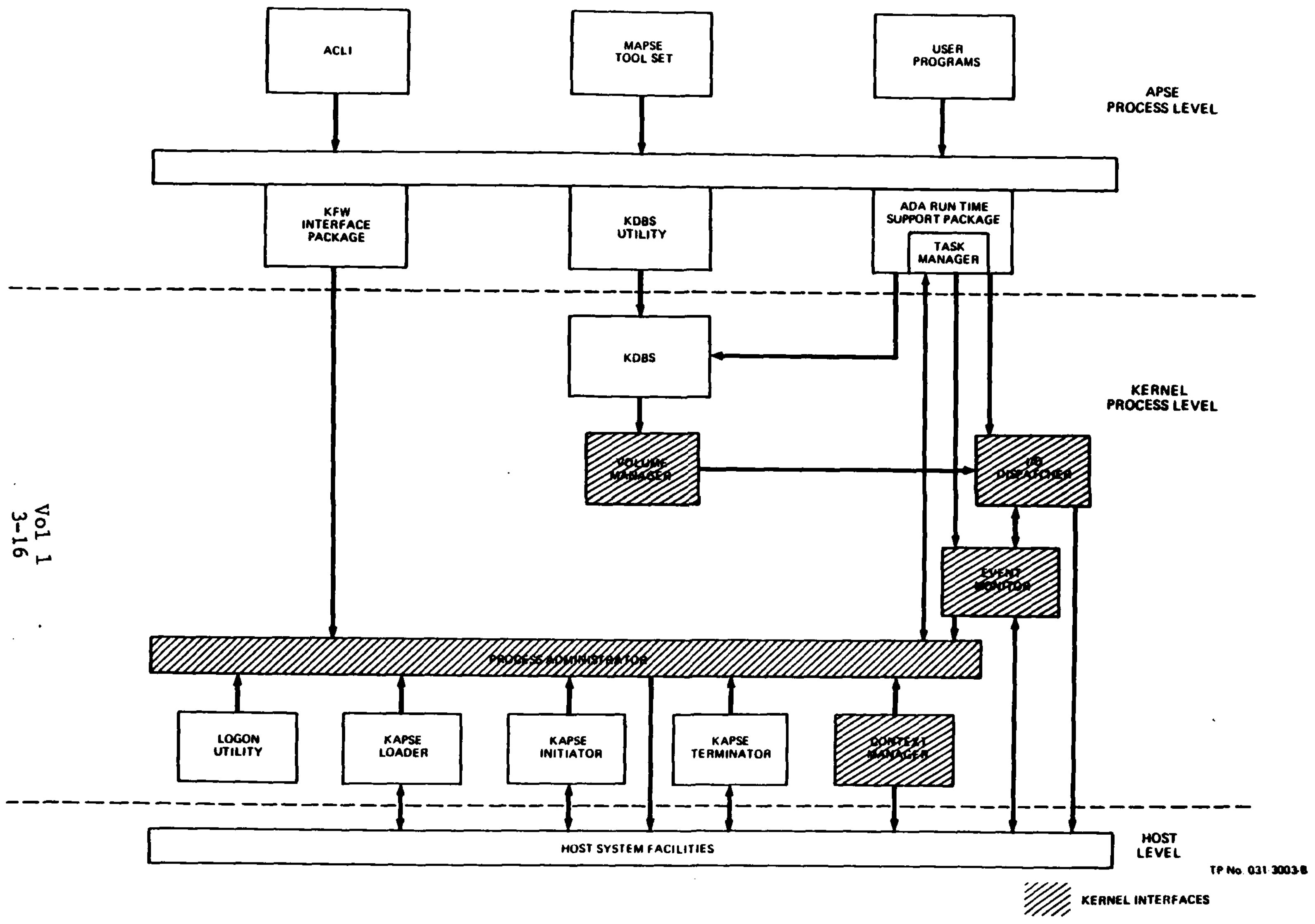


Figure 3-8. KFW Virtual Interfaces

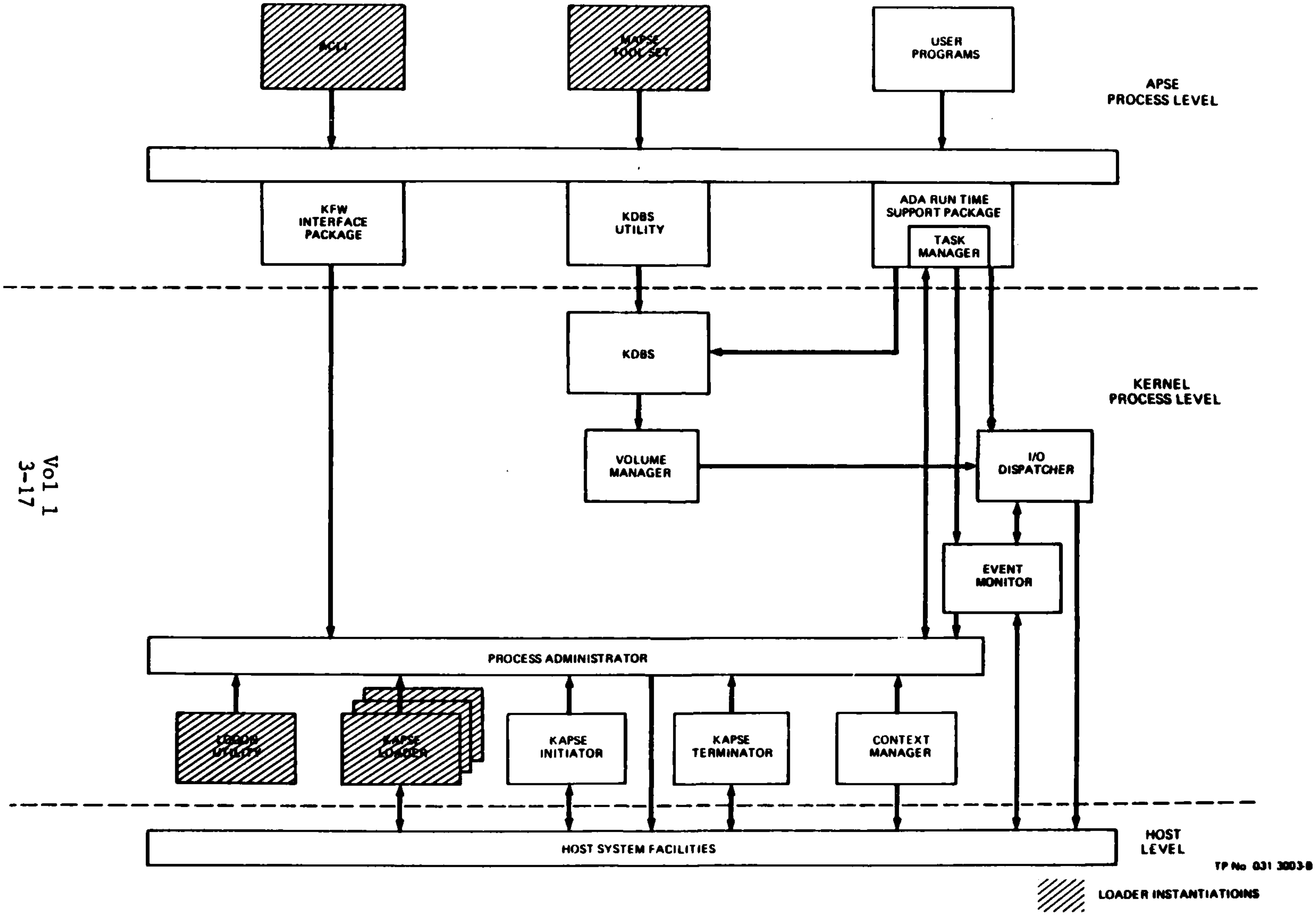


Figure 3-9. KAPSE Loader Instantiations

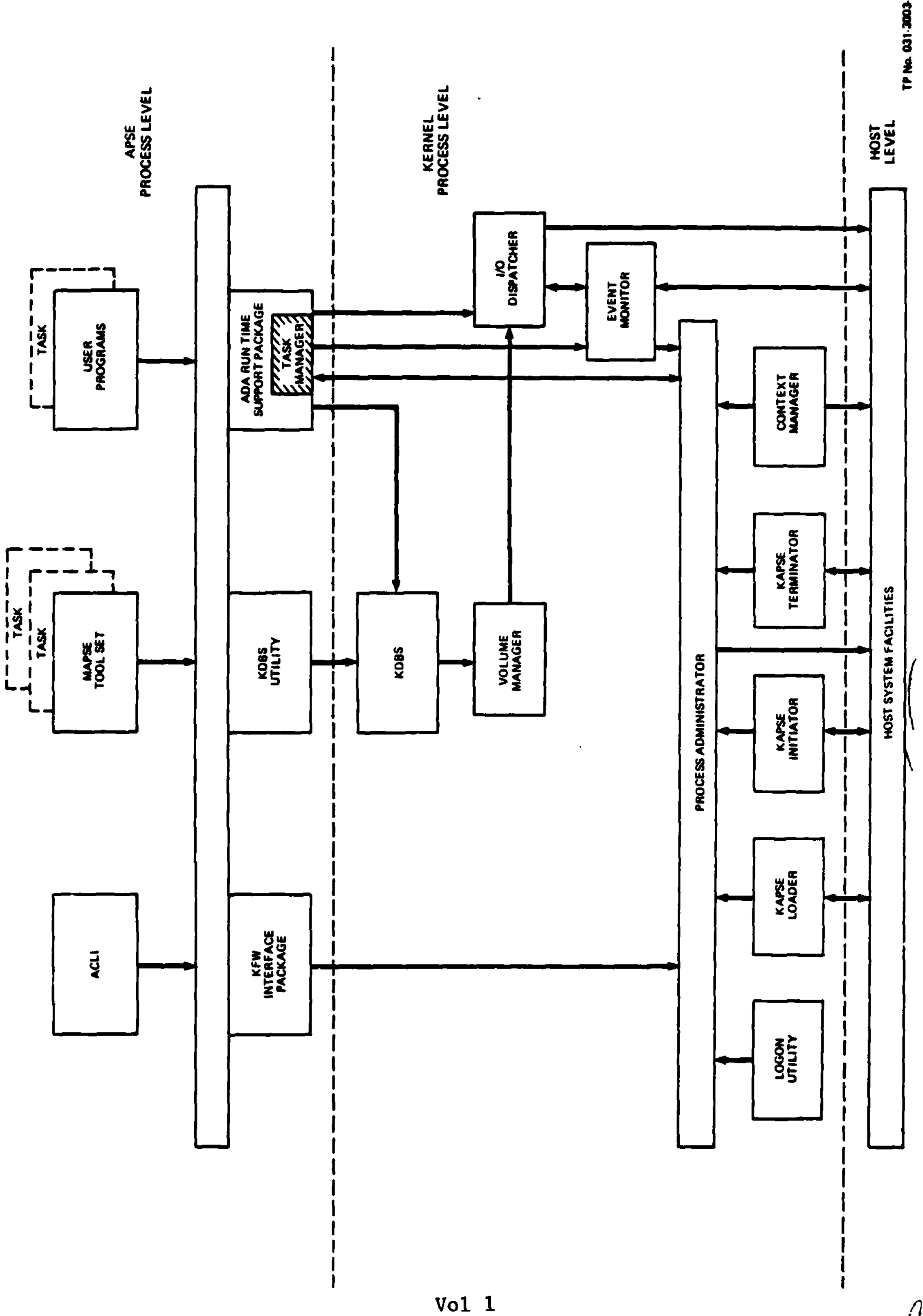


Figure 3-10. MAPSE Enclosed Task Objects

3.2.5.1 KAPSE Initiator

The purpose of the KAPSE Initiator is to establish the initial execution environment for the KAPSE once the Kernel process has been loaded in the host environment.

Upon establishing an instantiation of the KAPSE, the host system initially passes control to the KAPSE Initiator. Included as a part of this preparation is the allocation and loading of the Shared Execution Domain and the acquisition of the Dynamic Address Domain. Prior to relinquishing control, the KAPSE Initiator starts the Logon Utility to make the MAPSE available for user access.

3.2.5.2 Logon Utility

The purpose of the Logon Utility is to await input activity on the batch and interactive communication devices configured for MAPSE use.

The Logon Utility performs the prescribed Logon protocol, including user authentication. A process request is then issued to start execution of the ACLI. When the ACLI completes execution, the Logon Utility is reactivated and makes the device available for the next user.

3.2.5.3 Request Director

The purpose of the Request Director is to route requests for Kernel process level facilities from a MAPSE process to the appropriate KFW or KDBS component.

The Request Director implicitly handles all such requests for kernel level facilities.

3.2.5.4 KAPSE Terminator

The purpose of the KAPSE Terminator is to accomplish the orderly shutdown of the MAPSE.

The KAPSE Terminator terminates all MAPSE processes and disables each communication device to the MAPSE to prevent further user interaction. When the shutdown state is achieved, the KAPSE Terminator initiates the prescribed MAPSE cleanup processes to perform data base backup. The KAPSE Terminator releases the resources acquired by the Kernel Process and relinquishes control to the host.

3.2.5.5 Process Administrator

The Process Administrator controls the executions of logically concurrent MAPSE processes. The KFW Interface Package provides a portable interface from the MAPSE tool set and Ada user programs to the Process Administrator. This interface provides a consistent methodology for supporting the MAPSE loosely coupled process execution structure and the requirements of Ada tasks. A separate address domain is defined for each MAPSE process. Within this domain the Process Administrator schedules the various executions of the MAPSE process on the basis of the task control information maintained by the Task Manager. As a result, the execution of tasks from various MAPSE processes are interleaved while retaining the intraprocess execution sequence mandated by the task control information. Once the Process Administrator has scheduled a process for execution, the process is considered to be logically active because actual execution may be delayed by the host environment.

3.2.5.6 Task Manager

The Task Manager synchronizes the concurrent executions of a MAPSE process in conformance with the intertask communication performed by tasks within the process. The Task Manager executes within the execution domain of each MAPSE level process. The Task Manager is responsible for establishing, in conjunction with the Process Administrator, the execution domains required to support Ada tasking. In order to exploit the facilities of the host system, the Task Manager relies on the Process Administrator to schedule tasks for execution when a change in the tasking control within a process is required. The Process Administrator may schedule one or more tasks, depending on the number of tasks that are ready to be executed, the number of processes currently active, and the capabilities of the host system facilities.

3.2.5.7 Context Manager

The purpose of the Context Manager is to control access and use of the Dynamic Address Domain and Shared Execution Domain in the MAPSE.

The Context Manager is provided to change the address domain of an executing process. The domains are established by the Context Manager using the host system facilities that support storage space management for a dynamic execution environment.

The Dynamic Address Domain is used to enable a process to change its address domain as defined by the process context map. The Shared Execution Domain is used to build the MAPSE Run-time System that permits the shared execution of the KFW Task Manager and the KDBS I/O Support Package.

3.2.5.8 Event Monitor

The Event Monitor receives, identifies, and traps requested interrupts from the host environment that are made available to the Kernel process.

The Event Monitor, in conjunction with the Process Administrator, schedules both MAPSE and kernel level processes to respond to these traps and interrupts.

3.2.5.9 Volume Manager

The Volume Manager transfers data between the logical data base maintained by the KDBS and the logical and physical data devices.

Using the most appropriate features provided by the host system facilities, logical and physical data devices are manipulated to store and retrieve information. The Volume Manager, although dependent on the host system facilities, does not communicate directly with them but uses the facilities afforded by the I/O Dispatcher. A single request to the Volume Manager may result in one or more requests to manipulate the associated data device. In these instances, the requests may be chained together and forwarded to the I/O Dispatcher.

3.2.5.10 I/O Dispatcher

The purpose of the I/O Dispatcher is to synchronize data transfer requests that have originated from concurrently executing MAPSE processes.

The I/O Dispatcher provides a portable interface through the virtual interface that is used by the Volume Manager and MAPSE processes to initiate I/O operations to data devices configured in the host environment. The I/O

Dispatcher uses the facilities of the Event Monitor to recognize the completion of all operations it initiates. When necessary, the process originating the request is suspended through the Process Administrator.

3.2.5.11 KAPSE Loader

The purpose of the KAPSE Loader is to load a process for execution. The KAPSE Loader uses the host system facilities to retrieve and load a process that can execute under the control of the host environment. Once executing, the process becomes a MAPSE process by registering itself through the Process Administrator interface.

3.3 DETAILED FUNCTIONAL REQUIREMENTS

3.3.1 KAPSE Initiator

The KAPSE Initiator is the component within the Kernel that receives control when the Kernel process is loaded for execution in the host environment. The KAPSE Initiator provides no facilities to other MAPSE components. .

3.3.1.1 Inputs

There are no input arguments defined for Initiator.

3.3.1.2 Processing

The KAPSE Initiator is designed to prepare the KAPSE for process execution. The KAPSE Initiator uses environment system parameters to create the Dynamic Address Domain and the Shared Execution Domain. The KDBS and KFW packages that are to be executed as an extension of a MAPSE process are placed in the Shared Execution Domain. The batch and interactive device definitions that are available for user communication with the MAPSE are derived, and the user communication device table is formatted for subsequent use by the Logon Utility. Once the KAPSE data base is made available the prescribed MAPSE startup processes are begun and the KAPSE Initiator awaits their completion.

When the execution environment is ready, the Logon Utility is called through the Process Administrator to respond to user access from the user communication devices in the user communication device table and the KAPSE Initiator completes its execution.

3.3.1.3 Outputs

There are no output arguments defined for Initiator.

3.3.2 Logon Utility

The Logon Utility is called by the KAPSE Initiator to allow the MAPSE to be accessed through the user communication devices specified in the communication device table. The Logon Utility provides no facilities to other MAPSE components. It is designed to start execution of a MAPSE process for an authorized user.

3.3.2.1 Inputs

Upon receiving an input from a device the Logon Utility executes the authentication protocol that supplies the necessary data to identify and validate a user. In addition, sufficient information is extracted from the data to determine which MAPSE process is to be started for the user. Normally this process is an instantiation of the ACLI. (see Figure 3-11)

3.3.2.2 Processing

The Logon Utility derives the data base object name for the device table entry, starts an ACLI process for execution, and passes the object name as the standard input file to the process. The Logon Utility then awaits input from another device or for a previously started process to attain a finished or terminated state. In the latter case, the Process Control Block is deleted and the device table entry is released for a new user or the next job.

3.3.2.3 Outputs

When a MAPSE shutdown has been started, inputs from interactive user communication devices prompt the Logon Utility to display the shutdown greeting.

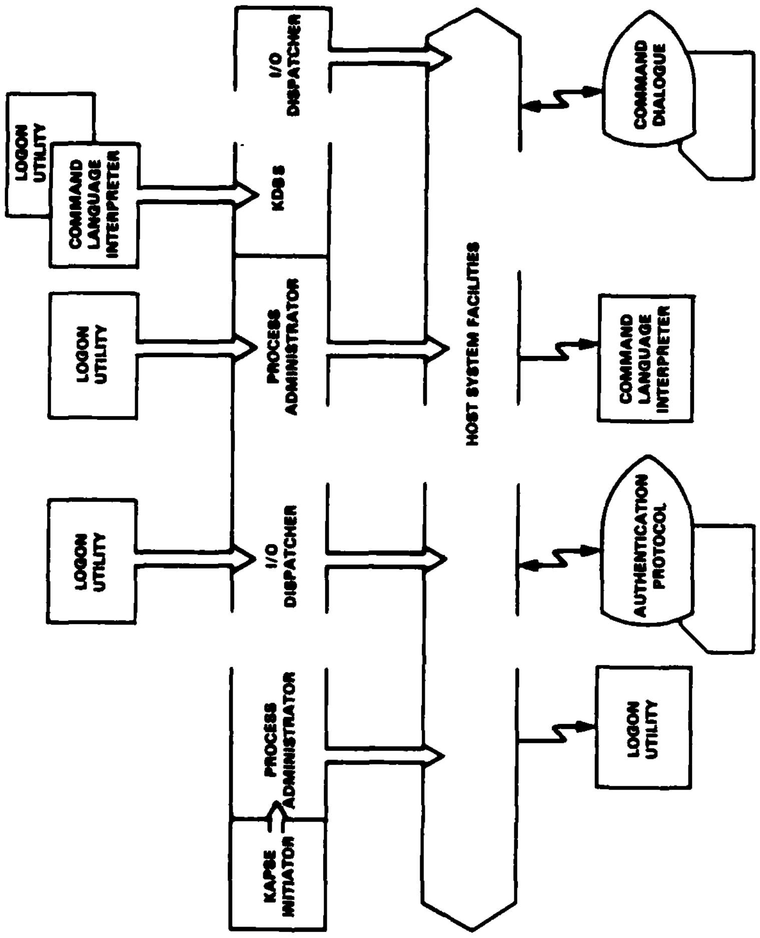


Figure 3-11. ACLI Instantiation

3.3.3 Request Director

The Request Director is the functional facility through which a MAPSE process requests a facility provided in the Kernel process. Appendix A includes the specification of the Ada package REQUEST_DIRECTOR that is used by those virtual interface packages that define an interface to the Kernel process. See Figure 3-12 for a logical breakdown of the Request Director.

From the Request Parameter List that is made available upon initiation, the Request Director calls the appropriate functional domain to service the request.

The parameter list is constructed by the Kernel process request in the MAPSE process to include the kind of request and its actual parameters. The Request_Kernel facility is then called to save the execution context and parameter list address in the task control block. When this execution of the MAPSE process is continued, the Request_Kernel facility returns to the Kernel process request in order to update its actual parameters.

The following example demonstrates the use of the Request_Kernel interface by an Ada package.

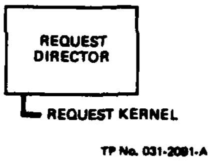


Figure 3-12. Logical Breakdown

```
with REQUEST_DIRECTOR; use REQUEST_DIRECTOR;
package body SOME_KVI_PACKAGE is

    procedure Some_Facility
         (Param_1:         SOME_TYPE;
          Param_2: in out SOME_TYPE;
          Param_3: out     SOME_TYPE) is

    RPL: REF_REQUEST_SHAPE := new REQUEST_SHAPE(Some_Facility);
    procedure This_Request is
         new Request_Kernel (REQUEST_SHAPE (Some_Facility)),

    REQUEST_EXCEPTION: exception;

    begin
         --Save in and in out parameters in RPL

    This_Request (RPL);

         --Restore in out and save out parameters from RPL

    exception
         when REQUEST_EXCEPTION =>

         --Handle Kernel exception made available in TCB

    end;

end SOME_KVI_PACKAGE;
```

The Request Director is initiated to route the specified request to the appropriate component in the Kernel. Request__Kernel is called by all virtual interfaces to the Kernel.

3.3.3.1 Inputs

The following input argument is defined for Request_Kernel:

Addr_RPL - The address of the Request Parameter List.

3.3.3.2 Processing

The address of the parameter list is entered in the task control block for the task of the MAPSE process that requested the Kernel facility. The control block is updated to save the current context of this process execution. The host system facility is initiated to start execution of the Request Director and to make available to it the control block address of the requesting task. When this execution of the process is continued, if an exception occurred during the processing of the request, the Kernel exception name that is made available in the block is raised.

3.3.3.3 Outputs

There are no outputs defined for the Request_Kernel.

3.3.4 KAPSE Terminator

The KAPSE Terminator is the component within the Kernel that is called to perform an orderly closure of the MAPSE. It is designed to prepare the MAPSE for shutdown and to terminate execution of the KAPSE in the host environment.

3.3.4.1 Inputs

There are no input arguments defined for the Terminator.

3.3.4.2 Processing

The KAPSE Terminator waits all current MAPSE processes except the Logon Utility through the Process Administrator and marks each entry in the user communication device table as unavailable. Once all MAPSE processes have achieved the wait state, they are terminated and deleted. When the execution environment is idle, the prescribed MAPSE shutdown processes are begun, and the KAPSE Terminator waits for their completion.

Upon completion, the Logon Utility is terminated and deleted. The acquired resources are released to the host environment and the Kernel process requests self-termination through the host system facilities.

3.3.4.3 Outputs

There are no output arguments defined for the Terminator.

3.3.5 Process Administrator

The Process Administrator functionally encapsulates a set of operations on the data structure defined as the process control block. Appendix A includes the specification of the Ada package PROCESS_ADMSTR that is made available in the virtual interface. See Figure 3-13 for a logical breakdown of the Process Administrator.

The fundamental executable entity within the MAPSE is defined as a process. A process results from the compiling and linking of an Ada main program and the subsequent loading for execution of its Load Object. The MAPSE is designed to support the logically concurrent execution of multiple processes through the services of the Process Administrator.

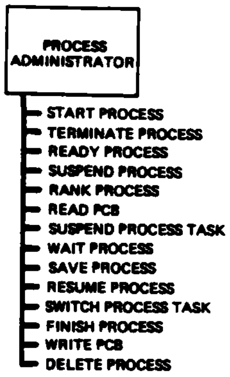
The execution domain of the MAPSE consists of the Kernel process and one or more MAPSE processes. The Kernel process executes in a privileged execution state while the MAPSE processes execute in a nonprivileged execution state. The Process Administrator is the part of the Kernel process that is designed to coordinate and schedule the MAPSE execution domain. The host system facilities are used by the Process Administrator where necessary to ensure the efficient, economic execution of a process in the host environment.

A MAPSE process may invoke the execution of another MAPSE process through the Process Administrator. After invocation, the calling and called processes are candidates for execution. Parameters may be passed between the calling and called process.

MAPSE processes are organized into a tree, where each process is a child process of the process that created it. Processes invoked through the Logon Utility are considered to be children of the Process Administrator. A process is permitted to terminate, suspend, or resume only itself or its descendent processes. The children of a terminated process are inherited by their grandparent.

A consequence of the Ada task semantics is for a MAPSE process to synchronize the execution of different tasks within the Load Object. The Process Administrator recognizes this requirement by maintaining the scheduling of a MAPSE process to be consistent with the task synchronization specified within the MAPSE process.

52



TP No. 031-2083-A

Figure 3-13. Logical Breakdown

The Process Administrator is designed to facilitate the physical parallel execution of processes where the host system facilities support multiprocessors in the host hardware suite. If such facilities are not available, the Process Administrator implements logically parallel (interleaved) execution of processes.

In host environments that provide system facilities precluding the Process Administrator from assuming direct control over the scheduling of process execution, the Process Administrator relinquishes final scheduling of process execution to the host environment.

The Process Administrator initiates a process by calling the KAPSE Loader through the host system facilities and making available to it the name of the Load Object and the process control block address. The KAPSE Loader communicates with the host system to read the Load Object file. The Process Administrator maintains a record of all control blocks in the Process Dictionary. For each MAPSE process that is started by the Process Administrator, a process control block is created. Instantiation of this block occurs for each separate thread of process execution control through the creation of a task control block. Each activated Ada task object results in the creation of a task control block that references the process control block of the enclosing process (Ada Main program). Consequently the former is an instantiation of the latter and identifies a unique name for each thread of control. When a process contains only a single thread of control, a single instantiation of the process block exists and is defined by the process task block created during process initiation. The Process Dictionary that is maintained through the process blocks retains the status of all registered MAPSE processes. (See Figure 3-14)

Upon expiration of a standard quantum of time, the Event Monitor calls the Process Administrator to service the Active Process List. For each active process, the Process Administrator computes the processing time provided to the process during the standard quantum of time. When this processing time has exceeded the prescribed limit, the process execution is suspended and its instance of the process control block is entered into the Process Ready Queue. When the process is executing under the control of the host

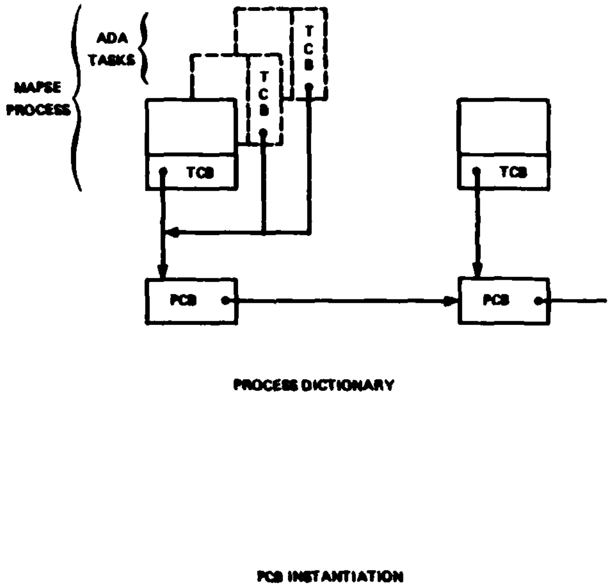


Figure 3-14. PCB Instantiation

environment, the Process Administrator does not maintain the execution context for the suspended process but relies on the host system facilities.

3.3.5.1 Start Process

This facility establishes a process execution domain in which the named Load Object can be loaded. Start_Process is requested when a MAPSE process requires that a new process be invoked.

3.3.5.1.1 Inputs

The following input arguments are defined for Start_Process:

Load_Object_Name - The name of the Load Object.
Process_Param - The actual parameters for the process.
Process_Priority - The process scheduling priority.
Process_Status - One of two values ("suspend" or "ready").

3.3.5.1.2 Processing

Start_Process creates a process control block for the new process. This control block is inserted in the Process Dictionary. The KAPSE Loader is called and passes the block address and referenced Load Object. If Process Status is "suspend" the process is created in a suspended state; otherwise, the process may be immediately scheduled for execution.

3.3.5.1.3 Outputs

The following output argument is defined for Start_Process:

Addr_PCB - The process control block address of the new process.

3.3.5.2 Terminate Process

This facility terminates all execution of the specified MAPSE process. Terminate_Process may be requested to terminate the requesting process or any process started by the requesting process.

3.3.5.2.1 Inputs

The following input argument is defined for Terminate_Process:

addr_PCB - The process control block address of the process to be terminated.

3.3.5.2.2 Processing

Terminate_Process validates the specified control block. The status of the specified block is changed to terminated in the Process Dictionary and all instances of the block are removed from the Active Process List and the Process Ready Queue. When the process is executing under the control of the host environment, all executions of it are terminated through host system facilities.

3.3.5.2.3 Outputs

There are no output arguments defined for Terminate_Process.

3.3.5.3 Ready Process

This facility schedules the execution of the specified MAPSE process. Ready_Process is requested to schedule the execution of an Ada task within a process.

3.3.5.3.1 Inputs

The following input argument is defined for Ready_Process:

Addr_PCB - The process control block address of the process.

3.3.5.3.2 Processing

Ready_Process validates the specified process control block. A new task enclosed by the process is scheduled for execution by inserting its instance of the block in the Process Ready Queue. The new task is selected from the Task Ready Queue maintained by the Task Manager and its status updated accordingly.

3.3.5.3.3 Outputs

There are no output arguments defined for Ready_Process.

3.3.5.4 Suspend_Process

This facility suspends the execution of a process or the execution of a task within a process.

3.3.5.4.1 Inputs

The following input arguments are defined for Suspend_Process:

Addr_PCB - The process control block address of the process enclosing the task.

Addr_TCB - The task control block address of the task to be suspended.

3.3.5.4.2 Processing

Suspend__Process validates the specified control blocks. The process execution specified by the task control block is removed from the Active Process List.

3.3.5.4.3 Outputs

There are no output arguments defined for Suspend_Process.

3.3.5.5 Rank Process

This facility modifies the scheduling priority of the specified process execution. Rank_Process is used to change the priority of an Ada task within a process.

3.3.5.5.1 Inputs

The following input arguments are defined for Rank_Process:

Addr_PCB - The process control block address of the process enclosing the task.

Addr_TCB - The task control block address of the task to be ranked.

3.3.5.5.2 Processing

Rank_Process validates the control blocks. The specified instance of the process block is set for execution and scheduled in the Process Ready Queue in accordance with the priority in the task control block.

3.3.5.5.3 Outputs

There are no output arguments defined for Rank_Process.

6

3.3.5.6 Read PCB

This facility reads the contents of the specified process control block into the designated space. Read_PCB may be requested to read the block of the requesting process or of any dependent process of the requesting process.

3.3.5.6.1 Inputs

The following input arguments are defined for Read_PCB:

Addr_PCB - The address of the process control block to be read.

Addr_VPCB - The address in the requesting process of where the contents of the process control block are to be placed.

3.3.5.6.2 Processing

Read_PCB validates the specified process block. The contents of the block are placed in the designated space.

3.3.5.6.3 Outputs

There are no output arguments defined for Read_PCB.

3.3.5.7 Terminate Process Task

This facility terminates a concurrent execution of the specified MAPSE process. Terminate_Process_Task is requested in order to terminate the execution of an Ada task within a process.

3.3.5.7.1 Inputs

The following input arguments are defined for Terminate_Process_Task:

Addr_PCB - The process control block address of the process enclosing the task.

Addr_TCB - The task control block address of the task to be terminated.

3.3.5.7.2 Processing

Terminate_Process_Task validates the specified process and task control blocks. The instance of the process block is entered into the Process Termination List. Any occurrence of it in either the Active Process List or Process Ready Queue is removed. When ·it is in the Active Process List and is executing under the control of the host environment, this execution of the process is terminated through the host system facilities.

3.3.5.7.3 Outputs

There are no output arguments defined for Terminate_Process_Task.

3.3.5.8 Wait Process

This facility waits all executions of a process depending upon the status of another process. Wait_Process may be requested to wait the requesting process or any process started by the requesting process.

3.3.5.8.1 Inputs

The following input arguments are defined for Wait_Process:

Addr_PCB	- The process control block address of the process to be waited.
Addr_PCB	- The process control block address of the process on which the wait depends.
Wait_Condition	- The condition on which to wait.

3.3.5.8.2 Processing

Wait_Process validates the specified process control blocks. The status of the block for the process to be waited is changed to waited in the Process Dictionary and all instances of it are removed from the Active Process List and placed in the Process Ready Queue. All process executions are removed from the Active Process List and are suspended as required using host system facilities. Resumption of process execution occurs upon the wait condition being satisfied or through an explicit request to resume execution.

3.3.5.8.3 Outputs

There are no output arguments defined for Wait_Process.

6V

3.3.5.9 Save Process

This facility saves a waited process as a Load Object. Save_Process may be requested by the process which started the waited process.

3.3.5.9.1 Inputs

The following input arguments are defined for Save_Process:

Addr_PCB - The process control block address of the process to be saved.

Load_Object_Name - The name of the Load Object.

3.3.5.9.2 Processing

Save_Process validates the specified process control block. The status of the block for the process is changed to saved in the Process Dictionary and a Load Object of the process execution domain is created with the name specified for the Load Object.

3.3.5.9.3 Outputs

There are no output arguments defined for Save_Process.

3.3.5.10 Resume Process

This facility resumes the execution of a waited MAPSE process. Resume_Process may be requested by the process that started the waited process.

3.3.5.10.1 Inputs

The following input argument is defined for Resume_Process:

Addr_PCB - The process control block address of the process to be resumed.

3.3.5.10.2 Processing

Resume_Process validates the specified process control block. The status of the block for the process to be resumed is changed to ready in the Process Dictionary. Instances of the block in the Process Ready Queue are now available to be scheduled for execution.

3.3.5.10.3 Outputs

There are no output arguments defined for Resume_Process.

3.3.5.11 Switch Process Task

This facility suspends and reschedules the execution of a process. Switch_Process_Task is requested so that a new Ada task within a process is scheduled for execution.

3.3.5.11.1 Inputs

The following input arguments are defined for Switch_Process_Task:

Addr_PCB - The process control block address of the process enclosing the task.

Addr_TCB - The task control block address of the task to be suspended.

3.3.5.11.2 Processing

Switch_Process_Task validates the process control block. The specified instance of the block is suspended by removing it from the Active Process List. A new task enclosed by the process is scheduled for execution by inserting its instance of the block in the Process Ready Queue.

3.3.5.11.3 Outputs

There are no output arguments defined for Switch_Process_Task.

3.3.5.12 Finish Process

This facility terminates the execution of a MAPSE process. Finish_Process is requested to perform self-termination of a process.

3.3.5.12.1 Inputs

the following input argument is defined for Finish_Process:

Process_Param - The actual parameters to be returned to the starting process.

3.3.5.12.2 Processing

Finish_Process removes the process control block from the Active Process List and the block's status is changed to finished in the Process Dictionary. Any actual parameters are placed in the process control block.

3.3.5.12.3 Outputs

There are no output arguments defined for Finish_Process.

3.3.5.13 Write PCB

This facility writes the contents of the designated space into the specified process control block. Write_PCB is a restricted request that is used to change the contents of the block of the requesting process or of any dependent process of the requesting process.

3.3.5.13.1 Inputs

The following input arguments are defined for Write_PCB:

Addr_PCB - The address of the process control block to be changed.

Addr_VPCB - The address in the requesting process of where the information to be written into the block is located.

3.3.5.13.2 Processing

Write_PCB validates the specified block. The contents of the block that are to be inserted are checked for validity and are then placed in the block. Only a limited set of visible block items may be changed.

3.3.5.13.3 Outputs

There are no outputs defined for Write_PCB.

3.3.5.14 Delete Process

This facility removes the existence of a MAPSE process. Delete_Process may be requested by the process which started the specified process.

3.3.5.14.1 Inputs

The following input argument is defined for Delete_Process:

Addr_PCB - The process control block address of the process to be deleted.

3.3.5.14.2 Processing

Delete_Process validates the specified process control block. The block is removed from the Process Dictionary and its space made available for reassignment. A process may only be deleted if it is in a finished or terminated state. When a process to be deleted has started processes that are in a finished or terminated state, these processes are automatically deleted. If the started processes are not in a finished or terminated state, the starting process for these processes is made the requesting process.

3.3.5.14.3 Outputs

There are no output arguments defined for Delete_Process.

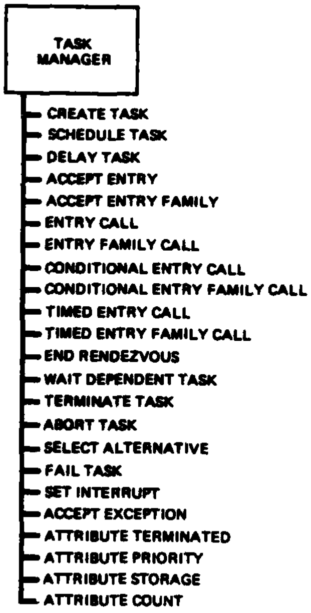
3.3.6 Task Manager

The Task Manager functionally encapsulates a set of operations on the data structure defined as the task control block. Appendix A includes the specification of the Ada package TASK_MANAGER which is made available in the virtual interface. See figure 3-15 for a logical breakdown of the Task Manager.

A MAPSE process may synchronize the concurrent execution of different code domains within the process in accordance with the semantics of Ada tasks. The Task Manager is designed to provide the necessary functionality to support Ada tasks using facilities available in the Kernel through the Process Administrator. Information required to control and schedule tasks is maintained with the Task Manager. This task information is accessible to the Process Administrator when process scheduling is to be performed within the MAPSE. A consequence of the design is that the Task Manager is insulated from changes in the host system that would affect task execution. In addition, because task information is accessible to the Process Administrator the number of explicit requests from the Task Manager to the Kernel is minimized.

The Task Manager is designed to cooperate with the Volume Manager, I/O Dispatcher, and Event Monitor to synthesize those functional requirements of a MAPSE process that may effect the harmonious execution of its tasks. Typically these requirements necessitate the use of facilities within the Kernel that result in the task being placed in the wait state pending delayed action in the host environment. An objective in supporting concurrent task execution is to ensure that such a task does not inadvertantly cause the enclosing MAPSE process to be stalled in its execution when other tasks within the same process are candidates for execution.

The Volume Manager in the Kernel performs data transfers between MAPSE level processes and the host environment. Normally the task requesting the data transfer, using Ada I/O, must await completion of the operation. Therefore, it is incumbent the Volume Manager to update the appropriate task information maintained by the Task Manager and to initiate a new scheduling decision by the Process Administrator.



TP No. 031-2094-A

Figure 3-15. Logical Breakdown

The Task Manager depends upon the Event Monitor to recognize that a data transfer has been completed and for the appropriate task information to be updated. The task may then be rescheduled for execution by the Process Administrator.

In addition, the Task Manager relies upon the Event Monitor to coordinate the scheduling of tasks that have been associated with specific interrupts by intercepting the interrupt so that the appropriate task information is updated.

3.3.6.1 Create Task

This facility creates a task control block. Create_Task is called by the prologue associated with the enclosing declarative part and executes as a procedure under the calling task.

3.3.6.1.1 Inputs

The following input arguments are defined for Create_Task:

Addr_TCB - The address of the space allocated for the task control block.

Addr_DTR - The address of the Dependent Task Record.

Addr_ESC - The address of the Enclosing Static Context.

Task_Priority - The static priority defined for the task.

Task_IEP - The Initial Execution Position for the task.

TCB_Alt - The Alternative Constraints for the task control block.

3.3.6.1.2 Processing

Create_Task initializes the space allocated to the task control block. The block chains of dependent tasks for the guardian task and scope are updated. The status of the specified task is set to indicate that the task is created and is awaiting activation (elaboration).

3.3.6.1.3 Outputs

There are no output arguments defined for Create_Task.

3.3.6.2 Schedule Task

This facility schedules a task for execution after the declarative part of the task body has been elaborated. Schedule_Task is called by the prologue associated with the enclosing declarative part and executes as a procedure under the calling task.

3.3.6.2.1 Inputs

The following input argument is defined for Schedule_Task:

Addr_TCB - The task control block address of the task to be scheduled.

3.3.6.2.2 Processing

The status of the specified task is changed to indicate that the task is ready for execution. The task control block is entered into the queue of tasks ready for execution.

3.3.6.2.3 Outputs

There are no output arguments defined for Schedule_Task.

3.3.6.3 Delay Task

This facility suspends execution of a task for at least the specified quantum of time. Delay_Task is called by a task executing a delay statement or a timed entry statement and a new task is scheduled through the Process Administrator.

3.3.6.3.1 Inputs

The following input argument is defined for Delay_Task:

Time_Delay - The quantum of time to suspend task execution.

3.3.6.3.2 Processing

The status of the task is changed to indicate that the task has been suspended for the specified quantum of time. The task control block is entered into the queue of tasks that are currently suspended.

3.3.6.3.3 Outputs

There are no output arguments defined for Delay_Task.

3.3.6.4 Accept Entry

This facility synchronizes a service task of a MAPSE process executing an accept statement with the execution of a task requesting the entry for this accept statement. Accept_Entry is called by the service task.

3.3.6.4.1 Inputs

The following input arguments are defined for Accept_Entry:

Entry_No - The identification of the accept statement entry.

Null_Accept - The condition that the entry is parameterless and the accept statement does not include executable statements.

3.3.6.4.2 Processing

The entry queue for the specified entry is inspected for a task waiting for this entry. If there is no waiting task, the service task status is changed to indicate that the task is awaiting a request for the specified entry and a new task for this process is scheduled through the Process Administrator. If there are tasks awaiting this entry the first task is removed from the queue for servicing. When the task to be serviced is in a delay status, the delay condition is cancelled. The actual parameters associated with the request are made available to the service task and execution control is directed to the service task to complete execution of the accept statement. When the Null_Accept condition is satisfied execution of the service task is continued if it is the highest priority task, otherwise a new task is scheduled through the Process Administrator.

3.3.6.4.3 Outputs

There are no output arguments defined for Accept_Entry.

3.3.6.5 Accept Entry Family

This facility synchronizes a service task of a MAPSE process executing an accept family statement with the execution of a task requesting the family member entry for this accept statement. Accept_Entry_Family is called by the service task.

3.3.6.5.1 Inputs

The following input arguments are defined for Accept_Entry_Family:

Entry_No - The identification of the accept statement entry.

Entry_Index - The identification of the entry family member.

Null_Accept - The condition that the entry is parameterless and the accept does not indicate executable statements.

3.3.6.5.2 Processing

The processing is identical to that defined for Accept_Entry once the entry queue has been located for the entry family member.

3.3.6.5.2 Outputs

There are no output arguments defined for Accept_Entry_Family.

3.3.6.6 Entry Call

This facility synchronizes a task of a MAPSE process executing an entry statement with the execution of the service task defining the entry. Entry_Call is called by the task executing the entry statement.

3.3.6.6.1 Inputs

The following input arguments are defined for Entry_Call:

Addr_TCB	-	The task control block address of the service task.
Entry_No	-	The identification of the requested entry.
Parameters	-	The actual parameters for the requested entry.

3.3.6.6.2 Processing

The actual parameters for the request are saved and the requesting task control block is appended to the specified entry queue. If the specified entry is closed, a new task is scheduled through the Process Administrator. Otherwise, when the specified entry is open, all other open entries for the service task are closed and the service task status is changed to indicate that the task is ready for execution at the control point currently associated with the entry. If the service task has been waiting at a delay statement, the delay condition is cancelled. When the service task has been waiting at a terminate statement, the changes in the dependent task relationships are propagated as required. If the service task is in a termination status the exception status of the requesting task is changed to indicate a tasking error exception. A new task is then scheduled through the Process Administrator.

3.3.6.6.3 Outputs

There are no output arguments defined for Entry_Call.

1

3.3.6.7 Entry Family Call

This facility synchronizes a task of a MAPSE process executing an entry statement for an entry family member with the execution of the service task defining the entry family. Entry_Family_Call is called by the task executing the entry statement.

3.3.6.7.1 Inputs

The following input arguments are defined for Entry_Family_Call:

Addr_TCB - The task control block address of the service task.
Entry_No - The identification of the requested entry.
Entry_Index - The identification of the entry family member.
Parameters - The actual parameters for the requested entry.

3.3.6.7.2 Processing

The processing is identical to that defined for Entry__Call except for locating the entry and entry queue for the entry family member.

3.3.6.7.3 Outputs

There are no output arguments defined for Entry_Family_Call.

3.3.6.8 Conditional Entry Call

This facility conditionally synchronizes a task of a MAPSE process executing a conditional entry statement with the execution of the service task defining the entry. Conditional_Entry_Call is called by the task executing the conditional entry statement.

3.3.6.8.1 Inputs

The following input arguments are defined for Conditional_Entry_Call:

Addr_TCB - The task control block address of the service task.
Entry_No - The identification of the requested entry.
Parameters - The actual parameters for the requested entry.

3.3.6.8.2 Processing

The processing is similar to that defined for Entry_Call. When the specified entry is closed, execution of the requesting task is continued with the condition that an immediate rendezvous has failed with the service task. The task is not appended to the specified entry queue.

3.3.6.8.3 Outputs

The following output argument is defined for Conditional_Entry_Call:

Condition - The condition as determined by the status of the requested entry.

3.3.6.9 Conditional Entry Family Call

This facility conditionally synchronizes a task of a MAPSE process executing a conditional entry statement for an entry family member with the execution of the service task defining the entry family. Conditional_Entry_Family_Call is called by the task executing the conditional entry statement.

3.3.6.9.1 Inputs

The following input arguments are defined for Conditional_Entry_Family_Call:

Addr_TCB - The task control block address of the service task.

Entry_No - The identification of the requested entry.

Entry_Index - The identification of the entry family member.

Parameters - The actual parameters for the requested entry.

3.3.6.9.2 Processing

The processing is identical to that defined for Conditional_Entry_Call but for locating the entry and entry queue for the entry family member.

3.3.6.9.3 Outputs

The following output argument is defined for Conditional_Entry_Family_Call:

Condition - The condition as determined by the status of the requested entry family member.

3.3.6.10 Timed Entry Call

This facility conditionally synchronizes a task of a MAPSE process executing a timed entry statement with the execution of the service task defining the entry. Timed_Entry_Call is called by the task executing the timed entry statement.

3.3.6.10.1 Inputs

The following input arguments are defined for Timed_Entry_Call:

Addr_TCB - The task control block address of the service task.
Entry_No - The identification of the requested entry.
Parameters - The actual parameters for the requested entry.

3.3.6.10.2 Processing

The processing is identical to that defined for Conditional_Entry_Call except that when the entry is closed the requesting task is appended to the specified entry queue.

3.3.6.10.3 Outputs

The following output argument is defined for Timed_Entry_Call:

Condition - The condition as determined by the status of the requested entry.

53

3.3.6.11 Timed Entry Family Call

This facility conditionally synchronizes a task of a MAPSE process executing a timed entry statement for an entry family member with the execution of the service task defining the entry family. Timed_Entry_Family_Call is called by the task executing the timed entry statement.

3.3.6.11.1 Inputs

The following input arguments are defined for Timed_Entry_Family_Call:

Addr_TCB - The task control block address of the service task.

Entry_No - The identification of the requested entry.

Entry_Index - The identification of the entry family member.

Parameters - The actual parameters for the requested entry.

3.3.6.11.2 Processing

The processing is identical to that defined for Timed_Entry_Call except for locating the entry and entry queue for the entry family member.

3.3.6.11.3 Outputs

The following output argument is defined for Timed_Entry_Family_Call:

Condition - The condition as determined by the status of the requested entry family member.

3.3.6.12 End Rendezvous

This facility decouples a service task of a MAPSE process from the task it is currently servicing. End_Rendezvous is called by the service task upon the completion of an accept statement.

3.3.6.12.1 Inputs

There are no input arguments defined for End_Rendezvous.

3.3.6.12.2 Processing

The status of the task that has been serviced is changed to indicate that it is ready for execution at the control point following the entry call unless a task failure has occurred. A new task is scheduled for execution through the Process Administrator, unless the accept statement completed by the service task is enclosed by an outer accept statement.

3.3.6.12.3 Outputs

There are no output arguments defined for End_Rendezvous.

3.3.6.13 Wait Dependent Task

This facility synchronizes continued execution of a MAPSE process or a task within a MAPSE process with the termination of any dependent tasks. Wait_Dependent_Task is called by the thread of execution that is to wait.

3.3.6.13.1 Inputs

The following input argument is defined for Wait_Dependent_Task:

Addr_DTR - The address of the Dependent Task Record.

3.3.6.13.2 Processing

To be determined during implementation.

3.3.6.13.3 Outputs

There are no output arguments defined for Wait_Dependent_Task.

3.3.6.14 Terminate Task

This facility terminates execution of a currently executing task within a MAPSE process. Terminate__Task is called by the task requesting to be terminated.

3.3.6.14.1 Inputs

There are no input arguments defined for Terminate_Task.

3.3.6.14.2 Processing

The status of the task is changed to indicate that it has terminated. The changes to any dependent task relationships resulting from its termination are propagated and the task control blocks of any dependent tasks are released.

3.3.6.14.3 Outputs

There are no output arguments defined for Terminate_Task.

3.3.6.15 Abort Task

This facility asynchronously terminates a task within a MAPSE process.

3.3.6.15.1 Inputs

The following input argument is defined for Abort_Task:

Addr_TCB - The task control block address of the task to be aborted.

3.3.6.15.2 Processing

The status of the task is changed to indicate that it has been terminated. If the task has requested a delay or an entry call, these requests are cancelled. If the task is currently servicing an entry request, the rendezvous is cancelled. The changes to any dependent task relationships resulting from the termination of the specified task are propagated.

3.3.6.15.3 Outputs

There are no output arguments defined for Abort_Task.

3.3.6.16 Select Alternative

This facility conditionally synchronizes a service task of a MAPSE process executing a select statement with the execution of a task requesting an open entry enclosed by the select statement. Failure to synchronize with a task may result in the service task being terminated. Select_Alternative is called by the service task.

3.3.6.16.1 Inputs

The following input argument is defined for Select_Alternative:

Select_Table	-	The table describing all the alternatives enclosed by the select statement.

3.3.6.16.2 Processing

The open alternatives are investigated in the task control block. When only entry alternatives are open, the corresponding entry queues are inspected for a waiting task. If only one entry has a nonempty queue, the first task is removed from the queue for servicing. If multiple entries have nonempty queues, a queue is arbitrarily chosen and the first task is removed for servicing. The removed task is placed in a rendezvous state. When the task to be serviced is in a delay status, the delay condition is cancelled. The actual parameters associated with the entry are made available to the service task and execution control is directed to the service task as defined in the Select Table. When the open alternatives include delay statements and there is no task to service, a delay alternative from the set of equivalent-valued delay statements is chosen and the service task is suspended for the specified quantum of time.

When the open alternatives include a terminate statement, it is chosen if the termination conditions are satisfied in the Dependent Task Record of the guardian task. The service task is terminated if all dependent tasks in the dependent task records of the guardian task have terminated or are suspended awaiting a terminate statement. Otherwise, the service task is suspended in a terminate state.

If the open alternatives do not include a delay or selectable terminate, the service task is suspended to await a request for an open entry alternative. In the event that there are no open alternatives, the else alternative if available, is selected by the service task. When no else alternative is available, the select error exception is raised in the service task.

Upon suspending the service task, a new task for this process is scheduled through the Process Administrator.

3.3.6.16.3 Outputs

There are no output arguments defined for Select_Alternative.

3.3.6.17 Fail Task

This facility causes the failure exception condition in a task within a MAPSE process.

3.3.6.17.1 Inputs

The following input argument is defined for Fail_Task:

Addr_TCB - The task control block address of the task to receive the failure exception.

3.3.6.17.2 Processing

The exception status of the task is changed to indicate that a failure exception has been received. If the task has requested a delay or an entry call, these requests are cancelled. In these instances and when the task has been suspended, the status of the task is changed to indicate that it is ready for execution and the Process Administrator is called to schedule a new task. When the failed task is currently running, it is made ready for execution at the failure exception control point, and the Process Administrator is called to suspend the task.

3.3.6.17.3 Outputs

There are no output arguments defined for Fail_Task.

3.3.6.18 Set Interrupt

This facility associates the specified interrupt with an entry statement of a task within a MAPSE process. Set_Interrupt is called by the prologue associated with the enclosing declarative part and executes under the current thread of control.

3.3.6.18.1 Inputs

The following input arguments are defined for Set_Interrupt:

Addr_TCB - The task control block address of the task enclosing the interrupt entry.

Entry_No - The identification of the interrupt entry.

Interrupt_Name - The name of the interrupt.

Entry_Index - The identification of the entry family member.

3.3.6.18.2 Processing

The specified entry is marked as an interrupt entry. The Event Monitor is called to set an event for the named interrupt.

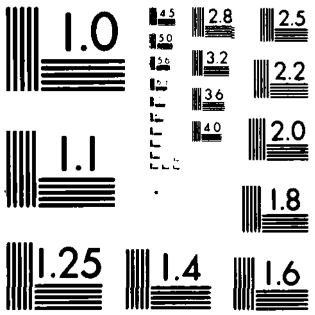
3.3.6.18.3 Outputs

There are no output arguments defined for Set_Interrupt.

AD-A109 980 COMPUTER SCIENCES CORP FALLS CHURCH VA F/G 9/2
ADA INTEGRATED ENVIRONMENT II COMPUTER PROGRAM DEVELOPMENT SPEC--ETC(U)
DEC 81 F30602-80-C-0292
UNCLASSIFIED RADC-TR-81-364-PT-1 NL

2 OF 4
AD
A109 980

A109 98



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

3.3.6.19 Accept Exception

This facility propagates an exception in an accept statement of a task within a MAPSE process. Accept_Exception is called by the service task enclosing the accept statement.

3.3.6.19.1 Inputs

The following input argument is defined for Accept_Exception:

Exception - The name of the exception to be propagated.

3.3.6.19.2 Processing

The exception status of all tasks synchronized with the service task is changed to indicate that the specified exception has occurred. When the task failure exception is propagated, it is renamed tasking error. The status of these tasks is changed to indicate that they are ready for execution and the Process Administrator is called to schedule a new task.

3.3.6.19.3 Outputs

There are no output arguments defined for Accept_Exception.

3.3.6.20 Attribute Terminated

This facility reports the termination status of a task within a MAPSE process.

3.3.6.20.1 Inputs

The following input argument is defined for Attribute_Terminated:

Addr_TCB - The task control block address of the task to be reported upon.

3.3.6.20.2 Processing

The termination status of the specified task is returned to the requesting task.

3.3.6.20.3 Outputs

The following output argument is defined for Attribute_Terminated:

Termination_Status - The value true is returned if the specified task has terminated.

3.3.6.21 Attribute Priority

This facility reports the priority of a task within a MAPSE process.

3.3.6.21.1 Inputs

The following input argument is defined for Attribute_Priority:

Addr_TCB - The task control block address of the task to be reported upon.

3.3.6.21.2 Processing

The priority value defined for the specified task is returned to the requesting task.

3.3.6.21.3 Outputs

The following output argument is defined for Attribute_Priority:

Priority - The priority value of the specified task.

3.3.6.22 Attribute Storage

This facility reports the number of storage units allocated to a task within a MAPSE process.

3.3.6.22.1 Inputs

The following input argument is defined for Attribute_Storage:

Addr_TCB - the task control block address of the task to be reported upon.

3.3.6.22.2 Processing

The number of storage units currently allocated to the specified task is returned to the requesting task.

3.3.6.22.3 Outputs

The following output argument is defined for Attribute_Storage:

Allocation - The number of storage units allocated to the specified task.

3.3.6.23 Attribute Count

The facility reports the number of outstanding calls for an entry of a service task within a MAPSE process.

3.3.6.23.1 Inputs

The following input arguments are defined for Attribute_Count:

Addr_TCB - The task control block address of the task enclosing the entry.

Entry_No - The identification of the entry to be reported upon.

Entry_Index - The identification of the entry family member

3.3.6.23.2 Processing

The length of the queue associated with the specified entry is returned to the requesting task.

3.3.6.23.3 Outputs

The following output argument is defined for Attribute_Count:

Queue_Length - The number of entry and interrupt calls currently queued.

3.5.7 Context Manager

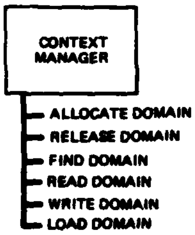
The Context Manager functionally encapsulates a set of operations on the Dynamic Address Domain and Shared Execution Domain that are defined within the MAPSE. Appendix A includes the specification of the Ada package CONTEXT_MANAGER that is made available in the virtual interface. See Figure 3-16 for a logical breakdown of the Context Manager.

The Context Manager is designed to provide the facilities that are necessary for a MAPSE process to dynamically change the address domain that it or another process may reference. Use of these facilities is restricted to MAPSE tools to safeguard the integrity of the MAPSE.

The address domain that may be referenced by a MAPSE process is initially established in the process context map when a process is started. The map associates an index with each address space that comprises the process address domain. The Context Manager allows a process to change its address space through a process context map index that may be dynamically associated with an address domain created in the Dynamic Address Domain.

The Context Manager through the Dynamic Address Domain provides the collection of storage units that may be acquired for a process. Each acquisition defines an address space that may be referenced by the process through the index associated with the domain.

In addition to supporting the Dynamic Address Domain, the Context Manager provides the functionality required by the Shared Execution Domain that enables multiple processes to share common executable domains, such as the Task Manager.



TP No. 031-2085-A

Figure 3-16. Logical Breakdown

3.3.7.1 Allocate Domain

This facility allocates storage units to a MAPSE process from the Dynamic Address Domain. Allocate__Domain is requested by a MAPSE process to dynamically update the process context map of a specified process.

3.3.7.1.1 Inputs

The following input arguments are defined for Allocate_Domain:

Addr_PCB - The process control block address of the process to receive the allocation.

Map_index - The process context map index to be associated with the domain.

Domain_length - The number of storage units to be allocated in the domain.

3.3.7.1.2 Processing

The allocation request is validated. The specified number of contiguous storage units is acquired from the Dynamic Address Domain. The process context map for the process is updated to reference the acquired domain, and the domain address is made available to the requesting process. When the request cannot be satisfied, the domain address is voided.

3.3.7.1.3 Outputs

The following output argument is defined for Allocate_Domain:

Addr_Domain - The domain address.

3.3.7.2 Release Domain

This facility frees the storage units that have been acquired for a MAPSE process from the Dynamic Address Domain. Release_Domain is requested by a MAPSE process to dynamically update the process context map of a specified process.

3.3.7.2.1 Inputs

The following input arguments are defined for Release_Domain:

Addr_PCB	-	The process control block address of the process for which the domain was acquired.
Map_Index	-	The process context map index associated with the domain.

3.3.7.2.2 Processing

The release request is validated. The process context map for the process is updated to void referencing the domain to be released. The storage units are returned to the Dynamic Address Domain for disposal. If the domain is not included in the context map of another process, the storage units are made available for subsequent allocation.

3.3.7.2.3 Outputs

There are no output arguments defined for Release_Domain.

3.3.7.3 Find Domain

This facility locates the domain address of the specified Load Object. Find Domain is requested by the Process Administrator to ascertain the sharability of a Load Object.

3.3.7.3.1 Inputs

The following input argument is defined for Find_Domain:

Load_Object_Name - The name of the Load Object.

3.3.7.3.2 Processing

The find request is validated. The Shared Execution Domain for the MAPSE is searched for the existence of the specified Load Object. If the Load Object is found the domain address is made available to the requesting process.

3.3.7.3.3 Outputs

The following output argument is defined for Find_Domain:

Addr_Domain - The domain address of the Load Object.

3.3.7.4 Read Domain

This facility reads the contents of a specified number of storage units. Read_Domain enables a MAPSE process to read the contents of a domain that is part of a descendent process. Read_Domain is provided specifically for the use of the MAPSE Debugger.

3.3.7.4.1 Inputs

The following input arguments are defined for Read_Domain:

Addr_Domain - The address of the domain containing the specified storage units.

Storage_Unit - The first storage unit to be read.

Unit_Length - The number of storage units to be read.

Addr_Buffer - The buffer address.

3.3.7.4.2 Processing

The read request is validated. The storage units within the address domain are located and their contents written to the referenced buffer in the requesting process.

3.3.7.4.3 Outputs

There are no outputs defined for Read_Domain.

3.3.7.5 Write Domain

This facility writes the contents of a specified number of storage units. Write_Domain enables a MAPSE process to write the contents of a domain that is part of a descendent process. Write_Domain is provided specifically for use by the MAPSE Debugger.

3.3.7.5.1 Inputs

The following input arguments are defined for Write_Domain:

Addr_Domain - The domain address containing the specified storage units.

Storage_Unit - The first storage unit to be written.

Unit_Length - The number of storage units to be written.

Addr_Buffer - The buffer address.

3.3.7.5.2 Processing

The write request is validated. The storage units within the address domain are located and are overwritten with the contents of the referenced buffer in the requesting process.

3.3.7.5.3 Outputs

There are no output arguments defined for Write_Domain.

3.3.7.6 Load Domain

This facility loads the specified Load Object into the Shared Execution Domain of the MAPSE. Load_Domain is requested by the Process Administrator to enable common executable domains to be shared among MAPSE processes.

3.3.7.6.1 Inputs

The following input argument is defined for Load_Domain:

Load_Object_Name - The name of the Load Object.

3.3.7.6.2 Processing

The load request is validated. The specified Load Object is loaded into t Shared Execution Domain and its domain address is made available t requesting process. If the Load Object cannot be accommodated in the Execution Domain, the domain address is voided.

3.3.7.6.3 Outputs

The following output argument is defined for Load_Domain:

Addr_Domain - The domain address of the Load Object.

3.3.8 Event Monitor

The Event Monitor functionally encapsulates a set of operations on the data structures defined as the Event Queues. Appendix A includes the specification of the Ada package EVENT_MONITOR that is made available in the virtual interface. See Figure 3-17 for a logical breakdown of the Event Monitor.

The Event Monitor is a functional unit within the Kernel of the KFW. It is designed to reconcile the asynchronous performance of host system facilities with the execution of concurrent MAPSE processes. A primary responsibility of the Event Monitor is the synchronization of the MAPSE clock. This is achieved by requesting an event to be posted at the expiration of a standard quantum of time. This event is typically represented in the host system either as a type of interrupt or through an event mechanism.

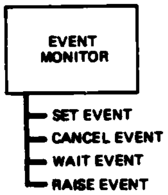
In addition to maintaining the MAPSE clock, the Event Monitor includes facilities to set, raise, wait, and cancel events. These facilities provide services essential to the functionality supplied by the I/O Dispatcher and Task Manager in support of the requirements of a MAPSE process.

The specification of an event control block contains information that describes an event. Using this control block, the Event Monitor associates the occurrence of an event with an execution domain that is to be scheduled or performed. When an event is defined to the Event Monitor, it is entered into the Event Queue for the class of event. The Event Monitor implements events for I/O completion, time delay expiration, and named hardware interrupts.

Depending upon the kind of the host system facilities available, the Event Monitor polls its entry queues periodically or is activated when an event is posted.

The schematic in Figure 3-18 illustrates the delay of a task for a quantum of time.

When an entry interrupt occurs the corresponding event control block is entered in the appropriate interrupt queue for the task enclosing the entry. If the task is currently suspended and the entry is open, then the



TP No. 031-2086-A

Figure 3-17. Logical Breakdown

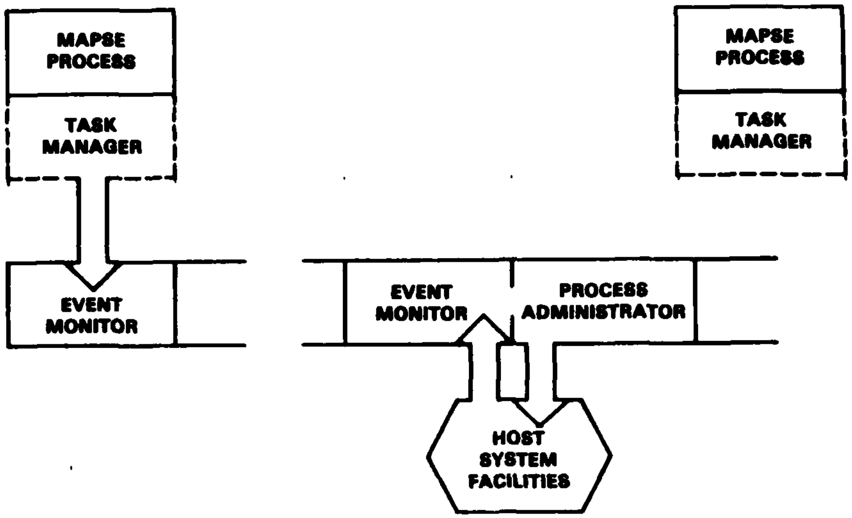


Figure 3-18. Task Delay

task status is changed to indicate that it is ready for execution and this instance of the process control block is entered in the Process Ready Queue. The resumption of this task results in the interrupt entry being serviced before any other open entries.

3.3.8.1 Set Event

This facility associates the occurrence of an action in the host environment with the execution domain specified in an event control block. Set_Event is called by the I/O Dispatcher and Task Manager.

3.3.8.1.1 Inputs

The following input argument is defined for Set_Event:

Addr_ECB - The event control block address.

3.3.8.1.2 Processing

The event control block is validated and entered into the appropriate Event Queue. An event may specify the expiration of a quantum of time, the completion of an I/O requ*st or the occurrence of a discernible interrupt. For interrupt and time events, the event control block includes a reference to the task control block that defines the execution domain. For I/O request events, the event control block includes a reference to the carrier control block or device control block associated with the command specification block that initiated the I/O request. Time events are entered into the Event Queue to ensure that the event with the smallest quantum is at the head of the queue.

3.3.8.1.3 Outputs

There are no outputs arguments defined for Set_Event.

3.3.8.2 Cancel Event

This facility cancels the event specified in an event control block. Cancel_Event is called by the Task Manager.

3.3.8.2.1 Inputs

The following input argument is defined for Cancel_Event:

Addr_ECB - The event control block address.

3.3.8.2.2 Processing

The event control block is validated and removed from the appropriate Event Queue.

3.3.8.2.3 Outputs

There are no output arguments defined for Cancel_Event.

3.3.8.3 Wait Event

This facility suspends execution of a task within a MAPSE process to await the occurrence of an action in the host environment that is specified in an event control block. Wait_Event is called by the I/O Dispatcher and the Task Manager.

3.3.8.3.1 Inputs

The following input argument is defined for Wait_Event:

Addr_ECB - The event control block address.

3.3.8.3.2 Processing

The event control block is validated and entered into the appropriate Event Queue. The specified execution of the task is suspended.

3.3.8.3.2 Outputs

There are no output arguments defined for Wait_Event.

3.3.8.4 Raise Event

This facility enables a previously set event to be activated. Raise_Event is called by the Volume Manager.

3.3.8.4.1 Inputs

The following input argument is defined for Raise_Event:

Event_Name - The name of the event to be made active.

3.3.8.4.2 Processing

The event control block corresponding to the named event in the Event Queue is removed. The process execution associated with the event is placed in the Process Ready Queue. Raise_Event is used by the Volume Manager to implement the terminal attention or breakin facilities.

3.3.8.4.3 Outputs

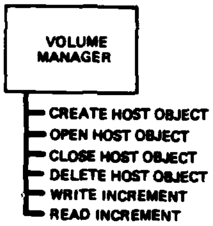
There are no output arguments defined for Raise_Event.

3.3.9 Volume Manager

The Volume Manager functionally encapsulates a set of operations on the data structure defined as the carrier control block. Appendix A includes the specification of the Ada package VOLUME_MANAGER. See Figure 3-19 for a logical breakdown of the Volume Manager.

The Volume Manager creates an abstract host object that can be manipulated in order to maintain the information contained in the KAPSE data base. The Volume Manager is designed to use the host system facilities to convert an abstract host object into the appropriate logical or physical device or file in the host environment.

An abstract host object is defined to the KDBS as a linear data space that may be referenced in data increments through the ordinal position assigned to the data increment. The length of the logical space and the increment may vary for each instantiation of an abstract host object. The correspondence between a data base object and an abstract host object is retained in the object control block maintained by the KDBS and the carrier control block maintained by the Volume Manager. The volume control block is used to denote the union of the object and carrier control blocks.



TP No. 031-2087-A

Figure 3-19. Logical Breakdown

115

The host dependent characteristics of the volume are retained by the Volume Manager in the carrier control block. When a data base object is created, the nature of the host file or device for the object may be optionally specified in the object control block. This may indicate a specific device in the the case of a device object. When no specific device is specified, the Volume Manager supplies a host file of a default nature. The name of the default file is retained in the directory entry for the object. The same manipulative operations are supported by the Volume Manager for all KDB objects, so that the KDBS is isolated from the nature of the host file or devices.

For host objects that represent interactive devices, the Volume Manager contains the necessary functionality to provide for simple editing of input lines. It is recognized that on some hosts it may not be possible to override the host line-editing facilities.

The Schematic in Figure 3-20 illustrates the role of the Volume Manager in a typical data retrieval cycle for a MAPSE process.

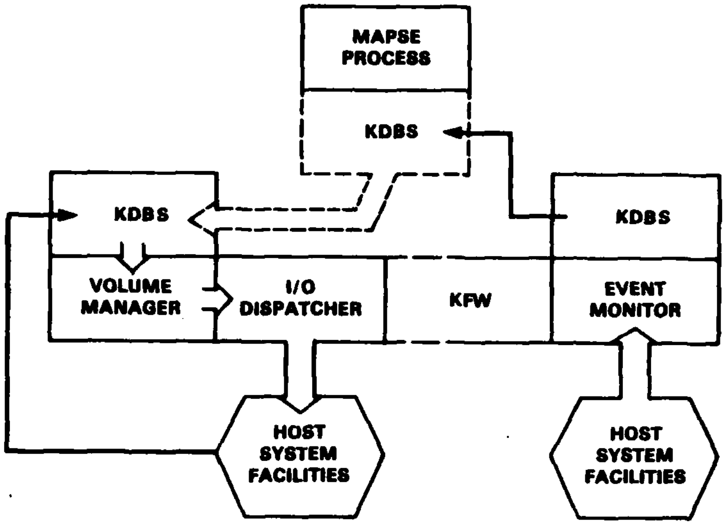


Figure 3-20. MAPSE Data Retrieval Cycle

3.3.9.1 Create Host Object

This facility creates a host object for the KDB object described in the object control block. The name and nature of the host object are made available through this block. Create_Host_Object is called by the KDBS from the Kernel Process.

3.3.9.1.1 Inputs

The following input argument is defined for Create_Host_Object:

Addr_OCB - The object control block address.

3.3.9.1.2 Processing

The KDB object description in the object control block is validated and a carrier control block is created. The KDB object description is used to generate a host object name and a command specification block. This block is linked to the carrier control block, and a reference to the carrier block and the host object name are placed in the object control block. The carrier control block is passed to the I/O Dispatcher to initiate the specified host system facility.

3.3.9.2 Open Host Object

This facility makes available the host object specified in the object control block. Open_Host_Object is called by the KDBS from the Kernel.

3.3.9.2.1 Inputs

The following input argument is defined for Open_Host_Object:

Addr_OCB - The object control block address.

3.3.9.2.2 Processing

The KDB object description in the object control block is validated and a carrier control block is created. The command specification block to make the host object available is formatted and linked to the carrier block. A reference to the latter is placed in the object control block and the carrier control block is passed to the I/O Dispatcher to initiate the specified host system facility. When the host object requires no explicit use of the host system facilities, control is returned to the KDBS without creating a command specification block.

3.3.9.2.3 Outputs

There are no output arguments defined for Open_Host_Object.

3.3.9.3 Close Host Object

This facility releases the host object specified in the object control block. Close_Host_Object is called by the KDBS from the Kernel.

3.3.9.3.1 Inputs

The following input argument is defined for Close_Host_Object:

Addr_OCB - The object control block address.

3.3.9.3.2 Processing

The carrier control block referenced in the object block is validated. The command specification block to close the host object is formatted and linked to the carrier block. The carrier block is passed to the I/O Dispatcher to initiate the specified host system facility. When the host object requires no explicit use of the host system facilities, the carrier block reference in the object block is removed and the carrier block released. Control may then be returned to the KDBS immediately.

3.3.9.3.3 Outputs

There are no output arguments defined for Close_Host_Object.

3.3.9.4 Delete Host Object

This facility deletes the host object specified by the object control block. Delete_Host_Object is called by the KDBS from the Kernel.

3.3.9.4.1 Inputs

The following input argument is defined for Delete_Host_Object:

Addr_OCB - The object control block address.

3.3.9.4.2 Processing

The host object description in the object control block is validated and a carrier control block is created. The command specification block to delete the host object identified in the object block is formatted and linked to the carrier block. A reference to the carrier block is placed in the object block, and the carrier block is passed to the I/O Dispatcher to initiate the specified host system facility.

3.3.9.4.3 Outputs

There are no output arguments defined for Delete_Host_Object.

3.3.9.5 Write Increment

This facility writes a data increment to the host object specified by the object control block. The data to be written is supplied in the buffer referenced in the object block. Write_Increment is called by the KDBS from the Kernel.

3.3.9.5.1 Inputs

The following input argument is defined for Write_Increment:

Addr_OCB - The object control block address.

3.3.9.5.2 Processing

The carrier command block referenced in the object block is validated. From the data increment description in the object block, the command specification blocks are formatted to write the contents of the buffer to the designated position in the host object. The command blocks are linked to the carrier control block, which is passed to the I/O Dispatcher to initiate the specified host system facility.

3.3.9.5.3 Outputs

There are no output arguments defined for Write_Increment.

3.3.9.6 Read Increment

This facility reads a data increment from the host object specified by the object control block. The data that are read are placed in the buffer referenced in the object block. Read_Increment is called by the KDBS from the Kernel.

3.3.9.6.1 Inputs

The following input argument is defined for Read_Increment:

Addr_OCB - The object control block address.

3.3.9.6.2 Processing

The carrier control block referenced in the object block is validated. From the data increment description in the object block, the command specification blocks are formatted to read the data from the designated position in the host object into the buffer. The command blocks are linked to the carrier block, which is passed to the I/O Dispatcher to initiate the specified host system facilities.

3.3.9.6.3 Outputs

There are no output arguments defined for Read_Increment.

3.3.10 I/O Dispatcher

The I/O Dispatcher functionally encapsulates a set of operations on the data structures defined as the Device Dispatch Queues . Appendix A includes the specification of the Ada package IO_DISPATCHER. See Figure 3-21 for a logical breakdown of the I/O Dispatcher.

The I/O Dispatcher is designed to coordinate requests resulting from KDBS manipulation or from Ada low level input and output. When data transfer operations have been requested from the same physical device, the I/O Dispatcher ensures that they are passed to the host system facilities in an order that provides for maximum efficiency in the host environment.

The KDBS coordinates the manipulation of data base objects to avoid logical data inconsistencies when concurrent processes request access to the same KDB object; however, it cannot guard against interference resulting from accesses to different objects that have been mapped to the same host object by the Volume Manager. Additional conflicts can occur when a physical device can be accessed through the facilities of Ada low level input and output, that are performed outside of the Volume Manager. The I/O Dispatcher reconciles these potential conflicts by entering all requests into the Device Dispatch Queues that it maintains and services.

An initiate request to the I/O Dispatcher specifies a carrier control block or a device control block. A carrier or device control block references one or more command specification blocks that indicate what specific host system facility is to be initiated. From the information in the carrier block, the I/O Dispatcher determines the appropriate device queue in which the command blocks are to be entered. As a result, command blocks from multiple requests that are directed to the same physical device are initiated from the same device queue, thereby precluding initiating interleaved blocks from different requests. The I/O Dispatcher cooperates with the Event Monitor to sustain servicing of the device queues. Prior to initiating a command specification block, the I/O Dispatcher sets an event with the Event Monitor. Upon completion of the event the Event Monitor activates the I/O Dispatcher for the next request. Other asynchronous requests that have entered command blocks in different device queues are also initiated when possible.

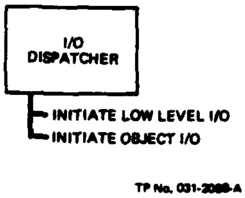


Figure 3-21. Logical Breakdown

While a request has outstanding command blccks on a device queue, the requesting task in the MAPSE process is suspended. It is rescheduled for execution through the Process Administrator by the I/O Dispatcher only when a device control block has been specified. When a carrier control block has been specified, rescheduling action is performed by the Volume Manager or KDBS.

The schematic in Figure 3-22 illustrates the servicing of concurrent initiate requests to the I/O Dispatcher.

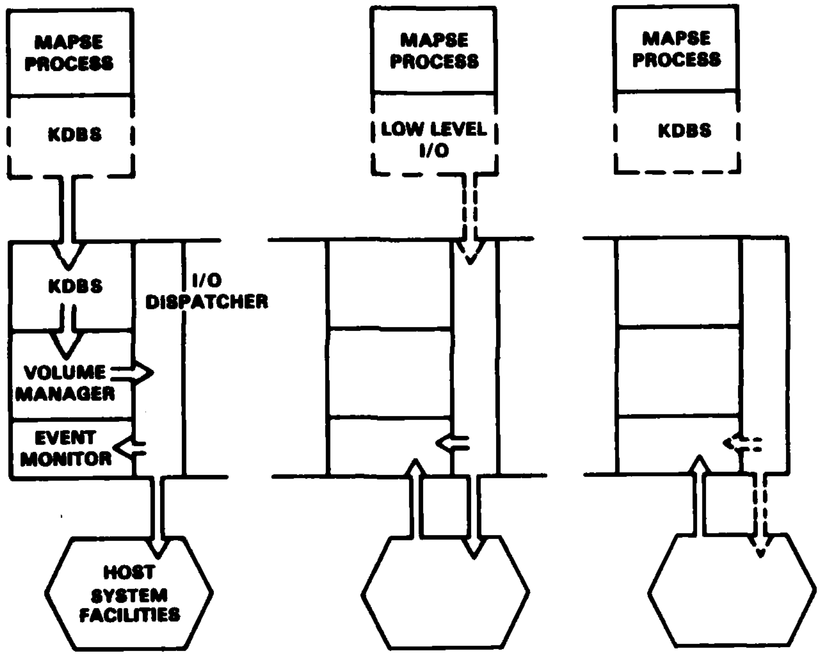


Figure 3-22. Concurrent Initiate Requests

3.3.10.1 Initiate Lowlevel IO

This facility schedules the initiation of the command specification blocks referenced in the device control block. Initiate_Lowlevel_IO is called through the Ada lowlevel I/O facilities.

3.3.10.1.1 Inputs

The following input argument is defined for Initiate_Lowlevel_IO:

Addr_CB	-	The device control block address.

3.3.10.1.2 Processing

The device control block is validated. The command specification blocks referenced in the device block are entered into the appropriate device queue. When there are existing entries in the device queue the Process Administrator is called to suspend the process or task that is dependent upon the request. Otherwise, an event is set for the first command block, which is then used to initiate the required host system facility and the process-task is suspended. The remaining command blocks are initiated as each posted event is received by the Event Monitor indicating the completion of the requested command block. When the last command block has been completed, the device control block is updated as required and the suspended process or task is rescheduled for execution.

3.3.10.1.3 Outputs

There are no output arguments defined for Initiate_Lowlevel_IO.

3.3.10.2 Initiate Object IO

This facility schedules the initiation of the command specification blocks referenced in the carrier control block. Initiate_Object_IO is called by the Volume Manager.

3.3.10.2.1 Inputs

The following input argument is defined for Initiate_Object_IO:

Addr_CCB - The carrier control block address.

3.3.10.2.2 Processing

The carrier control block is validated. The command specification blocks referenced in the carrier block are entered into the appropriate device queue. Processing is similar to that for Initiate_Lowlevel_IO, except that when the last command block has been completed control is returned to the Volume Manager.

3.3.10.2.3 Outputs

There are no output arguments defined for Initiate_Object_IO.

3.3.11 KFW Loader

The KFW Loader provides the MAPSE with the facility to load a Load Object that is to execute as a MAPSE process into an execution domain that has been created through host system facilities.

In order to load a Load Object the KFW Loader relies on the object name and PCB address being made available to it. The Load Object name identifies the host object that contains the Load Object. The KFW Loader uses this name to access the Load Object through the host system facilities.

The schematic in Figure 3-23 illustrates the loading of a new MAPSE process.

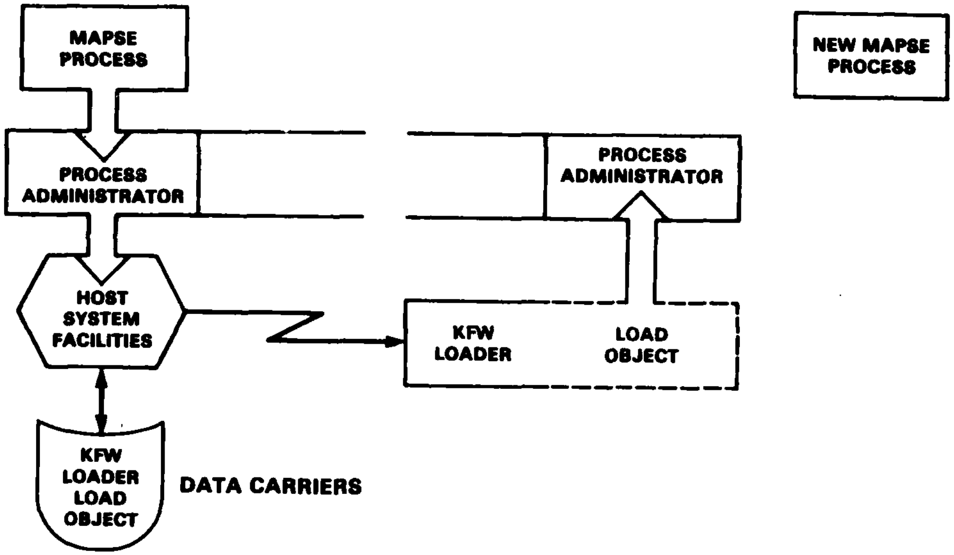


Figure 3-23. Loading New MAPSE Process

3.4 ADAPTATION

The initial implementations of the MAPSE are to use the IBM VM/370 and Interdata 8/32 under OS/32 host environments. Consequently, the specified KFW design must be adaptable to these two environments so that an economical, efficient instantiation can be specified. The following paragraphs discuss adaptation strategies for implementing the MAPSE.

3.4.1 General Environment

The two initial host environments are substantially different in the system facilities offered to the KFW. The IBM VM/370 offers a low level machine interface while the Interdata OS/32 offers an interface of a conventional multiprogramming system. Neither host system provides multiprocessing facilities that can be exploited by the KFW in the existing host configurations . Because instantiation of the KFW in the OS/32 environment interfaces with existing software, the efficient adaptation of the KFW represents a significant challenge.

3.4.1.1 IBM VM/370

VM/370 can be categorized as a virtual machine environment oriented to simulating concurrently operating virtual machines under the supervision of a Control Program (CP). The CP is the real machine resource manager. It allocates the control processing unit to concurrently operating virtual machines, handles all real machine hardware interrupts, schedules and initiates all real I/O operations and manages real and external page storage to support virtual storage.

The adaptation strategy for the KFW in VM/370 is the development of an operating system conforming to the virtual interfaces defined in this specification. This adaptation strategy is consistent with the objectives of both VM/370 and the MAPSE.

VM/370 provides virtual machines to support concurrently executing operating systems that service the needs of different programming communities. The MAPSE is one such programming community. Consequently, the requirements for multiuser support, economical portability and previous efforts to adapt other programming environments under CMS favor developing the KAPSE virtual operating system directly on a virtual machine.

3.4.1.1.1 Kernel Process

The overall adaptation strategy is the specification of the KFW as the operating system for the virtual machine. The Kernel process is created as a saved system that is IPLed in the virtual machine at logon time after the virtual machine is established. The name of the Kernel process is defined in the virtual machine configuration entry of the VM/370 directory. The Kernel process executes in the virtual supervisor state and may, as a result, issue privileged instructions. Executing in the supervisory state the Kernel process may reference any area of virtual storage (address domain) that is defined for the virtual machine. The LOCK option is used to eliminate paging activity for the most frequently used pages of the Kernel process. If necessary, additional pageable CP routines may be supplied for use by the Kernel process.

3.4.1.1.2 MAPSE Process

A MAPSE process is executed in the virtual problem state. The Kernel process establishes a MAPSE process through the virtual PSW and the KAPSE Loader.

3.4.1.1.3 Dynamic Address Domain

The Dynamic Address Domain is allocated in the virtual storage defined for the virtual machine. The Context Manager assigns page frames for the allocated virtual storage to a MAPSE process providing the required store and fetch protection using the SET STORAGE KEY instruction.

3.4.1.1.4 Shared Execution Domain

For MAPSE processes that require MAPSE facilities that execute as an extension of the process in the Shared Execution Domain (such as the Task Manager), the Context Manager assigns these facilities to page frames containing locked pages in virtual storage. The protection key for the page frames is set to permit shared execution.

3.4.1.1.5 Kernel Requests

A Kernel request from a MAPSE process is supported through the SVC instruction. This enables a MAPSE process to interrupt its execution and to invoke the virtual SVC interrupt handling routine supplied through the Event Monitor of the Kernel process.

3.4.1.1.6 MAPSE Process Execution

The execution of a MAPSE process is performed by the Kernel process through the KAPSE Loader and the use of the LOAD PSW instruction. The name of the Load Object to be executed as a MAPSE process is supplied in the call to the KAPSE Loader. The Process Administrator controls subsequent execution of the MAPSE through virtual interrupts and the virtual CPU Timer.

3.4.1.1.7 MAPSE Events

Interrupt handlers are defined in the Kernel process to receive virtual interrupts. Through the interrupt handlers virtual device interrupts, CPU Time expiration and SVC requests are received by the Event Monitor for action by the Kernel process.

3.4.1.1.8 Interactive Communication

Interactive communication with the MAPSE is supported through the virtual console devices configured in the virtual machine. A MAPSE terminal handler in the Kernel process services input directed to or from the virtual consoles recognizing the editing and attention or breakin characters that are significant to the MAPSE.

3.4.1.1.9 Standard Quantum of Time

The standard quantum of time for MAPSE process execution is provided in the Kernel process using the virtual interval timer and virtual CPU timer facilities.

3.4.1.1.10 Low Level I/O

Low Level I/O is provided through the I/O Dispatcher`s use of the START I/O instruction.

3.4.1.1.11 KAPSE Data Base I/O

The Volume Manager provides the required file structuring and manipulation to map a data base object on the virtual minidisks configured in the virtual machine. Delineation of where the virtual device handlers are located is to be determined.

3.4.1.1.12 Ada Tasks

Ada tasks are supported by multiprogramming within a MAPSE process. The Kernel process is used to schedule execution of the MAPSE through system control instructions and virtual timer interrupts.

3.4.1.1.13 KAPSE Loader

The KAPSE Loader is called to load a MAPSE process from a Load Object file. The KAPSE Loader places the Load Object in virtual storage and starts execution of the MAPSE through the Process Administrator.

3.4.1.2 Interdata 8/32

OS/32 can be categorized as a real-time operating system oriented towards dedicated applications. The system supports the execution of background programs while executing real-time programs in the foreground. Interactive support is provided by the Multi-Terminal Monitor (MTM) subsystem. The schemata in Figures 3-24 and 3-25 illustrate the adaptation strategy of the MAPSE to OS/32.

3.4.1.2.1 Kernel Process

The overall KFW adaptation strategy is the specification of a subsystem that is similar to MTM. This subsystem is the KAPSE Kernel process and establishes a privileged relationship with OS/32 through its declaration as an Executive task when the Kernel process is built at Task Establishment Time (TET). As an Executive task (E-task) the Kernel process may reference any area of memory (address domain) and may execute all host machine instructions. Additional system facilities are also provided for use by an

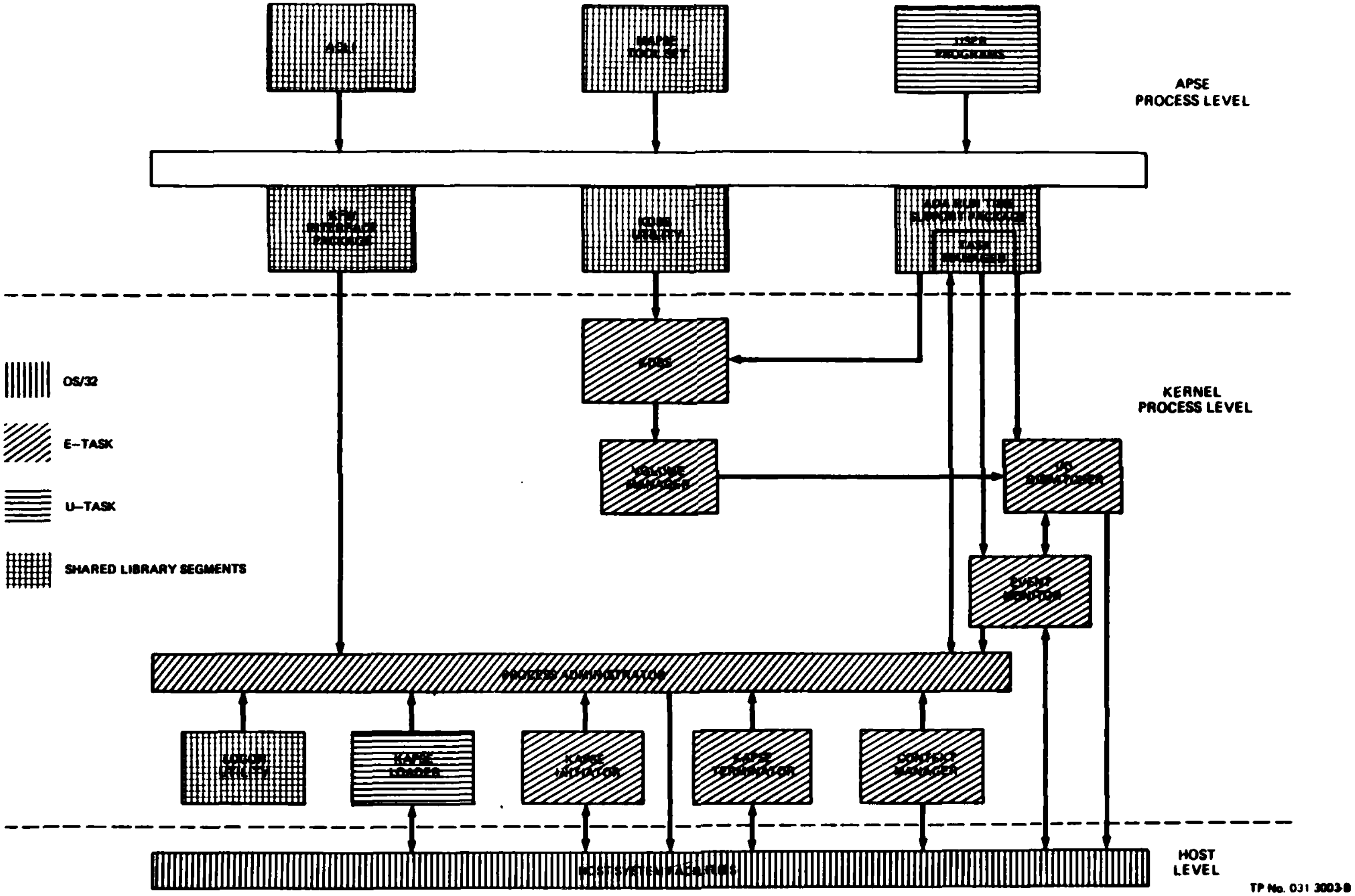


Figure 3-24. OS/32 Adaptation Strategy

OS/32 ADAPTATION STRATEGY (SVC USAGE)

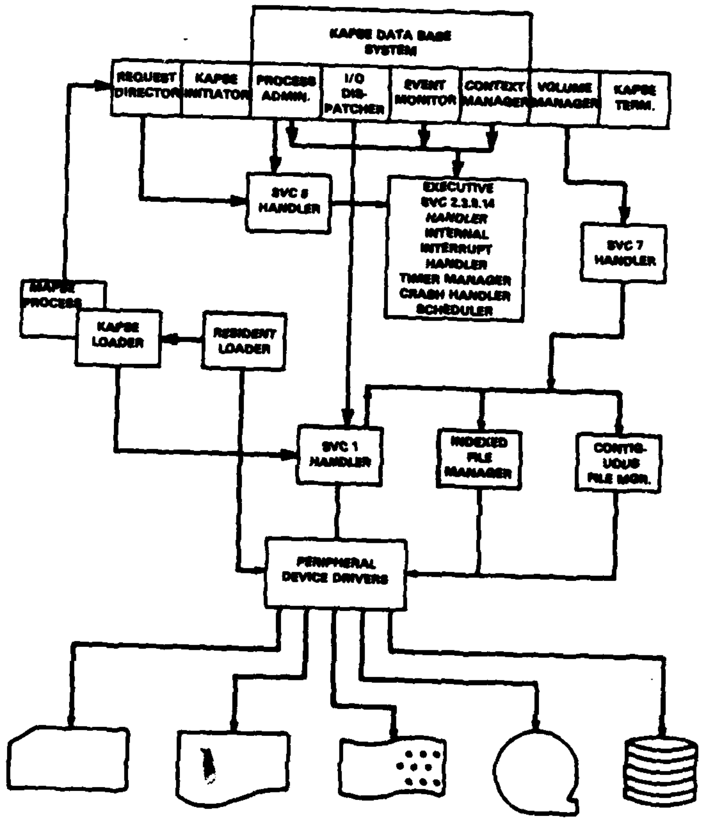


Figure 3-25

137

E-task including direct device manipulation to perform input and output. The restrictions placed on E-tasks do not present major difficulties for KFW adaptation. When the Kernel process is loaded into the OS/32 system the task resident option is specified to avoid it being rolled during MAPSE execution.

3.4.1.2.2 MAPSE Process

A MAPSE process is executed as a User task (U-task) by declaring the KAPSE Loader as a U-task when it is built. A MAPSE process is started as a monitor task of the Kernel process.

3.4.1.2.3 Dynamic Address Domain

MAPSE processes that reference the Dynamic Address Domain are supported by establishing a Global Task Common Segment that is referenced at task establishment time of the KAPSE Loader.

3.4.1.2.4 Shared Execution Domain

For MAPSE processes that require MAPSE facilities that execute as an extension of the process in the Shared Execution Domain (such as the Task Manager) a Shared Library Segment is established to include the MAPSE functional domain. These segments are again referenced at task establishment time of the KAPSE Loader.

3.4.1.2.5 Kernel Requests

A Kernel request from a MAPSE process is supported through the send message function and the trap wait condition in the task status word. This enables a MAPSE process (U-task) to place a message on the task queue of the Kernel process (E-task). The message is formatted to comply with the interface of the Request_Kernel facility specified by the Request Director.

3.4.1.2.6 MAPSE Process Execution

The execution of a MAPSE process is performed by the Kernel process using the load task function to load the KAPSE Loader and the start function to commence its execution. The name of the Load Object to be executed as a MAPSE process is supplied in the start options to the KAPSE Loader. The

Process Administrator in the Kernel process controls subsequent execution of the MAPSE process as a monitor task through the suspend, change priority, release and end the task functions.

3.4.1.2.7 MAPSE Events

A task queue is defined in the Kernel process to receive OS/32 events that are of significance to the execution of the MAPSE. Through this task queue, device interrupts, data transfer completions, interval time expirations and Kernel requests are received by the Event Monitor for processing by the Kernel process. The task queue for the MAPSE is defined at task establishment time of the Kernel process.

3.4.1.2.8 Interactive Communication

Interactive communication with the MAPSE is supported through terminal devices that are configured in the Kernel process when it is loaded. For these MAPSE terminal devices, a MAPSE terminal handler is made available to the OS/32 system that directs input to the Kernel process for processing. This processing may then recognize the editing and attention or break in characters that are significant to the MAPSE.

3.4.1.2.9 Standard Quantum of Time

The standard quantum of time for MAPSE process execution is provided in the Kernel process using the timer management functions. Expiration of the quantum of time results in a parameter entry on the task queue for the Kernel process that is available to the Event Monitor.

3.4.1.2.10 Low Level I/O

Low Level I/O is provided through the Kernel process by its status as an E-task. It may issue through the I/O Dispatcher the bare disk I/O functions read and write. Availability of this facility can be provided to a MAPSE process when necessary. The Ada constructs RECEIVE_CONTROL and SEND_CONTROL support is adapted to use the trap generating device functions. Trap generating device handlers can be made available to the OS/32 system to interface with the Kernel process.

3.4.1.2.11 KAPSE Data Base I/O

Support for the KAPSE Data Base is provided by indexed and contiguous file structures. The Volume Manager in the Kernel process maps a data base object to the required file structure. The allocate, assign, close, delete, read and write functions are used to perform file manipulation. The I/O Dispatcher may use the I/O proceed request to achieve asynchronous data transfers. Completion of a data transfer is recognized through a parameter block on the task queue of the Kernel process.

3.4.1.2.12 Ada Tasks

Ada tasks are supported by multiprogramming within a MAPSE process (U-task). The user SVC may be used to facilitate the implementation if the portability of the Task Manager can be maintained. The Kernel process (E-task) is used to schedule execution of the MAPSE process using the suspend and release functions. The full potential of the KFW design is constrained by restrictions that are presented by the send message and load task status word functions.

3.4.1.2.13 KAPSE Loader

The KAPSE Loader is created as a U-task and is established to reference the Global Task Common Segments and Shared Library Segments loaded through the Kernel process. Execution of the KAPSE Loader causes the specified MAPSE Load Object file to be read into the impure segment, and prepared for execution as a MAPSE process. When the KAPSE Loader is requested to load a MAPSE tool the appropriate Shared Library Segment is referenced for execution.

3.4.2 System Parameters

3.4.2.1 IBM VM/370

The system parameter that may change the operation of the MAPSE in the VM/370 system are those reconfiguration options for a virtual machine. These options are documented in IBM publication GC20-1757-2, Virtual Machine Facility/370 Features Supplement.

3.4.2.2 Interdata 8/32

The system parameters that may change the operation of the MAPSE in the Interdata OS/32 system include:

1. The number of Shared Library Segments
2. The number and size of Global Task Common Segments
3. The maximum size of a U-task
4. The maximum number of concurrently executing U-tasks
5. The E-task priority
6. The rolling of U-tasks that execute MAPSE tools
7. The size of the E-task queue
8. The devices to be assigned to the logical units of the E-task
9. The availability of the Spooler task
10. The availability of MAPSE supplied trap generating device handlers
11. The availability of a secondary file directory.

3.4.3 System Capabilities

3.4.3.1 IBM VM/370

In adapting the KFW design to VM/370 no contraints have been currently identified that are major hinderances to the implementation.

3.4.3.2 Interdata 8/32

In adapting the KFW design to the Interdata OS/32 system, the following constraints have been identified as potential hinderances to the implementation:

1. Insufficient protection control over the use of Global Task Common Segments
2. Exclusion of executable code in Global Task Common Segments
3. Lack of a system feature to change the purity status of a U-task segment
4. Lack of a system feature to modify the task status word of a U-task by another task

5. A system feature must be requested to transfer between execution states
6. The transfer between execution states does not optionally wait the directing task

During detailed design, every effort will made to minimize the impacts of these constraints on the KFW interfaces.

SECTION 4 - QUALITY ASSURANCE PROVISIONS

4.1 INTRODUCTION

This section contains the requirements for verification of the performance of the KFW. The test levels, verification methods, and test requirements for the detailed functional requirements in Section 3 are specified in this section. The verification requirements specified here shall be the basis for the preparation and validation of detailed test plans and procedures for the KFW. Testing shall be performed at the subprogram, program (CPCI), system integration, and acceptance test levels. The performance of all tests, and the generation of all reports describing test results, shall be in accordance with the Government-approved CPDP and the Computer Program Test Procedures.

The verification methods that shall be used in subprogram and program testing include the methods described below:

1. Inspection - Inspection is the verification method requiring visual examination of printed materials, such as source code listings, normal program printouts, and special printouts not requiring modification of the CPCI. This might include inspection of program listings to verify proper program logic flow.

2. Analysis - Analysis is the verification of a performance or design requirement by examination of the constituent elements of a CPCI. For example, a parsing algorithm might be verified by analysis.

3. Demonstration - Performance or design requirements may be verified by visual observation of the system while the CPCI is executing. This includes direct observance of all display, keyboard, and other peripheral devices required for the CPCI.

4. Review of Test Data - Performance or design requirements may be verified by examining the data output when selected input data are processed. For example, a review of hardcopy test data might be used to verify that the values of specific parameters are correctly computed.

5. Special Tests - Special tests are verification methods other than those defined above and may include testing one functional capability of the CPCI by observing the correct operation of other capabilities.

These verification methods shall be used at various levels of the testing process. The levels of testing to be performed are described in the paragraphs below. Data obtained from previous testing will be acceptable in lieu of testing at any level when certified by CSC/SEA and found adequate by the RADC representative. Any test performed by CSC/SEA may be observed by RADC representatives whenever deemed necessary by RADC.

Table 4-1 specifies the verification method for each functional requirement given in Section 3 of this specification. The listing in Table 4-1 of a Section 3 paragraph defining a functional requirement implies the listing of any and all subparagraphs. The verification methods required for the subparagraphs are included in the verification methods specified for the functional requirement. Acceptance test requirements are discussed in Paragraph 4.3.

4.1.1 Subprogram Testing

Following unit testing, individual modules of the KFW shall be integrated into the evolving CPCI and tested to determine whether software interfaces are operating as specified. This integration testing shall be performed by the development staff in coordination with the test group. The development staff shall ensure that the system is integrated in accordance with the design, and the test personnel shall be responsible for the creation and conduct of integration tests.

4.1.2. Program (CPCI) Testing

This test is a validation of the entire CPCI against the requirements as specified in this specification.

CPCI testing shall be performed on all development software of the KFW. This specification presents the performance criteria which the developed CPCI must satisfy. The correct performance of the KFW will be verified by testing its major functions. Successful completion of the program testing that the majority of programming errors have been eliminated and that the program is ready for system integration. The method of verification to be used in CPCI testing shall be review of test data. CPCI testing shall be performed by the independent test team.

4.1.3. System Integration Testing

System integration testing involves verification of the integration of the KFW with other computer programs and with equipment. The integration tests shall also verify the correctness of man/machine interfaces, and demonstrate functional completeness and satisfaction of performance requirements.

System integration testing shall begin in accordance with the incremental development procedures as stated in the CPDP. Final system integration shall occur subsequent to the completion of all the CPCIs comprising the MAPSE system. Two major system integration tests shall be performed: one for the IBM VM/370 implementation and one for the Interdata 8/32 implementation. The method of verification used for system integration testing shall be the review of test data.

The test team shall be responsible for planning, performing, analyzing monitoring, and reporting the system integration testing.

4.2 TEST REQUIREMENTS

Quality assurance tests shall be conducted to verify that the KFW performs as required by Section 3 of this specification. Table 4-1 specifies the methods that shall be used to verify each requirement. The last column refers to a brief description of the specified types of verification as given below. Test plans and procedures shall be prepared to provide details regarding the methods and processes to be used to verify that the developed CPCI performs as required by this specification. These test plans and procedures shall contain test formulas, algorithms, techniques, and acceptable tolerance limits, as applicable.

SECTION	TITLE	INSP.	ANAL.	DEMO.	DATA.	SECTION NO.
3.3.1	KAPSE Initiator				X	4.2
3.3.2	Logon Utility				X	4.2
3.3.3	Request Director				X	4.2
3.3.4	KAPSE Terminator				X	4.2
3.3.5	Process Administrator				X	4.2
3.3.6	Task Manager				X	4.2
3.3.7	Context Manager				X	4.2
3.3.8	Event Monitor				X	4.2
3.3.9	Volume Manager				X	4.2
3.3.10	I/O Dispatcher				X	4.2
3.3.11	KFW Loader				X	4.2

4.2 TEST REQUIREMENTS

All programs described in Table 4-1 will be tested using driver programs and examining output data. Drivers shall be written to generate input data and to log output data. Test input scripts and expected test output shall be developed by test personnel in accordance with subprogram and program specifications. Testing shall consist of comparing expected output data with test output data.

4.3 ACCEPTANCE TEST REQUIREMENTS

Acceptance testing shall involve comprehensive testing at the CPCI level and at the system level. The CPCI acceptance tests shall be defined to verify that the KFW satisfies its performance and design requirements as specified in this specification. System acceptance testing shall test that the MAPSE satisfies its functional requirements as stated in the System Specification. Acceptance testing shall be performed by review of test data.

These tests shall be conducted by the CSC/SEA team and formally witnessed by the Government representatives. Satisfactory performance of both CPCI and system acceptance tests shall result in the final delivery and acceptance of the MAPSE system.

SECTION 5 - DOCUMENTATION

5.1 GENERAL

The documents that will be produced during the implementation phase in association with the KFW are:

1. Computer Program Development Specification (Type B5) - Update
2. Computer Program Product Specification
3. Computer Program Listings
4. Maintenance Manual
5. User's Manual
6. Rehostability Manual

5.1.1 Computer Program Development Specification

The final KFW B5 Specification will be prepared in accordance with DI-E-30139 and submitted 30 days after the start of Phase II.

5.1.2 Computer Program Product Specification

A type C5 Specification shall be prepared during the course of Phase II in accordance with DI-E-30140. This document will be used to specify the design of the KFW and the development approach implementing the B5 specification. This document will provide the detailed description that will be used as the baseline for any Engineering Change Proposals. A single C5 will be produced for the KFW with different sections addressing the dependencies on the two host computers.

5.1.3 Computer Program Listings

Listings will be delivered that are the result of the final compilation of the accepted KFW. Each compilation unit listing will contain the corresponding source, cross-reference, and compilation summary. The source listing will contain the source lines from any Included source objects.

5.1.4 Maintenance Manual

A KFW Maintenance Manual will be prepared in accordance with DI-M-30422 to supplement the C5 and compilation listings sufficiently to permit the KFW to be easily maintained by personnel other than the developers. The documentation will be structured to relate quickly to program source. The procedures required for debugging and correcting the KFW, along with debugging aids that have been incorporated as an integral part of the KFW, will be described and illustrated. Sample scripts for compiling KFW components, for relinking the KFW in parts or as a whole, and for installing new releases will be supplied. Separate sections will be provided to address modifications that have been incorporated to tailor the KFW to individual hosts.

5.1.5 User`s Manual

A User`s Manual shall be prepared in accordance with DI-M-30421, which will contain all information necessary for the operation of the KFW. Because of the virtual user interface presented to the KFW, a single manual is sufficient for all host computers. Information relevant to specific hosts will be contained in an appendix. Supplemental information will be supplied to assist the user in locating and correcting KFW errors.

5.1.6 Rehostability Manual

In accordance with R&D-137-RADC and R&D-138-RADC, a manual will be prepared that describes step-by-step procedures for rehosting the KFW on a different computer.

APPENDIX A - KFW VIRTUAL INTERFACE PACKAGES

```
package REQUEST_DIRECTOR is

    type REQUEST_KIND is (Some_Facility,...);
    type REQUEST_SHAPE (Shape : REQUEST_KIND) is

         record
              Request : REQUEST_KIND := Shape;
                   case Shape is
                        when REQUEST_KIND`FIRST = >
                        -- Space for actual parameters
                        when REQUEST_KIND`LAST = >
                   end case;
         end record;

    type REF_REQUEST_SHAPE is access REQUEST_SHAPE;

    generic
         type REQUEST_PARAMETER is private;
         type REF_REQ_PAR is access REQUEST_PARAMETER;

    procedure Request_Kernel
         (Addr_RPL : REF_REQ_PAR);

end REQUEST_DIRECTOR;
```

150

```
package PCB_SHAPE is

    type PROCESS_CONTROL_BLOCK;
    type REF_PCB is access PROCESS_CONTROL_BLOCK;
    type PROCESS_CONTROL_BLOCK is
         record
              Load_Object_Name    :    LOAD_NAME;
              Priority            :    PRIORITY_VALUE;
              Level               :    PROCESS_LEVEL;
              Privileges          :    PRIVILEGE;
              Classification      :    CLASSIFICATION_LEVEL;
              Owner_Id            :    OWNER_NAME;
              User_Id             :    USER_NAME;
              Process_Map         :    REF_PCM;
              Status              :    PROCESS_STATUS;
              Time_Used           :    USAGE;
              Standard_Input      :    STD_IN;
              Standard_Error      :    STD_ERROR;
              Standard_Output     :    STD_OUT;
              Next_PD             :    REF_PCB;
              Previous_PD         :    REF_PCB;
              Parent_Params       :    REF_RPL;
              Child_Process       :    REF_PCB;
              Next_Sibling        :    REF_PCB;
              Wait_Process        :    REF_PCB;
              Task_Quota          :    TASK_LIMIT;
              Time_Quota          :    TIME_LIMIT;
              Tasks_Active        :    MAX_TASKS;
              Tasks_Ready         :    MAX_TASKS;
              Tasks_Terminated    :    MAX_TASKS;
              Tasks_Waiting       :    MAX_TASKS;
         end record;

end PCB_SHAPE;
```

```
package DTR_SHAPE is

    type DEPENDENT_TASK_RECORD;

    type REF_DTR is access DEPENDENT_TASK_RECORD;

    type DEPENDENT_TASK_RECORD is
         record
              Guardian_Task            :    REF_TCB;
              First_Dependent          :    REF_TCB;
              Next_DTR                 :    REF_DTR;
              Activate_Count,
              Dependent_Count,
              Terminate_Count          :    NATURAL;

         end record;

end DTR_SHAPE;
```

```
package ALTERNATIVE_TABLE_SHAPE is

    type ALTERNATIVE_KIND is

         (Entry_Arm, Constant_Family_Arm, Variable_Family_Arm,
          Constant_Delay_Arm, Variable_Delay_Arm,
          Else_Arm, Terminate_Arm);

    subtype EXCLUSIVE_ALTERNATIVE is ALTERNATIVE_KIND

    range Constant_Delay_Arm .. Terminate_Arm;

    type ALTERNATIVE_TABLE_ENTRY (Exclusive_Arm:  EXCLUSIVE_ALTERNATIVE) is
         record
              Accept_Body                  :    DOMAIN_OFFSET;
              Alternative_Arm              :    ALTERNATIVE_KIND;
              Entry_No                     :    ENTRY_VALUE;
              Constant_Entry_Index         :    INDEX_VALUE;
              Family_Alternative_Arm       :    ARM_VALUE;
              Null_Guard,
              Null_Accept                  :    BOOLEAN;
              case Exclusive_Arm is

                   when Constant_Delay_Arm BAR Variable_Delay_Arm = >
                     Constant_Delay        :    DURATION;
                     Delay_Alternative_Arm
                                           :    ARM_VALUE;

                   when Terminate_Arm BAR Else_Arm = >
                     null;

              end case;

         end record;

    type ALTERNATIVE_TABLE is array (1 .. Max_Arm_Value)
      of ALTERNATIVE_TABLE_ENTRY (Exclusive_Arm);

    type ALTERNATIVE_GUARDS is array (1 .. Max_Arm_Value) of BOOLEAN;

    type VARIABLE_DELAYS is array (1 .. Max_Delay_Arm) of DURATION;

    type VARIABLE_FAMILY is array (1 .. Max_Family_Arm) of INDEX_VALUE;

end ALTERNATIVE_TABLE_SHAPE;
```

```
with DTR_SHAPE, ALTERNATIVE_TABLE_SHAPE;
use DTR_SHAPE, ALTERNATIVE_TABLE_SHAPE;

package TCB_SHAPE is

    type TASK_CONTROL_BLOCK;

    type REF_TCB is access TASK_CONTROL_BLOCK;

    type TASK_CONTROL_BLOCK is
         record
              Task_PCB             :    REF_PCB;
              Guardian             :    REF_DTR;
              Static_Context       :    REF_ESC;
              Context_Map          :    REF_PCM;
              Initial_State        :    TASK_IEP;
              Elaborator           :    REF_DTR;

              Activated, Created,
              Suspended,
              Terminated, Kernel,
              Running, Ready       :    BOOLEAN;
              Kernel_RPL           :    REF_RPL;
              Kernel_Exception     :    -- To be defined
              Exception_Name       :    EXCEPTION_VALUE;
              Failure_Task         :    REF_TCB;
              Failure_Exception    :    -- To be defined

              --Ready Task Queue

              Next_Ready,
              Prev_Ready           :    REF_TCB;
              Static_Priority,
              Run_Priority         :    PRIORITY_VALUE;

              --Suspended Task Queue

              Next_Suspended,
              Prev_Suspended       :    REF_TCB;
              Wait_Condition       :    -- To be defined
              Wait_Time            :    DURATION;

              --Entry call Queue Links in Calling Task

              Next_Caller,
              Prev_Caller,
              Service_Task         :    REF_TCB;
              Service_Entry        :    ENTRY_VALUE;
              Synchronized         :    BOOLEAN;
              Entry_Call_Parameters
                                   :    -- To be defined
              --Nested Accept Stack in Task being serviced.

              This_Caller,
              Prev_Caller         :    REF_TCB;
              Service_Priority    :    PRIORITY_VALUE;

              --Entry Queue in Service Task.

              Open_Entries        :    OPEN_QUEUE (1 .. Max_Entries);
              Head,
              Tail                :    ENTRY_QUEUE (1 .. Max_Entries);
              Entry_Start_Index   :    ENTRY_INDEX;
              Queue_Lengths       :    array (1 .. Max_Entries) of INTEGER;
              Interrupt_Entries   :    -- To be defined
              Interrupt_ECB_Queue :    INTERRUPT_QUEUE;
              This_Select         :    -- To be defined
              Open_Alternatives   :    ALTERNATIVE_GUARDS;
              Delay_Alternatives  :    VARIABLE_DELAYS;
              Family_Alternatives :    VARIABLE FAMILY;

    end record;

end TCB_SHAPE;
```

```
with REQUEST_DIRECTOR, PCB_SHAPE, TCB_SHAPE;
use REQUEST_DIRECTOR, PCB_SHAPE, TCB_SHAPE;
package PROCESS_ADMSTR is

    procedure Start_Process
         (Load_Object_Name      :         LOAD_NAME;
          Process_Priority      :         PRIORITY_VALUE;
          Process_In_Params     :         IN_PROCESS_PARAMS;
          Process_InOut_Params  : in out  INOUT_PROCESS_PARAMS;
          Process_Out_Params    : out     OUT_PROCESS_PARAMS;
          Addr_PCB              : out     REF_PCB);

    procedure Finish_Process
         (Process_InOut_Params : IN_PROCESS_PARAMS;
          Process_Out_Params   : IN_PROCESS_PARAMS);

    procedure Suspend_Process
         (Addr_PCB : REF_PCB;
          Addr_TCB : REF_TCB);

    procedure Terminate_Process_Task
         (Addr_PCB : REF_PCB;
          Addr_TCB : REF_TCB);

    procedure Ready_Process
         (Addr_PCB : REF_PCB);
    procedure Wait_Process
         (Addr_PCB       : REF_PCB;
          Addr_PCB       : REF_PCB;
          Wait_Condition : PROCESS_STATUS);

    procedure Save_Process
         (Addr_PCB         : REF_PCB;
          Load_Object_Name : LOAD_NAME);

    procedure Resume_Process
         (Addr_PCB : REF_PCB);

    procedure Switch_Process_Task
         (Addr_PCB : REF_PCB;
          Addr_TCB : REF_TCB);

    procedure Terminate_Process
         (Addr_PCB : REF_PCB);

    procedure Rank_Process
         (Addr_PCB : REF_PCB;
          Addr_TCB : REF_TCB);

    procedure Read_PCB
         (Addr_PCB  : REF_PCB;
          Addr_VPCB : REF_VPCB);

    procedure Delete_Process
         (Addr_PCB : REF_PCB);

    procedure Write_PCB
         (Addr_PCB  : REF_PCB;
          Addr_VPCB : REF_VPCB);

end PROCESS_ADMSTR;
```

```
with REQUEST_DIRECTOR, TCB_SHAPE;
use REQUEST_DIRECTOR, TCB_SHAPE;

package TASK_MANAGER is

    procedure Create_Task
         (Addr_TCB          :  REF_TCB;
          Dep_Header_Record :  REF_DTR;
          Addr_ESC          :  REF_ESC;
          Task_Priority     :  PRIORITY_VALUE;
          Task_IEP          :  TASK_IEP;
          TCB_Alt           :  TCB_ALT);

    procedure Schedule_Task
         (Addr_TCB :  REF_TCB);

    procedure Delay_Task
         (Time_Delay :  DURATION);

    procedure Accept_Entry
         (Entry_No    :  ENTRY_VALUE;
          Null_Accept :  BOOLEAN);

    procedure Accept_Entry_Family
         (Entry_No    :  ENTRY_VALUE;
          Entry_Index :  INDEX_VALUE;
          Null_Accept :  BOOLEAN);

    generic
         type PARAMETER_LIST is private;
         type PARAMETER_LIST_ADDRESS
              is access PARAMETER_LIST;

    procedure Entry_Call
         (Addr_TCB   :  REF_TCB;
          Entry_No   :  ENTRY_VALUE;
          Parameters :  PARAMETER_LIST_ADDRESS);

    generic
         type PARAMETER_LIST is private;
         type PARAMETER_LIST_ADDRESS
              is access PARAMETER_LIST;

    procedure Entry_Call_Family
         (Addr_TCB    :  REF_TCB;
          Entry_No    :  ENTRY_VALUE;
          Entry_Index :  INDEX_VALUE;
          Parameters  :  PARAMETER_LIST_ADDRESS);
```

```
generic
     type PARAMETER_LIST is private;
     type PARAMETER_LIST_ADDRESS
          is access PARAMETER_LIST;

procedure Conditional_Entry_Call
     (Addr_TCB   :       REF_TCB;
      Entry_No   :       ENTRY_VALUE;
      Parameters :       PARAMETER_LIST_ADDRESS;
      Condition  :   out BOOLEAN);

generic
     type PARAMETER_LIST is private;
     type PARAMETER_LIST_ADDRESS
          is access PARAMETER_LIST;

procedure Conditional_Entry_Call_Family
     (Addr_TCB    :       REF_TCB;
      Entry_No    :       ENTRY_VALUE;
      Entry_Index :       INDEX_VALUE;
      Parameters  :       PARAMETER_LIST_ADDRESS;
      Condition   :   out BOOLEAN);

generic
     type PARAMETER_LIST is private;
     type PARAMETER_LIST_ADDRESS
          is access PARAMETER_LIST;

procedure Timed_Entry_Call
     (Addr_TCB   :       REF_TCB;
      Entry_No   :       ENTRY_VALUE;
      Parameters :       PARAMETER_LIST_ADDRESS;
      Condition  :   out BOOLEAN);

generic
     type PARAMETER_LIST is private;
     type PARAMETER_LIST_ADDRESS
          is access PARAMETER_LIST;

procedure Timed_Entry_Call_Family
     (Addr_ TCB   :       REF_TCB;
      Entry_No    :       ENTRY_VALUE;
      Entry_Index :       INDEX_VALUE:
      Parameters  :       PARAMETER_LIST_ADDRESS;
      Condition   :   out BOOLEAN);

procedure End_Rendezvous;

procedure Selective_Alternative
     (Select_Table :  ALTERNATIVES_TABLE);
     procedure Wait_Dependent_Tasks
          (Dependent_Tasks :  REF_DTR);

     procedure Terminate_Task;

     procedure Abort_Task
          (Addr_TCB :  REF_TCB);

     procedure Fail_Task
          (Addr_TCB : REF_TCB);

     procedure Set_Interrupt
          (Addr_TCB    : REF_TCB;
           Entry_No    :  ENTRY_VALUE;
           Entry_Index :  INDEX_VALUE;
           Interrupt   :  INTERRUPT_NAME);

     procedure Accept_Exception
          (Exception :  EXCEPTION_VALUE);

     function Attribute_Terminated
          (Addr_TCB :  REF_TCB) return TERMINATE_STATE;

     function Attribute_Priority
          (Addr_TCB :  REF_TCB) return PRIORITY_VALUE;

     function Attribute_Storage
          (Addr_TCB :  REF_TCB) return STORAGE_UNITS;

     function Attribute_Count
          (Addr_TCB :  REF_TCB) return INTEGER;

end TASK_MANAGER;
```

```
with PCB_SHAPE; use PCB_SHAPE;
package CONTEXT_MANAGER is

    procedure Allocate_Domain
         (Addr_PCB      :      REF_PCB;
          Map_Index     :      INDEX_VALUE;
          Domain_Length :      INTEGER;
          Addr_Domain   :  out REF_DOMAIN);

    procedure Release_Domain
         (Addr_PCB  :  REF_PCB;
          Map_Index :  INDEX_VALUE);

    procedure Find_Domain
         (Load_Object_Name :      LOAD_NAME;
          Addr_Domain      :  out REF_DOMAIN);

    procedure Load_Domain
         (Load_Object_Name :      LOAD_NAME;
          Addr_Domain      :  out REF_DOMAIN);

    procedure Read_Domain
         (Addr_Domain  :  REF_DOMAIN;
          Storage_Unit :  DOMAIN_OFFSET;
          Unit_Length  :  INTEGER;
          Addr_Buffer  :  REF_BUFFER);

    procedure Write_Domain
         (Addr_Domain  :  REF_DOMAIN;
          Storage_Unit :  DOMAIN_OFFSET;
          Unit_Length  :  INTEGER;
          Addr_Buffer  :  REF_BUFFER);

end CONTEXT_MANAGER;
```

```
with ECB_SHAPE; use ECB_SHAPE;
package EVENT_MONITOR is

    procedure Set_Event
         (Addr_ECB : REF_ECB);

    procedure Cancel_Event
         (Addr_ECB : REF_ECB);

    procedure Wait_Event
         (Addr_ECB : REF_ECB);

    procedure Raise_Event
         (Event_Name : EVENT);

end  EVENT_MONITOR;
```

```
with CCB_SHAPE, DCB_SHAPE;
use CCB_SHAPE, DCB_SHAPE;

package IO_DISPATCHER is

    proceaure Initiate_LowLevel_IO
         (Addr_DCB : REF_DCB);

    procedure Initiate_Object_IO
         (Addr_CCB : REF_CCB);

end IO_DISPATCHER;
```

```
with IO_DISPATCHER, OCB_SHAPE;
use IO_DISPATCHER, OCB_SHAPE;

package VOLUME_MANAGER is

    procedure Create_Host_Object
         (Addr_OCB : REF_OCB);

    procedure Open_Host_Object
         (Addr_OCB : REF_OCB);

    procedure Close_Host_Object
         (Addr_OCB : REF_OCB);

    procedure Delete_Host_Object
         (Addr_OCB : REF_OCB);

    procedure Write_Increment
         (Addr_OCB : REF_OCB);

    procedure Read_Increment
         (Addr_OCB : REF_OCB);

end VOLUME_MANAGER;
```

Volume 2

COMPUTER PROGRAM DEVELOPMENT SPECIFICATION

(TYPE B5)

COMPUTER PROGRAM CONFIGURATION ITEM

KAPSE Data Base System

Prepared for

Rome Air Development Center
Griffiss Air Force Base, NY 13441

Contract No. F30602-80-C-0292

166

TABLE OF CONTENTS

167

LIST OF ILLUSTRATIONS

169

SECTION 1 - SCOPE

1.1 IDENTIFICATION

This document presents the Computer Program Development Specification (Type B5) for the Computer Program Configuration Item (CPCI) called the Kernal Ada Programming Support Environment (KAPSE) Data Base System (KDBS). This CPCI provides the logical repository for all system- and user-generated data as well as the requisite data base management system functions.

This specification provides the performance, design, and testing requirements for the KDBS. Section 3 presents the performance and design requirements. Section 4 presents the testing and quality assurance requirements. This specification, after approval by Rome Air Development Center (RADC), will serve as the development baseline for the KDBS.

1.2 FUNCTIONAL SUMMARY

The KAPSE Data Base System (KDBS) is the cornerstone of the MAPSE and provides facilities for maintaining the different kinds of data needed in developing computer systems. The KDBS includes not only the facilities for storing the data, but also the data base management functions necessary to manipulate the data in a controlled manner. All data are represented as objects and attributes. The KDBS provides facilities to ensure correctness, consistency, security, and flexibility.

SECTION 2 - APPLICABLE DOCUMENTS

2.1 PROGRAM DEFINITION DOCUMENTS

1. Requirements for Ada Programming Support Environment - STONEMAN, United States Department of Defense, February 1980.
2. Reference Manual for the Ada Programming Language, United States Department of Defense, July 1980.
3. Revised Statements of Work for Ada Integrated Environment, Rome Air Development Center, 26 March 1980.

2.2 INTER-SUBSYSTEM SPECIFICATIONS

4. Specification for the Ada Integrated Environment.
5. Volume 1, Computer Program Development Specification for CPCI KAPSE Framework.
6. Volume 3, Computer Program Development Specification for CPCI APSE Command Language Interpretor.
7. Volume 4, Computer Program Development Specification for CPCI MAPSE Configuration Management System.
8. Volume 5, Computer Program Development Specification for CPCI Ada Compiler.
9. Volume 6, Computer Program Development Specification for CPCI MAPSE Linker.
10. Volume 7, Computer Program Development Specification for CPCI MAPSE Editor.
11. Volume 8, Computer Program Development Specification for CPCI MAPSE Debugger.

2.3 MILITARY SPECIFICATIONS AND STANDARDS

12. MIL-STD-483, Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs, 1 June 1971.
13. MIL-STD-490, Specification Practices, 30 October 1968.

2.4 MISCELLANEOUS DOCUMENTS

14. Ada Support System Study (for the United Kingdom Ministry of Defence), Systems Designers Limited, Software Sciences Limited, 1979-1980.

15. Feiertag, R. J., and E. I. Organick, The Multics Input-Output System, Proc. Third Symposium on Operating Systems Principles, October 1971.

16. Fisher, David A., Design Issues for Ada Program Support Environments, Science Applications Inc., SAI-81-289-WA, October 1980.

17. Ritchie, D. M., and K. Thompson, The UNIX Time-Sharing System, The Bell System Technical Journal, Vol. 57, No. 6, Part 2, July-August 1978.

18. Rochkind, M. J., The Source Code Control System, IEEE Transactions on Software Engineering, SE-1, December 1975.

19. Thompson, K., UNIX Implementation, The Bell System Technical Journal, Vol. 57, No. 6, Part 2, July-August 1978.

20. Dolotta, T. A., S. B. Olsson, and A. G. Petruccelli, ed., UNIX User`s Manual, Release 3.0, Bell Telephone Laboratories, June 1980.

SECTION 3 - REQUIREMENTS

3.1 INTRODUCTION

This section presents the design and performance requirements of the KDBS. The visible specifications for the KDBS, available to all MAPSE components, are incorporated in the KAPSE virtual interface and are presented as Appendix A of this specification. Data base requirements, as specified in the Statement of Work (SOW) and the System Specification (Type A) are included by reference.

3.1.1 General Description

The KDBS, the central element of the MAPSE system, maintains all user- and system-generated data and provides support to project management and configuration management tools, source program libraries, and provides basic object manipulation facilities. The KDBS is structured so that relationships between objects in the data base are maintained, but the KAPSE data base does not impose restrictions on the format of the information stored in the object.

The KDBS is designed to provide for the maintenance of data objects in a machine-independent manner. These objects include Ada source text, relocatable, executable, project documentation, configuration and partitions objects. The KDBS provides for creation, deletion, and modification of these data objects as well as support for configurations, versions, and partitions. The KDBS also supports Ada standard Input/Output (I/O) as part of the Ada Run-Time Support package.

3.1.2 Peripheral Equipment Identification

The KDBS has been designed to be portable, and therefore is not dependent on host computer characteristics. Services normally provided by host systems shall be supplied by the KAPSE Framework (KFW).

3.1.3 Interface Identification

This paragraph specifies the functional relationship of the KDBS to other MAPSE components. Paragraph 3.2.4 describes the interfaces between the KDBS and the other computer programs in the MAPSE system. The KDBS interfaces directly with the KFW and the KAPSE virtual interface. Thus, the KDBS has a machine independent interface to the host and a consistent interface to the rest of the MAPSE system.

The KDBS has two levels of interfaces to the MAPSE system, those that are made available to other components of the MAPSE and those required by the KDBS in order to process requests and remain transportable.

3.1.3.1 Visible Interfaces

The KDBS has facilities available to the users of MAPSE which are made visible through the virtual interface. These facilities are packaged into two logical areas, the Ada Run-time Support Package and the KDBS Utility Package. The Ada Run-time Support Package contains those functions necessary to support the Ada standard I/O facilities specified in the Ada language. The KDBS Utility Package provides those functions necessary to manipulate and control the KDBS and its objects. Specifications for the KDBS Utility Package are included as Appendix A. The separation between the two packages are made because of their relation to the MAPSE system. The Ada Run-time Support Package is a portable package from the host implementation to the target machine, while those facilities found in the KDB Utility Package are not generally needed on the target machine.

3.1.3.2 KDBS Required Interfaces

These are the interfaces needed by the KDBS inorder to process a request made to it by MAPSE Process/Tasks. The only required interface the KDBS has to other components of the MAPSE are those of the KFW. The KFW is needed in mapping the logical I/O from the MAPSE Process/Tasks to the physical I/O on the host environment. The KDBS requires the ability to signal the KFW to suspend or reschedule MAPSE Process/Tasks requesting KDBS services.

3.1.4 Functional Identification

The major functional areas of the KDBS are:

1. Attribute support
2. Partition support
3. Access support
4. Ada input/output support

5. Version support
6. Archive support
7. Backup support.

3.2 FUNCTIONAL DESCRIPTION

This section provides an introduction to the functional capabilities of the KDBS. Detailed functional requirements are described in Paragraph 3.3.

3.2.1 Equipment Description

The KDBS is designed to be machine-independent. It will be one of the portable components of the MAPSE system. There are no special requirements imposed on the KDBS by either the IBM 370 or the Interdata 8/32.

3.2.2 Computer Input/Output Utilization

All input and output requirements of the KDBS are satisfied by the KFW. No special requirements are imposed on the KDBS. Information flow between the KDBS and the host computer system is handled entirely by the KFW.

3.2.3 Computer Interface Block Diagram

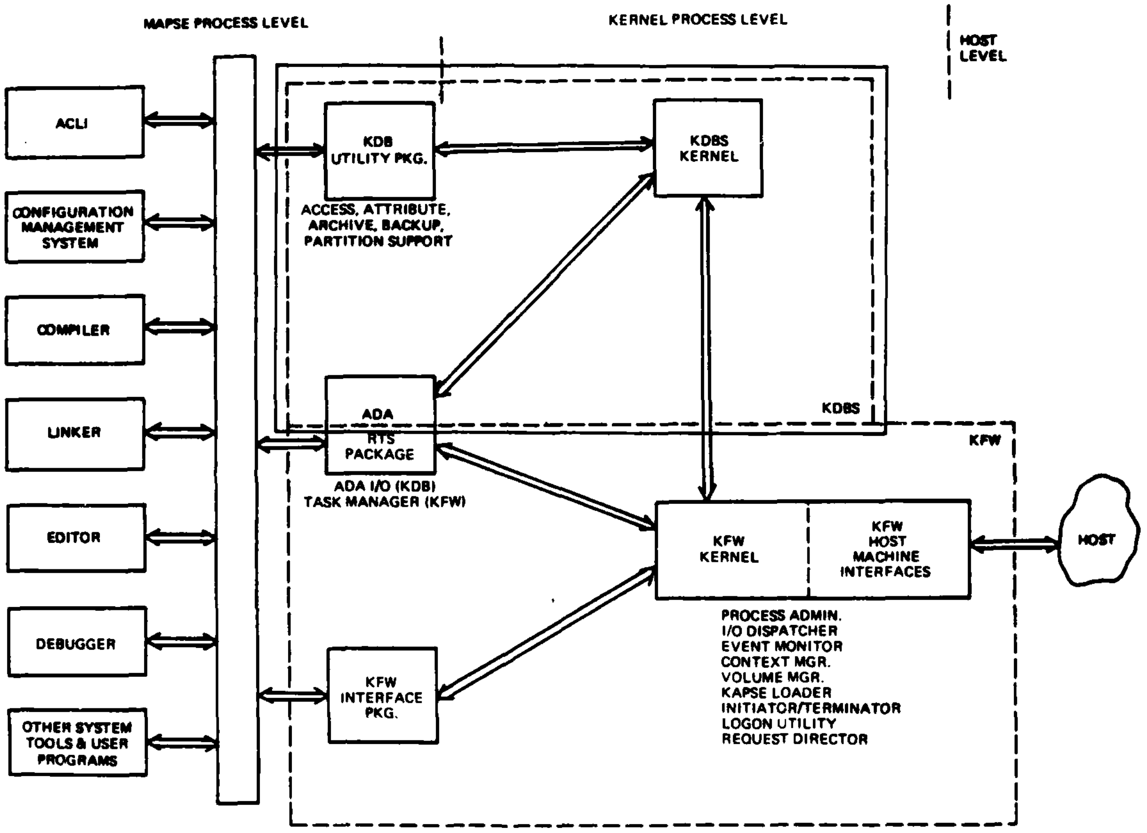
The functional interfaces between the KDBS and other MAPSE CPCIs are illustrated in Figure 3-1. Computer interfaces are given in the KFW B5 Specification.

3.2.4 Program Interfaces

The KDBS shall use host system facilities through its interface with the KFW at the KDBS Kernel level. Facilities of the KDBS shall be made available to MAPSE tools and user programs through the virtual interface. Functions of the KDBS available at each interface are also indicated in the figure.

The KDB Utility Package interfaces with the virtual interface and supplies support for access, attribute, archive, backup, version, and partition manipulation. The Ada I/O functions of the KDBS are supplied through the Ada Run-Time support package. KDBS I/O facilities are supplied through the Kernel level interface to the KFW.

The KDBS shall be written in Ada and therefore compilable by the Compiler.



TP NO. 021-2002-A

Figure 3-1. Interface Diagram

3.2.5 Function Description

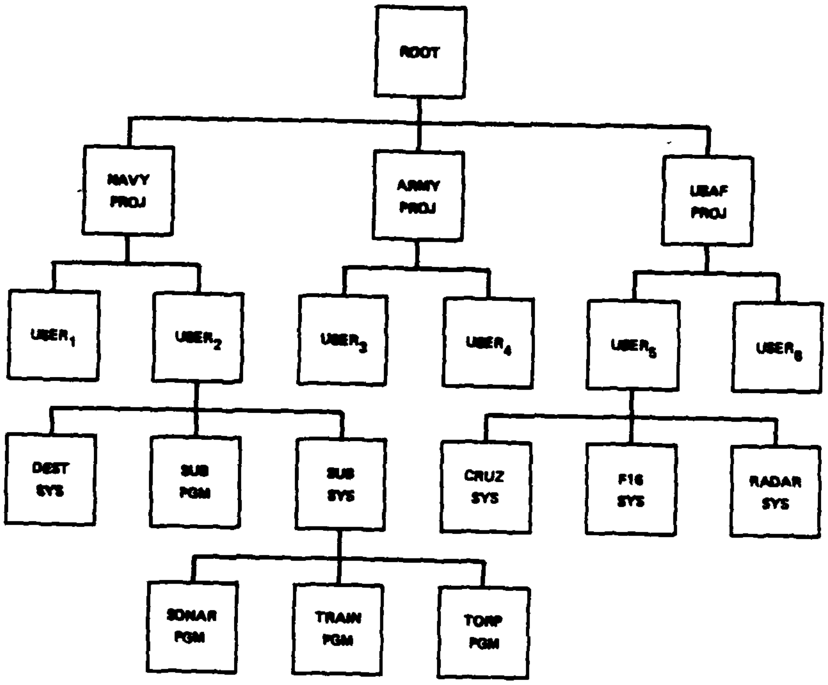
The KDBS provides administration and control of all user and system generated data within the MAPSE. In the KDB, all data is represented as objects. All objects have attributes and information content. Object names may have, in addition, category and version qualifiers. All objects are stored as a tree-structured hierarchy, the structure being induced by partitions. Access control features are permitted on both partitions and objects. The access control scheme allows partition access rights to control access for the entire subtree defined by the partition and thus, for efficiency purposes, indiviual objects rights need not be specified. The KDBS also supports virtualized I/O facilities through the Ada Run-time Support Package and provides the same access control facilities for an object that represents a device as for any other object in the data base. Backup, restore and archiving features are also an integral part of the KDBS design.

The following paragraphs describe the functional characteristics of these features.

3.2.5.1 Partitions

All objects are stored in a tree-structured hierarchy see (Figure 3-2). The structure of the hierarchy is induced via partitions, which provide a mapping from object names to the objects themselves see (Figure 3-3). Partitions are a special category of object used to provide logical groupings of other objects. A partition corresponds to the notion of a "directory" in other systems such as UNIX.

Each user registered to the MAPSE has an individually assigned home partition and may create additional subpartitions as needed. The KDBS maintains several predefined partitions for system use. One of these is the root partition. All objects in the KDB can be found by tracing a path through a chain of partitions until the desired object is reached. A fully specified object name identifies this chain of partitions. The syntax for this name is a sequence of partition names followed by the unadorned base name of the object, where the partition names are delimited with



NBS FTR - DWG NO. 6713A - 4

Figure 3-2. Tree-Structured Hierarchy

OBJECT NAME$_1$	HOST NAME$_1$
OBJECT NAME$_2$	HOST NAME$_2$
OBJECT NAME$_3$	HOST NAME$_3$
OBJECT NAME$_4$	HOST NAME$_4$
OBJECT NAME$_5$	HOST NAME$_5$
OBJECT NAME$_6$	HOST NAME$_6$
⋮	⋮
OBJECT NAME$_n$	HOST NAME$_n$

TP No. 021-1987-A

Figure 3-3. Partition Object

slashes (/). Full specification means that the partition chain begins at the root, which is indicated by an initial "/". The general form is thus:

$$/partition_1/partition_2/.../partition_n/base\_name$$

where $partition_{i+1}$ is a member of $partition_i$. A fully specified name is also called an absolute pathname. A partially specified pathname, or relative pathname, does not begin with an initial "/". The chain of partitions denoted by a relative pathname is traced beginning at, or relative to, the current working partition, which is identified as part of the environment of a process.

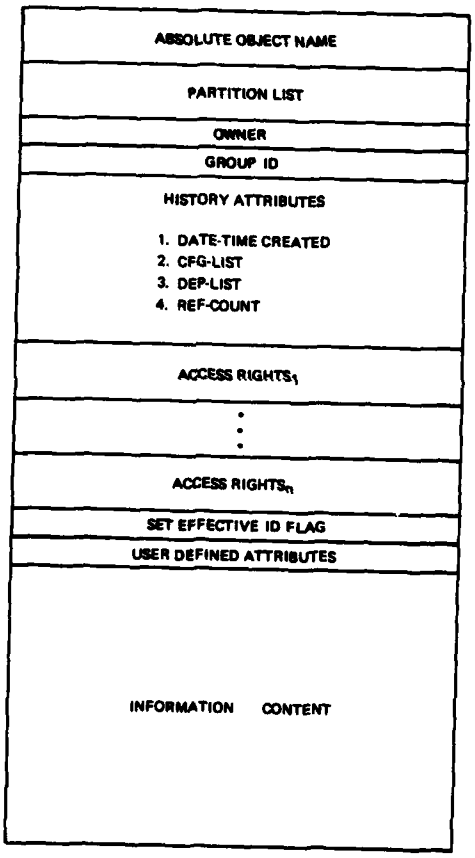
Any non-partition object may appear in several partitions under possibly different names. This feature is called linking, and a link in a partition maps a local name into an absolute pathname that identifies an object in a different partition. The local name thus serves as a synonym for the absolute pathname.

3.2.5.2 Objects

All KDB objects have attributes and information content see (Figure 3-4). Objects may also have a category (see Paragraph 3.2.5.3) and a version qualification. Different objects may have the same pathname; these are distinguishable only through qualification by category and version. The syntax for these qualifications is described in the indicated subsections.

Attributes supply additional meta-information about an object. Much of this information is required by the system in order to provide access control and configuration management, but user-defined attributes are also permitted. The user may define any new attributes that do not conflict in name with system-defined attributes. The value of a user-defined attribute must be a single character string or a list of character strings separated by semicolons (;).

The logical object structure depicted in Figure 3-4 is by no means the physical structure, which must be designed and optimized for each host. The system-defined attributes included in this logical structure are defined below. Some of these are optional, others are always present for all objects.



TP No. 021-1898-A

Figure 3-4. Object Structure

1. Name - contains the fully-qualified, unique name of the object. The name is composed of the absolute pathname of the object, the category of the object and the version of the object.

2. Partition List - a list of partitions that contain a link to this object. This list is of varying length and will often be empty.

3. Owner Id - the effective user-id of the process that created this object.

4. Group Id - the effective group-id of the process that created this object.

5. History Attributes - consists of four attributes:

   a. Date-time - the date-time when the object was last modified.

   b. Dependency List - a list of those objects referenced in configuring this object.

   c. Reference Count - the number of references made to this object, including references made in dependency lists and references from links.

   d. Configuration List - (optional) a list of those configurations that reference this object.

6. Access Rights - a list of users and groups along with their access rights to this object. An additional "default" entry exists to indicate the access rights of all other users. The access rights attribute may be empty. The <u>access permissions</u> for an object are computed as a function of the access rights for the individual object and the partition access rights for the containing partitions.

7. Partition Access Rights - (optional) may be supplied only for a partition object. Partition access rights have the same form as ordinary access rights but are used to control access to the entire subtree rooted by the partition.

8. Set Effective Ids - (optional) can only be set for XQT and CMD objects. Indicates when the program is loaded as a process, the effective user-id and group-id of the process are to be set to the user-id and group-id of the creating user.

3.2.5.3 Categories

The MAPSE has a number of system-defined object categories. The user is free to add new categories, but the system tools will ascribe special meaning to those listed below. A category name may be supplied as a qualifier to an object base name:

base_name`category_name

Names with the same base name but different category names denote different objects. Thus the category name will often be required to distinguish between object names. However, many system tools automatically supply a category for otherwise ambiguous names, based on context. Such automatic categorization is detailed in each of the system tool specifications. The system-defined categories are:

Help	HLP	Used to contain information for the Help facility of the APSE and contains the text for describing an object with the same base name.
Configuration	CFG	Used and maintained by the Configuration Management System; contains the information necessary to define and control the building of a configuration.
Data	DAT	The default category. All objects created without an explicit category are assigned the category DAT.
Device	DVC	Reserved for objects that serve to identify I/O devices. The information content of such objects is host-dependent.
Archive	ACV	Denotes an object that is used to maintain an archive. The archiving functions are described in Paragraph 3.2.5.5.

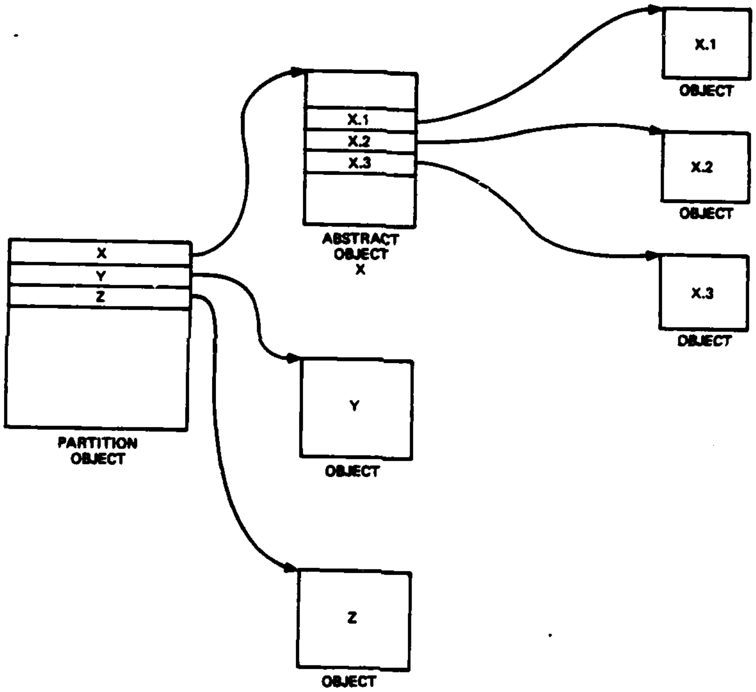
Command	CMD	Contains subprograms written in APSE Command Language (ACL). These are interpreted by the APSE Command Language Interpreter (ACLI).
Text	TXT	Contains data that can be processed with the Ada Text I/O Package. TXT objects are used to store Ada source modules.
List	LST	Contains listing output generated by system tools. In particular, Compiler listings are written into LST objects.
Relocatable	REL	Contains relocatable code.
Executable	XQT	Contains executable code.
Library	LIB	Denotes an Ada Program Library.
Partition	PTN	Denotes a KDB partition (see Paragraph 3.2.5.1).

3.2.5.4 Abstract Objects and Version Control

A version group is a set of objects that represent related iterations of a single abstract object. The name of the abstract object (the base name qualified by category) serves as a generic name for the version group. The abstract object serves as a directory for the objects in the version group. An object may be created initially as an abstract object, or may subsequently be converted to an abstract object.

A version group is tree-structured as shown in Figure 3-5. Each branch of the tree has a name that is unique within the tree. Each version along a branch has a number that is unique for that branch. Versions along a branch must be identified with monotonically increasing numbers. To denote a specific version of an abstract object, a version qualifier can be appended to the abstract object name. A version name can take one of three forms:

```
object_base_name`category_name.branch_name.version_number
```



TP No. 021-1988-A

Figure 3-5. Version Group

This is the fully-qualified form, and denotes a specific version.

object_base_name`category_name.branch_name

Denotes the last version on the named branch.

object_name`category_name

Denotes the last version on the last-created branch, or on the default branch which may be specified in the abstract object description.

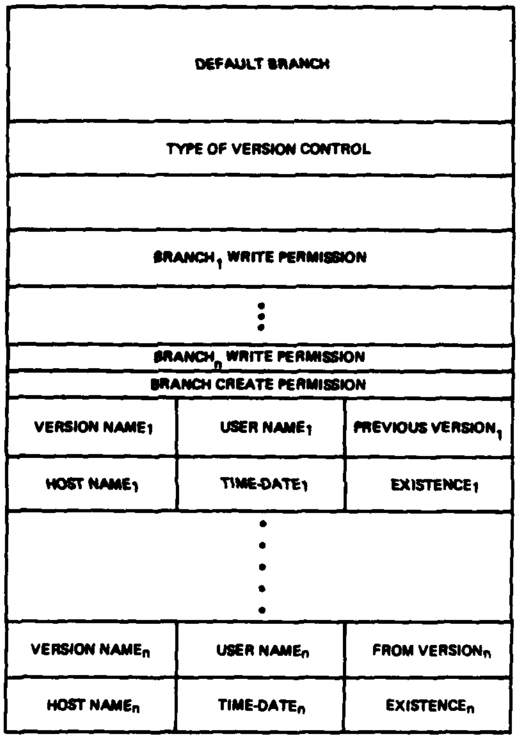
The information contained in the abstract object identifies the individual versions, defines the topology of the version tree, identifies those users and user groups permitted to create new branches or versions, and provides additional accessing descriptors. The logical structure of an abstract object is depicted in Figure 3-6 and described below.

Default Branch — The name of the branch to be used as the default when an unqualified reference is made. If this is empty, the last-created branch will be used as the default.

Version Control — Identifies the type of version control to be maintained for this object. The only two types of version control currently defined are delta and copy.

Branch Create Permission — Identifies the id`s for those users and user groups that may create a new branch. If not specified, create permission is the same as the write permission for the abstract object itself.

Branch Write Permission — Identifies the id`s for those users and user groups that may create a new version on a given branch. Branch write permission may be specified separately for each branch. If not specified for a given branch, write permission is the same as the write permission for the abstract object.



DEFAULT BRANCH

TYPE OF VERSION CONTROL

BRANCH$_1$ WRITE PERMISSION

•
•
•

BRANCH$_n$ WRITE PERMISSION

BRANCH CREATE PERMISSION

VERSION NAME$_1$	USER NAME$_1$	PREVIOUS VERSION$_1$
HOST NAME$_1$	TIME-DATE$_1$	EXISTENCE$_1$
	• • • • •	
VERSION NAME$_n$	USER NAME$_n$	FROM VERSION$_n$
HOST NAME$_n$	TIME-DATE$_n$	EXISTENCE$_n$

TP No. 021-1994-A

Figure 3-6. Logical Structure of Abstract Object

For each version, the following information is supplied:

Version Name — Name of a particular version of the abstract object.

User Name — The name (not the id) of the user who created the version.

Previous Version — The name of the version immediately preceding this version in the version tree.

Host Name — The host file name mapping for this version.

Date-Time — The date-time when this version was created.

Versions are ordinarily maintained in the KDB as separate objects, and this kind of version storage is *specified as the copy form of* version control. Considerable secondary storage may be required by copy version control, although the information content of versions defined by configuration objects may be deleted because reconstructability is guaranteed.

A second form of version control called delta may be requested for keyed objects. With delta version control, the information content of all versions is stored with the abstract object. Associated with each record is *an indication as to whether* the record is to be included or deleted for a given version. These version indications are relative to the version tree topology. Thus, unless an indication to the contrary is provided, a version automatically includes all records comprised in the previous version. For delta-controlled objects, version extraction and version creation is performed transparently to the user. The algorithm for version extraction is linear in the size of the abstract object.

3.2.5.5 Access Control

Access rights are associated with objects and partitions in the KDB. An individual access right is a pair: (user_or_group_id, access bits), which indicates that the types of access described by the access bits are to be associated to the identified user or group of users. The following types of access are defined with separate interpretations provided for ordinary objects and partition objects:

read r conveys the right to read from the object.
Partition: same interpretation.

write w conveys the right to write into the object.
Partition: conveys the right to delete an object in the partition.

execute e conveys the right to execute the object.
Partition: conveys the right to access objects in the partition access permissions for these objects must still be checked; without "e" permission, no access is allowed.

append a Conveys the right to append to the object.
Partition: conveys the right to create objects in the partition.

mod m Conveys the right to modify the access rights of the object.
Partition: same interpretation.

delete d Conveys the right to delete the object.
Partition: same interpretation.

For partition objects, a second set of access rights, called partition access rights, may be provided. These do not control access to the partition object itself, but to the objects that are members of the partition.

One more notion must be defined before the access permission algorithm can be given. Every process has associated with it four ids: real and effective user-ids, and real and effective group-ids. Ordinarily, when a process is created, it inherits all of these ids from the parent process. However, when a process is created from an executable object that has its set effective ids attribute set, the effective ids of the created process are set to the owner-id and group-id attributes of the object. These effective ids are the ones that are used for access control.

Access permission is computed as a function of the effective user and group ids of the process requesting access, the type of access requested, the partition access rights along the partition path to the object, and the

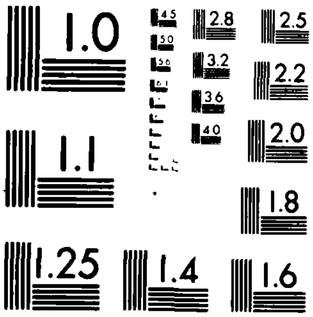
AD-A109 980　COMPUTER SCIENCES CORP FALLS CHURCH VA　F/G 9/2
ADA INTEGRATED ENVIRONMENT II COMPUTER PROGRAM DEVELOPMENT SPEC--ETC(U)
DEC 81　F30602-80-C-0292
UNCLASSIFIED　RADC-TR-81-364-PT-1　NL

3 of 4
AD
A109980

3 OF 4

AD

A109980



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

access rights on the object itself. Assume that the access bits in each access right are represented as a bit string, where the bit positions represent the access types: read, write, execute, append, delete, and modify. Consiuer a generalized access to an arbitrary object:

$$/partition_1/partition_2/.../partition_{n-1}/object_n$$

Assume that access be requested by a process with user id "u" and group id "g". For $partition_i$ in the above pathname, let $UPAB_i$ denote the partition access bits for user id "u", $GPAB_i$ the partition access bits for group id "g", and $DPAB_i$ the partition default access bits for all other users. For $object_n$, the ordinary access bits are used: UAB_i, GAB_i, and DAB_i. For all cases in which access bits are not specified, the default bit string of ones is used. The access permission bits, "APB", can now be computed as follows:

$$\begin{aligned} APB = {} & (UPAB_1 \;\&\; UPAB_2 \;\&\; . . . \;\&\; UPAB_{n-1} \;\&\; UAB_n) \\ & < (GPAB_1 \;\&\; GPAB_2 \;\&\; . . . \;\&\; GPAB_{n-1} \;\&\; GAB_n) \\ & < (DPAB_1 \;\&\; DPAB_2 \;\&\; . . . \;\&\; DPAB_{n-1} \;\&\; DAB_n) \end{aligned}$$

where "&" represents the bitwise "and" operation, and < represents the bitwise "or" operation. If, in the APB, the bit corresponding to the requesteo access type is set, access is granted, otherwise access is denied.

This access control scheme allows the partition access rights of a partition to control access to the entire subtree rooted by the partition. Access rights for individual objects need not be supplied. If absent, the rights are implicitly and recursively inherited from the partition access rights of the containing partitions.

3.2.5.6 KDB Input/Output

The KDBS Input/Output facilities defined in the Ada Run-time Support Package virtualize devices and methods of device access, reducing user knowledge of host idiosyncracies. An object exists in the KDB for each device supported by APSE. In this context the device name is the same as that of the object name. Moreover, the same access control facilities defined for objects of the KDB apply to the device objects.

The KDBS maintains no user-visible locks, nor are there any restrictions on the number of users who may have a specific object open. There are, however, sufficient internal locks to maintain the logical consistency of the KDB when two users try to update the same object, create objects of the same name in the the same partition, or delete objects that are currently being used.

The facilities of version control and access control are performed automatically and are tranparent to the user in processing I/O requests. The access control mechanism is initiated when a requesting MAPSE Process/Task issues an open or create function. The requesting user's access rights to the specified object are checked before any further processing of I/O requests. Once it has been determined that the requesting user does have the access rights corresponding to the type of I/O request made, the KDBS determines whether the effective user-id is to be modified for the execution of the initiated MAPSE Process/Task. This is done so that a user may only have access to a portion of KDB when executing the particular program.

The version control mechanism is initiated at several points in the processing of I/O requests. At create time, the user may have selected the version option of the create function and the KDBS creates an abstract object and initializes it. At open time, the KDBS must resolve ambiguous references to objects under version control by examining the abstract object for the default version and, if necessary under delta-type versioning, making a pass through the delta object to retrieve the desired version. The other I/O function that calls upon the versioning mechanism is the close. At close time, the KDBS updates the abstract object with the information about the new version and, if under delta versioning, noting those differences between the previous version and the current version for later references to the object.

The context of execution for the I/O, supported by the KDBS, occurs in two address spaces within the MAPSE, as shown in Figure 3-7; that which executes as a part of the MAPSE Process/Task and that which executes as part of the kernel process. The main reason for this separation is that I/O within the MAPSE is interrupt-driven and the requesting MAPSE Process/Tasks may

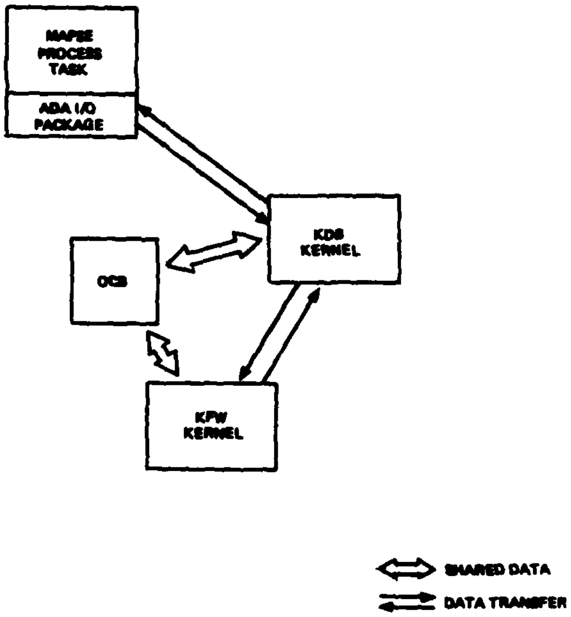


Figure 3-7. Execution Context

not be in core at the time the requested I/O has completed. In this way, some MAPSE Process/Tasks can be scheduled and executed while others wait for their I/O requests to complete.

The MAPSE Process/Task, upon declaring an open or create, has allocated an area in its address space that contains what is called the file descriptor. The file descriptor contains the relative index of what is termed the object control block (OCB) for future I/O operations issued by the MAPSE Process/Task. The object control block is allocated in kernel space and contains the name of the host file, the current read and current write positions, and relative index of the next host I/O block.

3.2.5.6.1 Ada I/O Open

The Ada I/O open converts the relative object name to an absolute object name and calls the KDBS Kernel to set up the necessary areas and interfaces required to interface the KFW Kernel to issue a host file open.(See Figure 3-8) The KDBS Kernel program first verifies that no object exists in the specified partition and checks the requesting users access rights to open the object in the specified partition. The KDBS Kernel then allocates an object control block for the object and requests the KFW Kernel to open the host file. When the KFW has completed the request control is returned to the KDBS Kernel. The KDBS Kernel then returns the relative index of the object control block to the requesting MAPSE Process/Task. The MAPSE Process/Task then places the relative index in the file descriptor for later I/O request references.

3.2.5.6.2 Ada I/O Create

The Ada I/O create converts the relative object name to an absolute object name and calls the KDBS Kernel to set up the necessary areas and interfaces required to interface the KFW Kernel to issue a host file create. (See Figure 3-9) The KDBS Kernel program first determines that no object exists in the specified partition and checks the requesting user`s access rights to create an object in the specified partition. The KDBS Kernel then allocates an object control block for the object and requests the KFW Kernel to create a host file. The KFW Kernel is responsible for generating a unique, host-dependent

MAPSE Process/Task	Kernel KAPSE Data Base
1. Issue Ada Open Request	
2. Convert the relative object reference to an absolute object pathname	
3. Allocate a file descriptor for the opened file	
4. Request Kernel KDBS Open services	
	1. Issue a suspend process for the requesting MAPSE Process/Task
	2. Search for referenced object starting at the root
	3. Determine whether referenced object is under version control and select version
	4. Check access right of the requesting user to the specified object
	5. Check the list of currently open objects for possible concurrency conflicts
	6. Allocate an object control block for the referenced object
	7. Request KFW to issue a host file open
	8. Receive control back from the KFW
	9. Request the rescheduling Process/Task and return control to lock index
5. Receive control block index and place in the appropriate file descriptor for later I/O referencing	

Figure 3-8 Ada I/O Open

MAPSE Process/Task	Kernel KAPSE Data Base
1. Issue Ada Create Request	
2. Convert the relative object reference to an absolute object pathname	
3. Allocate a file descriptor for the opened file	
4. Request Kernel KDBS Create services	
	1. Issue a suspend process for the requesting MAPSE Process/Task
	2. Check that the referenced object does not exist
	3. Check access right of the requesting user to the specified object
	4. Check the list of currently open objects for possible concurrency conflicts
	5. Allocate an object control block for the referenced object
	6. Request KFW to issue a host file create
	7. Receive control back from the KFW
	8. Request the rescheduling Process/Task and return control to lock index
5. Receive control block index and place in the appropriate file descriptor for later I/O referencing	

Figure 3-9 Ada I/O Create

name and issuing the appropriate host requests to create a file. When the KFW has completed the request control is returned to the KDBS Kernel to complete a partition entry with the host file name for later reference. The KDBS Kernel then returns the relative index to the object control block to the requesting MAPSE Process/Task. The MAPSE Process/Task then places the relative object control block index in the file descriptor for later I/O request references.

3.2.5.6.3 Ada Close

The Ada I/O close function disassociates an object from the MAPSE Process/Task by interfacing to the KDBS Kernel. See Figure 3-10. The KDBS Kernel demonstrates whether the object is opened for any othe MAPSE Process/Task currently executing in determining whether a host file close is to be issued. If another MAPSE user currently has the object open, the KDBS Kernel deallocates the OCB and returns control to the MAPSE Process/Task. When the object is not currently opened for anothe MAPSE Process/Task, the KDBS Kernel requests the KFW Kernel to issue a host file close and when the KFW returns control the KDBS deallocates the object control block and returns control to the requesting MAPSE Process/Task.

3.2.5.6.4 Ada I/O Read

The MAPSE Process/Task calls the KDBS Kernel with the relative object control block index and the number of characters to read. Control is passed to the KDBS Kernel and it determines whether a host file read must be issued in order to satifiy the request. See Figure 3-11. If the object control block buffer is empty then the KDBS Kernel requests the KFW Kernel to perform a host file read. The interface consists of passing the relative object control block number to the KFW. When the host file read has been completed, the KDBS Kernel gets the number of characters requested and returns them to the requesting MAPSE Process/Task.

MAPSE Process/Task

1. Issue Ada Close Request
2. Convert the relative object reference to an absolute object pathname
3. Request Kernel KDBS Close services

Kernel KAPSE Data Base

1. Check the open object list to make sure no other MAPSE Process/Tasks reference the object
2. Issue a suspend process for the requesting MAPSE Process/Task
3. Determine whether referenced object is under version control, type, and perform versioning
4. Request KFW to issuse a host file close
5. Receive control back from the KFW
6. Deallocate the object control block for the referenced object
7. Request the rescheduling Process/Task and return control to lock index

Figure 3-10. Ada Close

MAPSE Process/Task	Kernel KAPSE Data Base
1. Issue Ada Read Request	
	1. Determine whether object control block buffer is empty
	2. Issue a suspend process for th requesting MAPSE Process/Task
	3. Request the KFW to issue a host read inorder to fill the object control block buffer
	4. Determine whether object is under delta versioning and alter contents for specific version
	5. Process the read request
	6. Issue a reschedule of the suspended Process/Task

Figure 3-11. Ada I/O Read

3.2.5.6.5 Ada I/O Write

The MAPSE Process/Task calls the KDBS Kernel with the relative object control block index and the characters to be written. See Figure 3-12. Control is passed to the KDBS Kernel and it determines whether a host file write must be issued to satisfy the request. If the object control block buffer is full, the KDBS Kernel requests the KFW Kernel to perform a host file write. The interface consists of passing the relative object control block number to the KFW. When the host file write has been completed, the KDBS Kernel resets the buffer and returns to the requesting Process/Task.

3.2.5.6.6 Ada I/O Delete

The Ada I/O delete function enables the user to delete an object from the KDB. See Figure 3-13. The following conditions must be met before the object is really deleted from the KDB:

1. The requesting user must have the appropriate access to the specified object.
2. The object must not be in current use by another MAPSE user.
3. The object if under version control must be the current version of the branch. It can not be an iteration in the branch.
4. Configuration rules must be followed, in that objects in the dependency lists of other objects cannot be deleted.

3.2.5.7 Archiving Objects

An archive object is formed by combining an arbitrary number of separate objects into on single object. The constituent objects comprised the archive are called members of the archive object. The process of placing objects in an archive is particularly useful as a means of eliminating wasted space that occurs when individual objects do not occupy complete blocks of storage. Archiving is also convenient as a means of packaging sets of related objects and providing a means to save volatile copies of objects in the KAPSE data base.

MAPSE Process/Task	Kernel KAPSE Data Base
1. Issue Ada Write Request	
	1. Determine whether object control block buffer is full
	2. Issue a suspend process for the requesting MAPSE Process/Task
	3. Request the KFW to issue a host write inorder to fill the buffer
	4. Process the write request
	5. Issue a reschedule of the suspended MAPSE Process/Task

Figure 3-12. Ada I/O Write

210

MAPSE Process/Task

1. Issue Ada Delete Request
2. Convert the relative object reference to an absolute object path name
3. Request Kernel KDBS Delete services

Kernel KAPSE Data Base

1. Issue a suspend process for the requesting MAPSE Process/Task
2. Search for referenced object starting at the root
3. Determine whether referenced object is under version control and select version
4. Check access right of the requesting user to the specified object
5. Check the list of currently opened object for possible concurrency conflicts
6. Request the KFW to issue a host file delete
7. Request that the suspended MAPSE Process/Task be rescheduled

Figure 3-13. Ada I/O Delete

The archive facility provides a set of operations that the user of MAPSE can employ to create new archive objects and to maintain existing ones. The operations are:

1. List - Lists the members of a particular archive object.

2. Append - Creates an archive object and adding a new member to the particular archive.

3. Replace - Creates an archive, replacing an exisiting member or adding the a new member.

4. Update - Replaces only recent members with a more recent version of an object.

5. Delete - Deletes a member from an archive object.

6. Extract - Retrieves a copy of the member for use with in the KDBS.

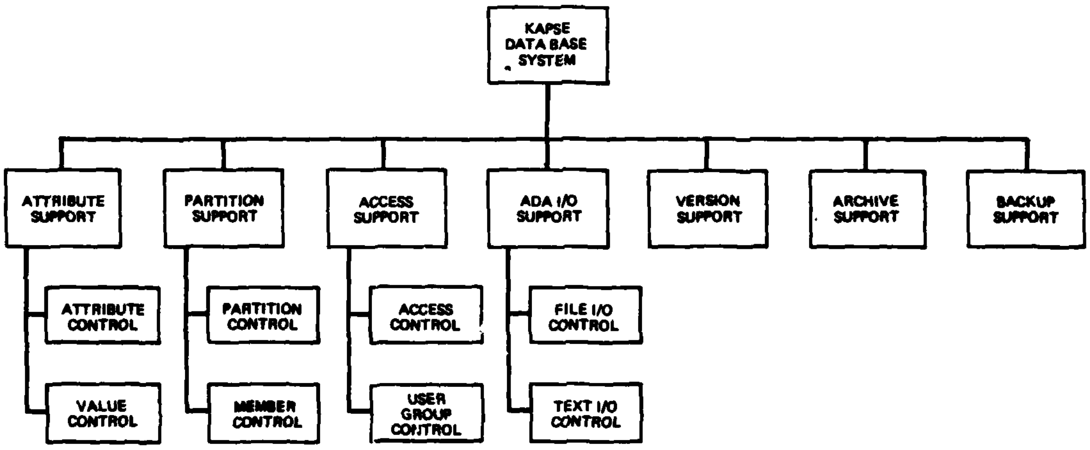
For a more detailed discussion of the archiving capabilities of the MAPSE system see Paragraph 3.3.6 of this document.

3.2.5.8 Backup and Restore

The Backup/Restore feature of the MAPSE system is defined in order to augment those facilities that may or may not exist on the implemented host. This facility is flexible enough to allow the user to backup any specific branch of the KDBS hierarchy. The Backup is a complete dump, starting from the specified branch and continuing down the hierarchy until all objects have been copied to an external medium. The Backup capability establishes a checkpoint in time, essentially a snapshot of the branch being dumped, formatted in such a way as to permit restoration of the branch in case of lost or damaged objects. The Restore capability provides the user a means to restore the lost or damaged objects from the Backup medium. A single object or the entire branch of the hierarchy may be retrieved from the Backup medium. The frequency with which system Backup is performed is installation-dependent for this facility may be used a frequently as every hour to once a day, depending on the volatility of the data base.

3.3 DETAILED FUNCTIONAL DESCRIPTION

The following sections describe those functions that are used to control and maintain objects in the KAPSE data base. These functions are made visible through the virtual interface and are available to the general MAPSE user. Figure 3-14 shows a logical breakdown of those functions.



TP No. 021-1889-A

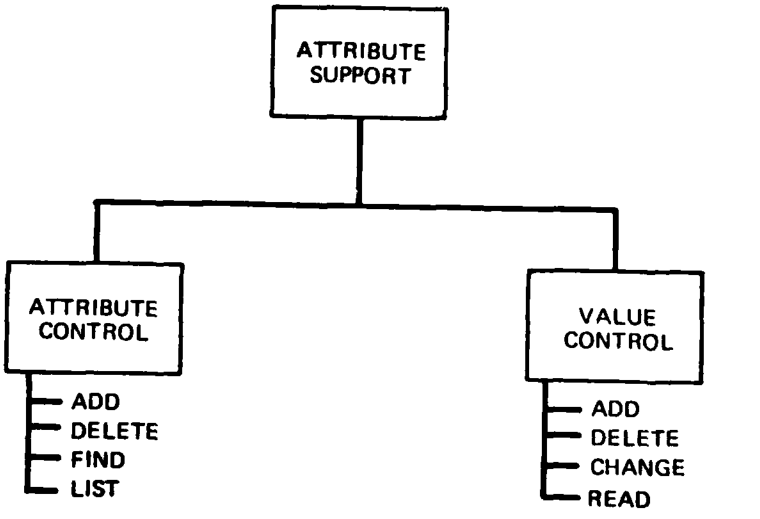
Figure 3-14. KDBS Functional Diagram

3.3.1 Attribute Support

This section describes facilities that enable the users of MAPSE to associate and maintain object attributes. These facilities are included in the KDBS Utility Package and their specifications are made visible through the virtual interface. The following functions have been logically grouped into those that are used to control the attribute and those which control the value associated with that attribute. See Figure 3-15 for a logical breakdown of Attribute Support Functions.

3.3.1.1 Attribute Facilities

This section describes those functions available to the MAPSE user in associating and maintaining attributes of the objects in the KDB. The restrictions for associating and maintaining attributes are that the requesting process must have "mod" access to the object and that the attribute name must be unique for the specified object.



TP No. 021-1987-A

Figure 3-15. Attribute Support Functions

3.3.1.1.1 Add Attribute - Adda

This function defines and adds a new attribute to a specified object in the KAPSE data base. An initial null value of the attribute is supplied if the user fails to specify one. Adda is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Adda calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.1.1.1.1 Inputs

There are three input arguments defined for Adda:

Oname - The name of the object to which the attribute is to be added
Aname - The name of the attribute
Avalue - The initial value of the attribute.

3.3.1.1.1.2 Processing

Adda locates the object specified by Oname. The attribute named by Aname is associated with the specified object and initialized to the value provided in Avalue.

Errors detected:

1. Requesting process does not have "mod" access to the specified object.
2. Object specified does not exist.
3. Attribute specified already defined for the object.

3.3.1.1.1.3 Outputs

There is one output argument defined for Adda; it indicates success of execution or an error condition.

3.3.1.1.2 Delete Attribute - Dela

This function deletes an attribute from a specified object in the KAPSE data base. Dela is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Dela calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.1.1.2.1 Inputs

There are two input arguments defined for Dela:

Oname - The name of the object to which the attribute is to be deleted.

Aname - The name of the attribute to be deleted.

3.3.1.1.2.2 Processing

Dela locates the object specified in Oname. The attribute specified by Aname is then deleted from the specified object.

Errors Detected:

1) Requesting process does not have "mod" access to the specified object.

2) Attribute specified is undefined for the specified object.

3) Object specified does not exist.

3.3.1.1.2.3 Outputs

There is one output argument defined for Dela; it indicates success of execution or an error condition.

3.3.1.1.3 Find Attribute - Finda

This function finds a set of objects that contain a set of specified attribute values. The default search is limited to the specified partition, but an option permits a search of all subpartitions. Finda is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Finda calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.1.1.3.1 Inputs

There are two input arguments defined for Finda:

Avstring - The string of attribute value pairs separated by boolean operators (i.e., attr$_1$=value$_1$ & attr$_2$=value$_2$ ..).

Search - The option to search subpartitions (yes/no).

3.3.1.1.3.2 Processing

Finda locates a set of process visible objects which satisfy the attribute value pairs specified. If the Search option is set, then all subpartitions are also searched for objects containing the specified attribute values.

Errors Detected:

1. Value of the Avstring is of an invalid format.
2. Invalid option selection for the Search argument.

3.3.1.1.3.3 Outputs

Finda indicates the success of execution or the an error condition. If Finda was successful, a list of process-visible objects is returned as well.

3.3.1.1.4 List Attributes - Lista

This function lists all attributes and their values for a specified object in the KAPSE data base. Lista is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Lista calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.1.1.4.1 Inputs

There is one input argument defined for Lista:

Oname - The name of the object in which all attributes and their values are to be listed.

3.3.1.1.4.2 Processing

Lista locates the object specified by Oname. A list of attributes and their corresponding values is retrieved for the specified object.

Error Detected:

1. Requesting process does not have "read" access to the specified object.
2. Object specified does not exist.

3.3.1.1.4.3 Outputs

Lista returns an indication of success or an error condition. If Lista was successful a list of attributes and their values is retrieved and returned to the requesting process.

3.3.1.2 Value Facilities

This section describes those facilities available to the MAPSE user which provides the ability to manipulate values of the attribute.

3.3.1.2.1 Add a Value to an Attribute - Addv

This function adds another value to an attribute value list for a specified object in the KAPSE data base. Addv is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Addv calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.1.2.1.1 Inputs

There are three input arguments defined for Addv:

Oname - The name of the object to which the value is to be added.
Aname - The defined attribute name to which the value is to be added.
Avalue - The value to be associated with the defined attribute.

3.3.1.2.1.2 Processing

Addv locates the object specified by Oname. The value specified Avalue is added to the list of values for the attribute specified by Aname.

Errors Detected:

1. Requesting process does not have "mod" access to the specified object.
2. Object specified does not exist.
3. Attribute specified is not defined for the specified object.

3.3.1.2.1.3 Outputs

There is one output argument defined for Addv; it indicates success of execution or an error condition.

3.3.1.2.2 Change Attribute Value - Chgv

This function changes the value of the attribute for a specified object in the KAPSE data base. Chgv is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Chgv calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.1.2.2.1 Inputs

There are three input arguments defined for Chgv:

Oname - The name of the object to which the value of an attribute is to be changed.

Aname - The name of the attribute to which its value is to be changed.

Avalue - The new value of the attribute.

3.3.1.2.2.2 Processing

Chgv locates the object specified by Oname. The value specified by Avalue replaces the current value of the attribute specified by Aname.

Errors Detected:

1. Requesting process does not have "mod" access to the specified object.
2. Attribute specified is undefined for the specified object.
3. Object specified does not exist.
4. Attribute contains a list of values and this function is not available for changing attribute values.

3.3.1.2.2.3 Outputs

There is one output argument defined for Chgv; it indicates success of execution or an error condition.

3.3.1.2.3 Delete Attribute Value - Delv

This function deletes a value from a list of values for an attribute of a specified object in the KAPSE data base. Delv is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Delv calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.1.1.2.3 Inputs

There are three input arguments defined for Delv:

Oname - The name of the object to which an attribute value is to be deleted.

Aname - The name of the attribute to be deleted.

Avalue - The attribute value to be deleted from the list of attribute values.

3.3.1.2.3.2 Processing

Delv locates the object specified by Oname. The value specified by Avalue is deleted from the list of values for the attribute specified by Aname.

Errors Detected:

1. Requesting process does not have "mod" access to the specified object.
2. Attribute specified is undefined for the specified object.
3. Object specified does not exist.
4. Attribute specified does not contain a list of values and this function is not valid for this request.

3.3.1.2.3.3 Outputs

There is one output argument defined for Delv; it indicates the success of execution or an error condition.

3.3.1.2.4 Read Value - Readv

This function reads the value of an attribute for a specified object in the KAPSE data base. Readv is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Readv calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.1.2.4.1 Inputs

There are two arguments passed in the call to Readv:

Oname - The name of the object in which the attribute value is to be read.

Aname - The name of the attribute in which the value is to be read.

3.3.1.2.4.2 Processing

Readv locates the object specified by Oname. The value of the attribute specified by Aname is then read and returned to the requesting process.

Errors Detected:

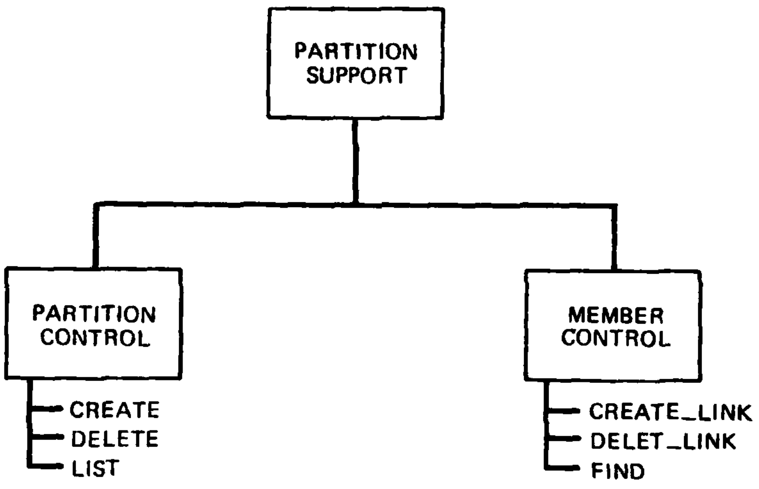
1. Requesting process does not have "read" access to the specified object.
2. Object specified does not exist.
3. Attribute specified does not exist for the specified object.

3.3.1.2.4.3 Outputs

Readv returns the success of execution or an error condition. If Readv was successful, the value of the specified attribute is returned to the requesting process.

3.3.2 Partition Support

This paragraph describes facilities which provides the users of MAPSE with the ability to create and maintain partitions within the KAPSE data base. These facilities are included in the KDBS Utility Package and their specifications are made visible through the virtual interface. The following functions have been logically grouped into those that are used to control partition objects and those that control the members of a partition. See Figure 3-16 for a logical break down of Partition Support functions.



TP No. 021-1988-A

Figure 3-16. Partition Support Functions

3.3.2.1 Partition Facilities

This paragraphs describes those functions available to the MAPSE user in creating and maintaining partitions in the KAPSE data base.

3.3.2.1.1 Create Partition - Createp

This function creates and adds a partition object in a specified partition of the KAPSE data base. The partition is added to the current working partition if no path is specified. Createp is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Createp calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.2.1.1.1 Inputs

There is one input argument defined for Createp:

Pname - The name of the partition to be created. If the partition is to be created in another partition then Pname must have a path specified as part of the argument value.

3.3.2.1.1.2 Processing

Createp locates the point in the KAPSE data base hierarchy at which the partition specified by Pname is to be created. If a path is not specified, the partition is created in the current working partition.

Errors Detected:

1. Requesting process does not have "write" access for the partition in which a new partition is to be created.
2. Partition in which a new partition is to be created does not exist.
3. Partition already exists with the same attributes as the new one to be created.

3.3.2.1.1.3 Outputs

There is one output argument defined for Createp; it indicates success of execution or an error condition.

3.3.2.1.2 Delete Partition - Deletep

This function deletes a partition object from the KAPSE data base. All members of the partition must be deleted before a partition object can be deleted. Deletep is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Deletep calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.2.1.2.1 Inputs

There is one input argument defined for Deletep:

Pname - The name of the partition object to be deleted.

3.3.2.1.2.2 Processing

Deletep locates the partition specified by Pname. The partition is then deleted from the KAPSE data base only if all members of the specified partition have been deleted.

Errors Detected:

1. Specified partition does not exist.
2. Requesting process does not have "delete" access to the specified partition object.
3. Partition object still has member defined.

3.3.2.1.2.3 Outputs

There is one output argument defined for Deletep; it indicates the success of execution or an error condition.

3.3.2.1.3 List Partition - Listp

This function lists the members of a partition object in the KAPSE data base. Listp is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Listp calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.2.1.3.1 Inputs

There is one input argument defined for Listp:

Pname - The name of the partition in which its membership list is to be retrieved.

3.3.2.1.3.2 Processing

Listp locates the partition specified by Pname. If Listp is successful, a list of objects that are members of the specified partition is returned to the requesting process.

Errors Detected:

1. Requesting process does not have "read" access to the specified partition.
2. Partition specified does not exist.

3.3.2.1.3.3 Outputs

Listp returns an indication of success or an error condition. If Listp was successful, a list of member objects is returned to the requesting process.

3.3.2.2 Member Control

This section describes those facilities available to the MAPSE user to allow manipulation of the members of partition objects.

3.3.2.2.1 Create Link - Linkc

This function creates a link entry in the specified partition object of the KAPSE data base. Linkc is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Linkc calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.2.2.1.1 Inputs

There are two input arguments defined for Linkc:

Pname - The partition name in which a link entry is to be created.
Oname - The name of the object in which the link is to be made.

3.3.2.2.1.2 Processing

Linkc locates the partition specified by Pname. A link entry specified by Oname is created in the specified partition. The name of the object is assumed to be the name of the link entry in the specified partition.

Errors Detected:

1. Requesting process does not have "write" access to the specified partition object.
2. Object specified does not exist.
3. Link already extablished with the same name.
4. Requesting user does not have read access to the object in which the link is to be established.

3.3.2.2.1.3 Outputs

There is one output argument defined for Linkc; it indicates success of execution or an error condition.

3.3.2.2.2 Delete Link - Linkd

This function deletes a link entry in the specified partition object of the KAPSE data base. Linkd is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Linkd calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.2.2.2.1 Inputs

There are two input arguments defined for Linkd:

Pname - The partition name in which a link entry is to be created.
Oname - the name of the object in which the link is to be made.

3.3.2.2.2.2 Processing

Linkd locates the partition specified by Pname. A link entry specified by Oname is deleted from the specified partition.

Errors Detected:

1. Requesting process does not have write access to the specified partition object.

2. Link specified does not exist.

3.3.2.2.2.3 Outputs

There is one output argument defined for Linkd; it indicates success of execution or an error condition.

3.3.2.2.3 Find Partition Entry - Findpe

This function finds a particular entry in the specified partition object in the KAPSE data base. Findpe is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Findpe calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.2.2.3.1 Inputs

There is one argument passed in the call to Findpe:

Oname - the object name in which to locate in the specified partition.

3.3.2.2.3.2 Processing

Findpe locates the partition in which to search for the object specified by Oname.

Errors Detected:

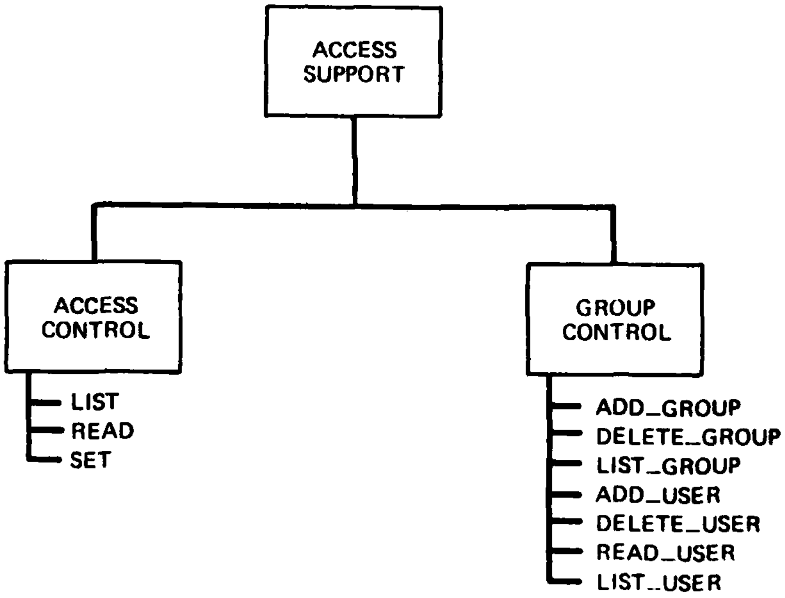
1. Requesting process does not have read access to the specified partition object.
2. Object specified does not exist in the specified partition.
3. Oname arguments specifies a nonexistent partition.

3.3.2.2.3.3 Outputs

There is one output argument defined for Findpe; it indicates the success of execution or an error condition.

3.3.3 Access Support

This section describes facilities that enable MAPSE users to create and maintain access controls on objects in the KAPSE data base and to create and maintain groups. These facilities are included in the KDBS Utility Package and their specifications are made visible through the virtual interface. The following functions have been logically grouped into those that create and maintain the access attributes and those that maintain and control groups. See Figure 3-17 for a logical break down of Access Support functions.



TP No. 021-1989-A

Figure 3-17. Access Support Functions

3.3.3.1 Access Control

This paragraph describes those functions available to the MAPSE user in creating and maintaining access controls on objects.

3.3.3.1.1 List Access Attribute - Laccess

This function lists users and groups which have access rights to an object in the KAPSE data base. Laccess is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Laccess calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.3.1.1.1 Inputs

There is one input argument defined for Laccess:

Oname - The name of the object in which users and groups that have access to it are to be retrieved.

3.3.3.1.1.2 Processing

Laccess locates the object specified by Oname. A list of users and groups that have access to the specified object is retrieved for the requesting process.

Errors Detected:

1. Requesting process does not have "read" access to the specified object.
2. Object specified does not exist.

3.3.3.1.1.3 Outputs

The value returned by Laccess indicates success of execution or an error condition. If Laccess was successful, a list of users and groups with access to the specified object is returned to the requesting process.

3.3.3.1.2 Read Access - Raccess

This function reads the access rights of a specified user or group for a specified object in the KAPSE data base. Raccess is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Raccess calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.3.1.2.1 Inputs

There are two input arguments defined for Raccess:

Oname - The name of the object in which the access rights are to be read.

Ugname - The user or group name in which the access rights are to be retrieved.

3.3.3.1.2.2 Processing

Raccess locates the object specified by Oname. The access rights are then retrieved for the user or group name specified by Ugname.

Errors Detected:

1. Requesting process does not have "read" access to the specified object.
2. Object specified does not exist.
3. User or group name is invalid.

3.3.3.1.2.3 Outputs

The value returned by Raccess indicates success of execution or an error condition. If Raccess was successful, a value is returned to show the access rights for the specified user or group to the requesting process.

3.3.3.1.3 Set Access - Saccess

This function creates, modifies, and deletes access rights to a specified object in the KAPSE data base. Saccess is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Saccess calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.3.1.3.1 Inputs

There are three input arguments defined for Saccess:

Oname - The name of the object in which the access rights are to be set.

Ugname - The name of the user or group in which the specified access rights are to be assigned.

Accval - The access rights to be assigned for the specified user or group.

3.3.3.1.3.2 Processing

Saccess locates the object specified by Oname. The access rights specified by Accval are set for the user or group specified by Ugname.

Errors Detected:

1. Requesting process does not have "mod" access to the specified object.
2. Object specified does not exist.
3. Access rights to be associated are invalid.
4. User or group name specified is invalid.

3.3.3.1.3.3 Outputs

There is one output argument defined for Saccess; it indicates success of execution or error condition.

3.3.3.2 Group Control

This section describes those facilities that enable the MAPSE user to to create and maintain groups.

3.3.3.2.1 Add Group Member - Addgm

This function adds a user or set of users to a specified group in the MAPSE system. Addgm is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Addgm calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.3.2.1.1 Inputs

There are two arguments passed in the call to Addgm.

Gname - The name of the group.

Unames - The set of users to be added to the group definition and is in the form of: $user\_name_1$;$user\_name_2$; . . . $user\_name_n$.

3.3.3.2.1.2 Processing

Addgm locates the group specified by Gname and associates those members listed by Unames to it.

Errors Detected:

1. Group specified does not exist.
2. User name specified is an authorized user.
3. Requesting process does not have the same user id as the creating user id.

3.3.3.2.1.3 Outputs

There is one output argument defined for Addgm; it indicates success of execution or an error condition.

3.3.3.2.2 Create Group - Createg

This function creates a group definition for the access control mechanism of the KAPSE data base. Createg is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Createg calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.3.2.2.1 Inputs

There is one input argument defined for Createg.

Gname - The name of the group to be created.

3.3.3.2.2.2 Processing

Createg creates a group entry with the name specified by Gname and adds it to the MAPSE group list.

Errors Detected:

1. Group already exists with the same name.

3.3.3.2.2.3 Outputs

There is one output argument defined for Createg; it indicates success of execution or an error condition.

3.3.3.2.3 Delete Group - Deleteg

This function deletes a group definition from the access control mechanism of the KAPSE data base. Deleteg is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Deleteg calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.3.2.3.1 Inputs

There is one input argument defined for Deleteg:

Gname - The name of the group to be deleted.

3.3.3.2.3.2 Processing

Deleteg locates the group specified by Gname and deletes it from the MAPSE group list.

Errors Detected:

1. Requesting process does not have the same user-id as the creating process.
2. Group specified does not exist.

3.3.3.2.3.3 Outputs

There is one output argument defined for Deleteg; it indicates success of execution or an error condition.

3.3.3.2.4 Delete Group Member - Deletegm

This function deletes a user from a specified group in the MAPSE system. Deletegm is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Deletegm calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.3.2.4.1 Inputs

There are two input arguments defined for Deletegm:

Gname - The name of the group in which a member is to be deleted.
Uname - The name of the user to be deleted from the specified group.

3.3.3.2.4.2 Processing

Deletegm locates the group specified by Gname and deletes the user specified by Uname from it.

Errors Detected:

1. Requesting process does not have the same user-id as the creating process.
2. Group specified does not exist.
3. User specified does not defined in the specified group.

3.3.3.2.4.3 Outputs

There is one output argument defined for Deletegm; it indicates success of execution or an error condition.

3.3.3.2.5 Find Group Member - Findgm

This function finds a user in a specified group. Findgm is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Findgm calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.3.2.5.1 Inputs

There are two input arguments defined for Findgm:

Gname - The of the group in which to find a specified user.

Uname - The name of the user in which to check if a member of the specified group.

3.3.3.2.5.2 Processing

Findgm locates the group specified by Gname and locates the user specified by Uname in the specified group.

Errors Detected:

1. Requesting process does not have the same user-id as the creator.
2. Group specified is undefined.
3. User specified is not an authorized user of MAPSE.
4. User specified is not a member of the specified group.

3.3.3.2.5.3 Outputs

There is one output argument defined for Findgm; it indicates success of execution or an error condition.

3.3.3.2.6 List Group - Listg

This function retrieves a list of currently defined groups for the MAPSE system. Listg is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Listg calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.3.2.6.1 Inputs

There are no input arguments defined for Listg.

3.3.3.2.6.2 Processing

Listg retrieves a list of currently defined groups for the MAPSE system.

Errors Detected:

1. There are no currently defined groups for the MAPSE system.

3.3.3.2.6.3 Outputs

The value returned by Listg indicates success of execution or an error condition. If Listg was successful, a list of currently defined groups is returned to the requesting process.

3.3.3.2.7 List Group Members - Listgm

This function returns a list of users currently defined for a specified group. Listgm is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Listgm calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.3.2.7.1 Inputs

There is one input argument defined for Listgm:

Gname - The name of the group in which a list of its members is to be retrieved.

3.3.3.2.7.2 Processing

Listgm locates the group specified by Gname and retrieves a list of the members in the specified group.

Errors Detected:

1. Group specified does not exist.
2. Requesting process does not have the same user id as the group creator.

3.3.3.2.7.3 Outputs

The value returned by Listgm indicates success of execution or an error condition. If Listgm was successful then a membership list of the group is returned to the requesting process.

3.3.4 Ada Input/Output Support

I/O facilities are predefined in the Ada language by means of two packages. The generic package INPUT_OUTPUT defines a set of I/O primitives applicable to files containing elements of a single type. Additional primitives for text input-output are supplied in the package TEXT_IO. These facilities are described in the following sections. See Figure 3-18 for a logical break down of Ada Input/Output Support functions.

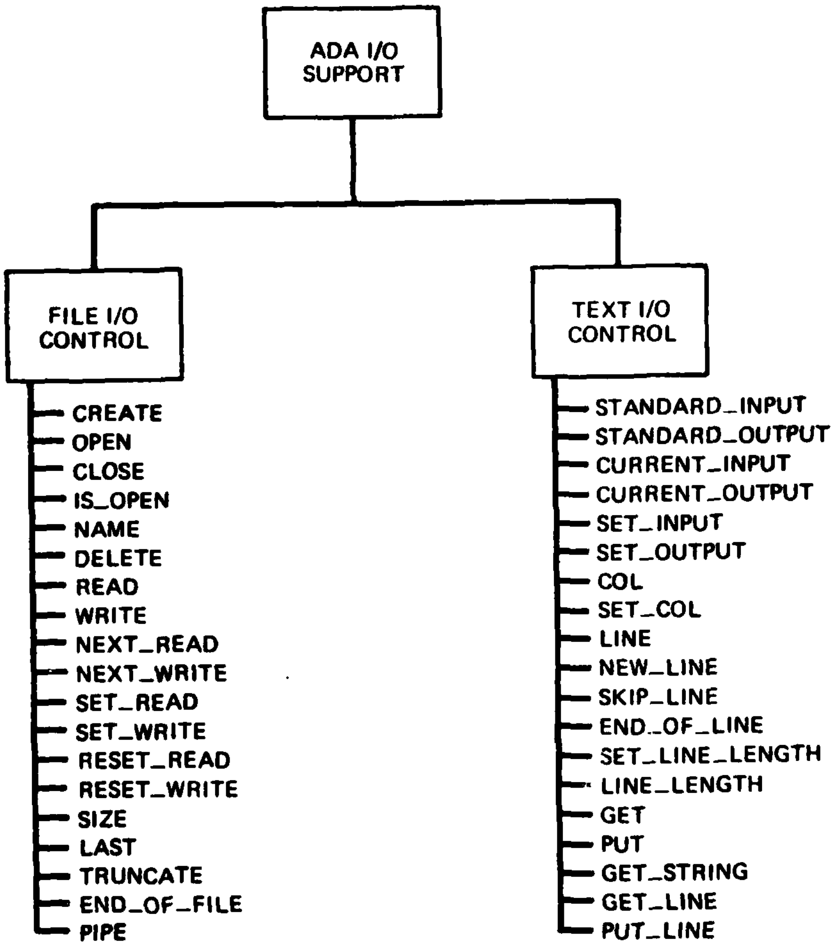


Figure 3-18. Ada Input/Output Support Functions

x 36

3.3.4.1 Ada File Input/Output

Files are declared and subsequently associated with the appropriate sources and destinations, called external files, such as peripheral devices or data sets. Distinct file types are defined to provide either read-only access, write-only access or read-write access to external files. The corresponding file types are called IN_FILE, and OUT_FILE.

An open IN_FILE or INOUT_FILE can be read; an open OUT_FILE or INOUT_FILE can be written. A file that can be read has a current read position, which is the position number of the element available to the next read operation. A file that can be written to has a current write position, which is the position number of th element available to be modified by the next write operation. The current read or write positions can be changed. Positions in a file are expessed in the implementation-defined integer type FILE_INDEX.

A file has a current size, which is the number of defined elements in the file, and an end position, which is set to the position number of the first defined element if any, and is otherwise zero.

When a file is opened or created, the current write position is set to one, and the current read position is set to the position number of the first defined element, or one if no element is defined.

The operations available for file processing are described in the following paragraphs and apply only to open files. The exception STATUS_ERROR is raised if one of these operations is applied to a file that is not open. The exception USE_ERROR is raised if an operation is incompatible with the properties of the external file. The exception DEVICE_ERROR is raised if an I/O operation cannot be completed because of a malfunction of the underlying MAPSE system.

3.3.4.1.1 Create a File - Create

This function creates and associates an object to a MAPSE process or task. Create is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Create calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.1.1 Inputs

There are two input arguments defined for Create:

File - The internal file name for the external object to be created.
Name - The name of the object to be created.

3.3.4.1.1.2 Processing

Create expands the object name reference to an absolute object reference and creates an entry in the specified partition for the object. Create then allocates an object control block for the object and requests the KFW Kernel to issue a host file create passing the relative index to the control block. When control returns to the Create function, the host file name generated is placed in the specified partition and control is returned to the requesting MAPSE Process/Task.

Errors Detected:

1. File is already open.
2. Object specified already exists.
3. Requesting process does not have "write" access to the specified partition.

3.3.4.1.1.3 Outputs

There is one output argument defined for Create; it indicates success of execution or an error condition. If Create was successful, a relative index to the object control block is returned to the requesting process.

3.3.4.1.2 Open a File - Open

This function opens an object in the KAPSE data base for a MAPSE process or task. Open is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Open calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.2.1 Inputs

There are two input arguments defined for Open:

File	- The internal file name for the object to be opened.
Name	- The name of the object to be opened.

3.3.4.1.2.2 Processing

Open locates the object specified by Name and allocates an object control block for the object. Open then requests the KFW Kernel to issue a host file open. When control is returned to Open an entry is made in the KAPSE data base Open Object Table and returns control to the requesting MAPSE process or task.

Errors Detected:

1. Object specified is already open for the requesting MAPSE process or task.
2. Requesting process does not have "read" and/or "write" access to the specified object.
3. Object specified does not exist.

3.3.4.1.2.3 Outputs

There is one output argument defined for Open; it indicates the success of execution of an error condition. If Open was successful then a relative index to the object control block is returned to the requesting process.

3.3.4.1.3 Close a File - Close

This function closes an object in the KAPSE data base for a MAPSE process or task. Close is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Close calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.3.1 Inputs

There is one input argument defined for Close:

File - The internal name of the file to be closed.

3.3.4.1.3.2 Processing

Close locates the object control block for the file specifed by File and checks the KAPSE data base Open Object Table for any other MAPSE process or tasks that may have the specified object open. If the specified object is under version control, the type of version control is determined and the abstract object is updated. If no other process or tasks are currently using the object, Close requests the KFW Kernel to issue a host file close. The control block is deallocated and control is returned to the requesting process.

3.3.4.1.3.3 Outputs

There is one output argument defined for Close; it indicates success of execution or an error condition.

3.3.4.1.4 Check if File is Open - Is_Open

This function determines whether a specified object is currently open for the requesting process. Is_Open is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Is_Open calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.4.1 Inputs

There is one input argument defined for Is_Open:

File - The internal name of the file in which its open status is to be checked.

3.3.4.1.4.2 Processing

Is_Open accesses the object control block corresponding to the file specified by File in order to determine whether the object is open.

Errors Detected:

1. None

3.3.4.1.4.3 Outputs

Is_Open returns a boolean value as to the status of the specified file.

3.3.4.1.5 Get External File Name - Name

This function returns the absolute object name associated with the internal file name. Name is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Name calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.5.1 Inputs

There is one input argument defined for Name:

File - The internal name of the file in which its external association is to be retrieved.

3.3.4.1.5.2 Processing

Name locates the object control block for the object specified by File and retrieves the absolute object name associated with the internal file name for the requesting process.

Errors Detected:

1. No external object associated with the internal file name.

3.3.4.1.5.3 Outputs

Name returns a string representing the fully qualified name of the external object currently associated with the given internal file.

3.3.4.1.6 Delete a File - Delete

This function deletes a specified object from the KAPSE data base. Delete is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Delete calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.6.1 Inputs

There is one input argument defined for Delete:

Name - The name of the external file to be deleted.

3.3.4.1.6.2 Processing

Delete locates the object specified by Name and requests the KFW Kernel to issue a host file delete. When the host file delete has taken place the Delete function removes the associated partition entry and returns control to the requesting process.

Errors Detected:

1. Requesting process does not have "delete" access to the specified object.

2. Object specified is currently in use by another MAPSE process or task.

3. Object specified is a member of a currently active configuration.

4. Object specified does not exist.

3.3.4.1.6.3 Outputs

There is one output argument defined for Delete; it indicates success of execution or an error condition.

3.3.4.1.7 Read a File - Read

This function reads the next item in a specified object of the KAPSE data base. Read is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Read calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.7.1 Inputs

There is one input argument defined for Read:

File - The internal name of the file to be read.

3.3.4.1.7.2 Processing

Read locates the object control block coresponding to the internal file name specified by File. Read examines the OCB Buffer to see if it is empty to determine whether a host file read must be issued to the KFW Kernel. Read then returns the next item to the requesting process.

Errors Detected:

1. Object specified is not open.

3.3.4.1.7.3 Outputs

There is one output argument defined for Read; it indicates success of execution or an error condition.

3.3.4.1.8 Write to a File - Write

This function writes an item to a specified object in the KAPSE data base. Write is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Write calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.8.1 Inputs

There are two input arguments defined for Write:

File - The internal name of the file to be written to.
Item - The next element to be written to the external file.

3.3.4.1.8.2 Processing

Write ensures that this operation is compatable to the properties of the external object and that the internal file name specified by File is open. The Item is placed in the object control block Buffer and the current write position is incremented by one. The current file size is incremented by one if the element in the current write position was not defined, and sets the end position to the written position if the written position exceeds the end position. If Write determines that the control block buffer is full, an interface is established to the KFW Kernel to issue a host file write and clears the control block buffer. Control is then returned to the requesting process.

Errors Detected:

1. Object specified is not open.
2. Write operation cannot be completed because of a device error.
3. Write operation is incompatable with the properties of the object.

3.3.4.1.8.3 Outputs

There is one output argument defined for Write; it indicates the success of execution or an error condition.

3.3.4.1.9 Get Current Read Position - Next_Read

This function returns the current read position for a specified object in the KAPSE data base. Next_Read is incluoed in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Next_Read calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.9.1 Inputs

There is one input argument defined for Next_Read:

File - The internal name of the file in which the current read position is to be returned.

3.3.4.1.9.2 Processing

Next_Read locates the object control block corresponding to the internal file name specified by File is open. The object control block is then accessed for the current read position and returns.

Errors Detected:

1. Object specified is not open.

3.3.4.1.9.3 Outputs

Next_Read returns the current read position of the specified object or raises a STATUS_ERROR because the specified object was not open.

3.3.4.1.10 Get Current Write Position - Next_Write

This function returns the current write position for the specified object in the KAPSE data base. Next_Write is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Next_Write calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.10.1 Inputs

There is one input argument defined for Next_Write:

File - The internal name of the file in which the current write position is to be returned.

3.3.4.1.10.2 Processing

Next_Write locates the object control block corresponding to the object specified by File. The object block is accessed for the current write position and returns.

Errors Detected:

1. Object specified is not open.

3.3.4.1.10.3 Outputs

Next_Write returns the current write position of the specified object or raises a STATUS_ERROR because the specified object was not open.

3.3.4.1.11 Set Current Read Position - Set_Read

This function sets the current read position for a specified object in the KAPSE data base. Set_Read is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Set Read calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.11.1 Inputs

There are two input arguments defined for Set_Read:

File - The internal name of the file in which the current read position is to be set.

To - The value to be assigned to the current read position of the file.

3.3.4.1.11.2 Processing

Set_Read locates the object control block corresponding to the object specified by File. Set_Read then accesses the control block, and sets the current read position, and returns.

Errors Detected:

1. Object specified is not currently open.

3.3.4.1.11.3 Outputs

Set_Read raises a STATUS_ERROR when the specified file is not open.

3.3.4.1.12 Set Current Write Position - Set_Write

This function sets the current write position of a specified object in the KAPSE data base. Set_Write is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Set Write calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.12.1 Inputs

There are two input arguments defined for Set_Write:

File - The internal name of the file in which the current write position is to be set.

To - The value to be assigned to the current write position of the file.

3.3.4.1.12.2 Processing

Set_Write locates the object control block corresponding to the object specified by File. Set_Write then accesses the control block and sets the current write position and returns.

Errors Detected:

1. Object specified is not open.

3.3.4.1.12.3 Outputs

Set_Write raises a STATUS_ERROR when the specified file was not open.

3.3.4.1.13 Reset Current Read Position - Reset_Read

This function resets the current read position of a specified object in the KAPSE data base. Reset_Read is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Reset_Read calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.13.1 Inputs

There is one input argument defined for Reset_Read:

File - The internal name of the file in which the current read position is to be set.

3.3.4.1.13.2 Processing

Reset_Read locates the object control block corresponding to the object specified by File, accesses the control block, and resets the current read position to one or the first element of the object, and returns.

Errors Detected:

1. Object specified is not currently open.

3.3.4.1.13.3 Outputs

Reset_Read raises a STATUS_ERROR when the specified file is not open.

3.3.4.1.14 Reset Current Write Position - Reset_Write

This function resets the current write position of a specified object in the KAPSE data base. Reset_Write is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Reset_Write calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.14.1 Inputs

There is one input argument defined for Reset_Write:

File - The internal name of the file in which the current write position is to be reset.

3.3.4.1.14.2 Processing

Reset_Write locates the object control block corresponding to the object specified by File, accesses the control block, resets the current write position to one, and returns.

Errors Detected:

1. Object specified is not currently open.

3.3.4.1.14.3 Outputs

Reset_Write raises a STATUS_ERROR when the specified file is not open.

3.3.4.1.15 Get Current Size of File - Size

This function returns the current size of a specified object in the KAPSE data base. Size is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Size calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.15.1 Inputs

There is one input argument defined for Size:

File - The internal name of the file of which its current size is to be returned.

3.3.4.1.15.2 Processing

Size locates the object control block corresponding to the object specified by File, accesses the control block, gets the current size of the specified file, and returns.

Errors Detected:

1. Object specified is currently not open.

3.3.4.1.15.3 Outputs

Size returns the current size of the specified file or raises a STATUS_ERROR when the specified file is not open.

3.3.4.1.16 Get End Position of File - Last

This function returns the current end position of a specified object in the KAPSE data base. Last is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Last calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.16.1 Inputs

There is one input argument defined for Last:

File - The internal name of the file in which its end position is to be returned.

3.3.4.1.16.2 Processing

Last locates the object control block corresponding to the object specified by File, accesses the control block, gets the end position of the specified file, and returns.

Errors Detected:

1. Object specified is currently not open.

3.3.4.1.16.3 Outputs

Last returns the end position of the specified file or raises a STATUS_ERROR when the specified file is not open.

3.3.4.1.17 Check End of File - End_Of_File

This function determines whether the end of file has been reached for the specified object in the KAPSE data base. End_Of_File is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. End_Of_File calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.17.1 Inputs

There is one input argument defined for End_Of_File:

File - The internal name of the file in which the end of file condition is to be checked.

3.3.4.1.17.2 Processing

End_Of_File locates the object control block corresponding to the object specified by File, accesses the control block and determines whether the current read position exceeds the the end position of the file.

Errors Detected:

1. Specified object is not currently open.

3.3.4.1.17.3 Outputs

End_Of_File returns a boolean indicating whether the end of file has been reached, or raises the STATUS_ERROR if the specified file is not open.

3.3.4.1.18 Truncate a File - Truncate

This function truncates the specified object in the KAPSE data base. Truncate is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Truncate calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.18.1 Inputs

There are two input arguments defined for Truncate:

File - The internal name of the file in which the end of file condition is to be checked.

To - The index in which the end pointer must be reset.

3.3.4.1.18.2 Processing

Truncate locates the object control block corresponding to the object specified by File, accesses the control block and sets the end position to the number specified by To.

Errors Detected:

1. Value of end position must be greater than the reset value.

2. Object specified is not currently open.

3.3.4.1.18.3 Outputs

Truncate raises the USE_ERROR exception if the specified index is greater than the current end position of the file or raises the STATUS_ERROR if the specified file is not open.

3.3.4.1.19 Create Interprocess Communication - Pipe

This function creates a mechanism for interprocess communication in the MAPSE. Pipe is included in the Run-time Support Package, and its specification is made visible as part of the virtual interface. Pipe calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.1.19.1 Inputs

There are two input arguments defined for Pipe:

Relocbi - The relative index of the input object control block for the requesting process.

Relocbo - the relative index of the output object control block for the requesting process.

3.3.4.1.19.2 Processing

Pipe takes the object control blocks specified by Relocbi and Relocbo, creates two more object blocks with the same characteristics, and returns the indexes of the newly created block.

Errors Detected:

1. None

3.3.4.1.19.3 Outputs

There is one output argument defined for Pipe; it indicates the success of execution or an error condition. If Pipe was successful, two relative indexes to the newly created object control blocks are returned to the requesting process.

3.5.4.2 Ada Text File Input/Output

Facilities are available for I/O in human-readable form, with the external file consisting of characters. The package defining these facilities is called TEXT_IO. It uses the the general INPUT_OUTPUT package of files of type CHARACTER, so all the facilities described in the following sections are available. In addition to these general facilities, procedures are provided to get values of suitable types from external files of characters, and put values to them, carrying out conversions between the internal values and appropriate character strings.

All the Get and Put procedures have an Item parameter, whose type determines the details of the action and determines the appropriate character string in the external file. Note that the Item parameter is an out parameter for Get and an in parameter for Put. The general principle is that the characters in the external file are composed and analyzed as lexical elements.

For all Get and Put procedures, there are forms with and without files specified. If a file is specified, it must be of the correct type (IN_FILE for Get, OUT_FILE for Put). If no file is specified, a default input and output files are the so-called standard input file and standard output file, which are open and associated with two defined files.

Although the package TEXT_IO is defined in terms of the package INPUT_OUTPUT, the execution of an operation of one of these packages need not have a well defined effect on the execution of subsequent operations of the other package.

3.3.4.2.1 Get Initial Default Input File - Standard_Input

This function returns the initial default input file for the requesting process. Standard_Input is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Standard Input calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.1.1 Inputs

There are no input arguments defined for Standard_Input.

3.3.4.2.1.2 Processing

Standard_Input accesses and checks for the initial default input file and returns its name.

Errors Detected:

1. None

3.3.4.2.1.3 Outputs

Standard_Input returns the name of the default initial input file for the MAPSE process.

3.3.4.2.2 Get Initial Default Output File - Standard_Output

This function returns the name of the initial default output file for the requesting process. Standard_Output is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Standard_Output calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.2.1 Inputs

There are no input arguments defined for Standard_Output.

3.3.4.2.2.2 Processing

Standard_Output accesses and checks for the initial default output file and returns its name.

Errors Detected:

1. None

3.3.4.2.2.3 Outputs

Standard_Output returns the name of the default initial output file for the MAPSE process.

3.3.4.2.3 Get Current Default Input File - Current_Input

This function returns the current default input file for a requesting process. Current_Input is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Current_Input calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.3.1 Inputs

There are no input arguments defined for Current_Input.

3.3.4.2.3.2 Processing

Current_Input accesses the checks for the existence of the current default input file and return its name.

Errors Detected:

1. No default input file defined.

3.3.4.2.3.3 Outputs

Current_Input returns the name of the default current input file for the MAPSE process, or raises the STATUS_ERROR exception if there exists no current default input file.

3.3.4.2.4 Get Current Default Output File - Current_Output

This function returns the name of the current output file for a requesting process. Current_Output is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Current_Output calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.4.1 Inputs

There are no input arguments defined for Current_Output.

3.3.4.2.4.2 Processing

Current_Output accesses and checks for the existence of the current default output file and returns its name.

Errors Detected:

1. No current default output file defined.

3.3.4.2.4.3 Outputs

Current_Output returns the name of the default current output file for the MAPSE process or raises the STATUS_ERROR if there is no current output file.

3.3.4.2.5 Set Current Default Input File - Set_Input

This function sets the current default input file for the requesting process. Set_Input is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Set_Input calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.5.1 Inputs

There is one input argument defined for Set_Input:

File - The name of the file that is to become the default input file.

3.3.4.2.5.2 Processing

Set_Input sets the default current input file to the object specified by File.

Errors Detected:

1. Object specified is not currently open.

3.3.4.2.5.3 Outputs

Set_Input raises the STATUS_ERROR exception if the specified file is not open.

3.3.4.2.6 Set Current Default Output File - Set_Output

This function sets the current default output file for the requesting process. Set_Output is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Set_Output calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.6.1 Inputs

There is one input argument defined for Set_Output:

File - The name of the file that is to become the default output file.

3.3.4.2.6.2 Processing

Set_Output sets the current default output file to the object specified by File.

Errors Detected:

1. Object specified is currently not open.

3.3.4.2.6.3 Outputs

Set_Output raises the STATUS_ERROR exception if the specified file is not open.

3.3.4.2.7 Get Current Column Number - Col

This function returns the current column number for the next get or put to a specified object in the KAPSE data base. Col is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Col calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.7.1 Inputs

There is one input argument defined for Col:

File - The name of the file in which the current column is to be returned.

3.3.4.2.7.2 Processing

Col locates the object control block associated with the object specified by File and returns the current column for the next get or put.

Errors Detected:

1. Object specified is currently not open.

3.3.4.2.7.3 Outputs

Col returns the current column number for the next get or put operation or raises the STATUS_ERROR exception if the specified file is not open.

3.3.4.2.8 Set Current Column Number - Set_Col

This function sets the current column number of the next get or put for the specified object in the KAPSE data base. Set__Col is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Set_Col calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.8.1 Inputs

There are two input arguments defined for Set_Col:

File	- The name of the file in which the current column is to be set.
To	- The column in which the current column is to be set to.

3.3.4.2.8.2 Processing

Set__Col locates the object control block associated with the object specified by File and sets the current column to the value specified by To. The value of To must be less than the line length for the object.

Error Detected:

1. Object specified is not currently open.
2. Value to which the column is set is greater than the line length for the object.

3.3.4.2.8.3 Outputs

Set_Col returns no arguments but raises the STATUS_ERROR if the specified file is not open, or raises the LAYOUT_ERROR if the line length for the object is less than the new column value.

3.3.4.2.9 Get Current Line Number - Line

This function returns the current line number of a specified object in the KAPSE data base. Line is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Line calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.9.1 Inputs

There is one input argument defined for Line:

File - The name of the file in which the current line number is to be returned.

3.3.4.2.9.2 Processing

Line locates the object control block associated with the object specified by File and retrieves the current line length for the specified object.

Errors Detected:

1. Object specified is not currently open.

3.3.4.2.9.3 Outputs

Line returns the current line number for the next get or put to the file, or raises the STATUS_ERROR exception if the specified file was not open.

3.3.4.2.10 Start a New Line - New_Line

This function starts a new line in the specified object of the KAPSE data base. New_Line is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. New_Line calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.10.1 Inputs

There are two input arguments defined for New_Line:

File	- The name of the file in which a new line is to be started.
N	- The number of lines to increment the current line number.

3.3.4.2.10.2 Processing

New_Line locates the object control block associated with the object specified by File, resets the current column number to one, and increments the current line number by the value contained in Spacing. A spacing of one corresponds to single spacing, a spacing of two to double spacing. New_Line terminates the current line and adds spacing-minus-one empty lines. When the line length is fixed, extra space characters are inserted where needed to fill the current line and add empty lines.

Errors Detected:

1. Object specified is not currently open.

3.3.4.2.10.3 Outputs

New_Line raises the STATUS_ERROR exception if the specified file is not open.

3.3.4.2.11 Skip Lines - Skip_Line

This function skips a specified number of lines in a specified object of the KAPSE data base. Skip_Line is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Skip_Line calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.11.1 Inputs

There are two input arguments defined for Skip_Line:

File - The name of the file in which line are to be skipped.

N - The number of lines to be skipped.

3.3.4.2.11.2 Processing

Skip_Line locates the object control block associated with the object specified by File. Skip_Line resets the current column number to one and increments the current line number by the value specified by N. A value of N greater than one causes spacing-minus-one lines to be skipped as well as the remainder of the current line.

Errors Detected:

1. Object specified is not currently open.

3.3.4.2.11.3 Outputs

Skip_Line raises the STATUS_ERROR exception if the specified file was not open.

3.3.4.2.12 Check if End of Line - End_Of_Line

This function determined whether the end of line has been reached for a specified object in the KAPSE data base. End_Of_Line is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. End_Of_Line calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.12.1 Inputs

There is one input argument defined for End_Of_Line:

File - The name of the file in which to check if at end of line.

3.3.4.2.12.2 Processing

End_Of_Line locates the object control block associated with the object specified by File and checks if at end of line.

Errors Detected:

1. Object specified is not currently open.

3.3.4.2.12.3 Outputs

End_of_Line returns a boolean true if the line length of the specified input file is not set, and the current column number exceeds the length of the current line (that is, if there are no more characters to be read on the current line), otherwise false, or raises the STATUS_ERROR exception if the specified file is not open.

3.3.4.2.13 Set Line Length - Set_Line_Length

This function sets the line length for a specified object in the KAPSE data base. Set_Line_Length is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Set_Line_Length calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.13.1 Inputs

There are two input arguments defined for Set_Line_Length:

File - The name of the file in which line length is to be set.
N - The new line length of a file.

3.3.4.2.13.2 Processing

Set_Line_Length locates the object control block associated with the object specified by File. The line length is set to the value specified by N for the specified object. The value zero indicates that the line length is not set; it is the initial value for any file.

Errors Detected:

1. Line mark does not correspond to the specified line length.

2. Object specified is not currently open.

3.3.4.2.13.3 Outputs

Set_Line_Length raises the STATUS_ERROR exception if the specified file was not open. The LAYOUT_ERROR exception is raised by a Get operation if a line mark does not correspond to the specified line length.

3.3.4.2.14 Get Line Length - Line_Length

This function gets the current line length for a specified object in the KAPSE data base. Line_Length is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Line Length calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.14.1 Inputs

There is one input argument defined for Line_Length:

File - The name of the file in which to get the current line length.

3.3.4.2.14.2 Processing

Line_Length locates the object control block associated with the object specified by File and gets the current line length of the specified object. The value zero indicates that the line length is not set.

Errors Detected:

1. Object specified is not currently open.

3.3.4.2.14.3 Outputs

Line_Length raises the STATUS_ERROR exception if the specified file was not open.

3.3.4.2.15 Get a Character - Get

This function gets the current character in a specified object of the KAPSE data base. Get is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Get calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.15.1 Inputs

There is one input argument defined for Get:

File - The name of the file in which to get the current character.

3.3.4.2.15.2 Processing

Get locates the object control block associated with the object specified by File. The current character is retrieved from the specified input object at the position given by the current line number and the current column number. Get adds one to the current column number, unless the line length is fixed and the current column number equals the line length, in which case the current column number is set to one and the current line number is increased by one.

Errors Detected:

1. Object specified is not currently open.
2. Line mark does not correspond to the specified line length.

3.3.4.2.15.3 Outputs

Get returns the current character in the specified input file or raises the STATUS_ERROR exception if the specified file was not open or raises the LAYOUT_ERROR when the line mark does not correspond to the specified line length.

3.3.4.2.16 Put a Character - Put

This function puts a character in a specified object of the KAPSE data base. Put is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Put calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.16.1 Inputs

There are two input arguments defined for Put:

File	- The name of the file in which to put a character.
Item	- The character to put in the specified file.

3.3.4.2.16.2 Processing

Put locates the object control block associated with the object specified by File. The character specified by Item is written to the specified output file on the current column and current line. Put adds one to the current column number, unless the line length is fixed and the current column number equals the line length, in which case a line mark is output, the current column is set to one, and the current line number is increased by one.

Errors Detected:

1. Object specified is currently not open.

3.3.4.2.16.3 Outputs

Put raises the STATUS_ERROR exception if the specified file is not open.

3.3.4.2.17 Get a String - Get_String

This function gets the next sequence of characters in a specified object of the KAPSE data base. Get_String is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Get_String calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.17.1 Inputs

There is one input argument defined for Get_String:

File - The name of the file in which to get the next sequence of characters.

3.3.4.2.17.2 Processing

Get_String locates the object control block associated with the object specified by File. Get_String performs get operations on the specified input file, skipping any leading blanks (that is, spaces, tabulation characters, or line marks) and returns as a result the next sequence of characters up to (and not including) a blank.

Errors Detected:

1. Object specified is currently not open.

3.3.4.2.17.3 Outputs

Get_String returns the next sequence of characters in the specified input file or raises the STATUS_ERROR exception if the specified file is not open.

74

3.3.4.2.18 Get a Line - Get_Line

This function gets the next sequence of character in a specified object of the KAPSE data base. Get_Line is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Get_Line calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.18.1 Inputs

There is one input argument defined for Get_Line:

File - The name of the file in which to get a line of characters.

3.3.4.2.18.2 Processing

Get__Line locates the object control block associated with the object specified by File_Get_Line performs get operations on the specified input file, returning the next sequence of characters up to, but not including, a line mark. If the input line is already at the end of a line, a null string is returned. The input file is advanced just past the line mark, so successive calls to Get_Line return successive lines.

Errors Detected:

1. Object specified is not currently open.

3.3.4.2.18.3 Outputs

Get_Line returns the next sequence of characters in the specified input file or raises the STATUS_ERROR exception if the specified file is not open.

3.3.4.2.19 Put a Line - Put_Line

This function outputs a string of text to a specified object in the KAPSE data base. Put_Line is included in the Run-time Support Package, and its specification is visible as part of the virtual interface. Put_Line calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.4.2.19.1 Inputs

There are two input arguments defined for Put_Line:

File	- The name of the file in which to put a line of characters.
Item	- The sequence of character to put to the specified file.

3.3.4.2.19.2 Processing

Put__Line locates the object control block corresponding to the object referenced by File. Put__Line performs put operations on the specified output object to write a string of characters specified by Item to the specified file and appends a line mark.

Errors Detected:

1. Specified object is currently not open.

3.3.4.2.19.3 Outputs

Put_Line raises the STATUS_ERROR exception if the specified file is not open.

3.3.5 Version Support

This section describes facilities which provides the MAPSE user to access versioned objects. These facilities are included in the KDBS Utility Package and their specifications are made visible through the KAPSE virtual interface. See Figure 3-19 for a logical break down of Version Support functions.

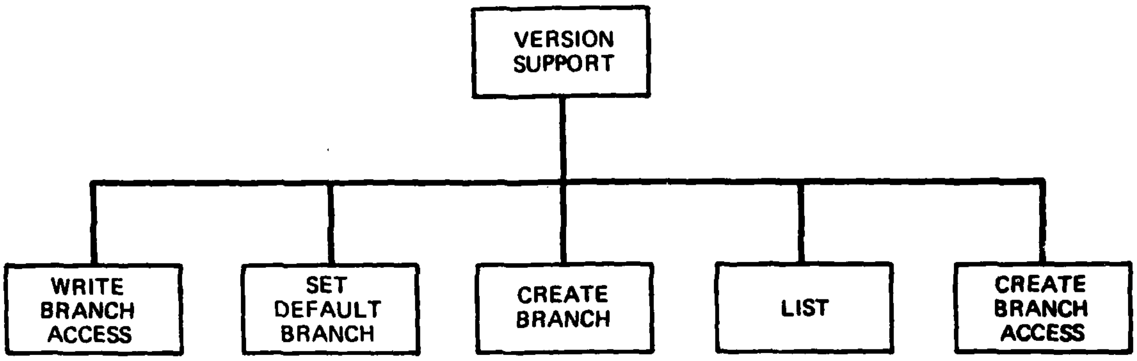


Figure 3-19. Version Support Functions

3.3.5.1 List Versions - Listv

This function lists information about all versions of a specified abstract object. Listv is included in the KDBS Utiltity Package, and its specification is visible as part of the virtual interface. Listv calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.5.1.1 Inputs

There is one input argument defined for Listv:

Oname - The name of the abstract object in which information is to be retrieved.

3.3.5.1.2 Processing

Listv locates the object specified by Oname and retrieves information about all versions of the abstract object.

Errors Detected:

1. Requesting process does not have "read" permission to the abstract object.
2. Object specified does not exist.
3. Object specified is not under version control.

3.3.5.1.3 Outputs

There is one output argument defined for Listv; it indicates the success of execution or an error condition. If Listv is successful, information about the versions of the abstract object is returned to the requesting process.

3.3.5.2 Create Branch Access - Cbranch_Access

This function sets the create branch access area defined in the abstract object. Cbranch_Access is included in the KDBS Utility Package, and is visible as part of the virtual interface. Cbranch_Access calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.5.2.1 Inputs

There are three arguments defined as input to Cbranch Access:

Obname - The name of the object in which the create branch access is to be set.

Uname - The name of the user or group in which the create branch access is to be set.

Atype - The indicator of whether to add or delete the user or group name.

3.3.5.2.2 Processing

Cbranch_Access locates the object specified by Obname and adds or deletes, depending on the value of Atype, the user or group name specified by Uname to the area in the abstract object to create a branch in the version structure.

Errors Detected:

1. Requesting process does not have "mod" permission to the abstract object.
2. Object specified is not under version control.
3. Object specified does not exist.

3.3.5.2.3 Outputs

There is one output argument defined for Cbranch_Access; it indicates success of execution or an error condition.

3.3.5.3 Write Branch Access - Wbranch_Access

This function sets the write branch access area defined in the abstract object. Wbranch__Access is visible as part of the virtual interface. Wbranch_Access calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.5.3.1 Inputs

There are three arguments defined as input to Wbranch-Access:

Obname - The name of the object and branch in which write access is to be set.

Uname - The name of the user or group in which write branch access is to be set.

Atype - The indicator of whether to add or delete the user or group name.

3.3.5.3.2 Processing

Wbranch_Access locates the object specified by Obname and adds or deletes, depending on the value of Atype, the user or group name specified by Uname to the area in the abstract object to write a new version on a branch of the version structure.

Errors Detected:

1. Requesting process does not have "mod" permission to the abstract object.
2. Object specified is not under version control.
3. Object specified does not exist.
4. Branch specified does not exist.

3.3.5.3.3 Outputs

There is one output argument for Wbranch_Access; it indicates success of execution or an error condition.

/

3.3.5.4 Create Branch - Cbranch

This function creates a branch in the version tree structure at a specified version. Cbranch is included in the KDBS Utility Package, and is visible as part of the virtual interface. Cbranch calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.5.4.1 Inputs

There are two arguments defined as input to Cbranch:

Ovname - The name of the object an version in which a branch is to be created.

Bname - The name to be associated with the created branch.

3.3.5.4.2 Processing

Cbranch locates the object specified by Ovname and creates a branch with the name specified by Bname.

Errors Detected:

1. Requesting process does not have "create branch" access to the specified object.
2. Object specified does not exist.
3. Object specified is not under version control.
4. Branch already exists with same name specified.

3.3.5.4.3 Outputs

There is one output argument defined for Cbranch; it indicates the success of execution or an error condition.

3.3.5.5 Set Default Version - Set_Dversion

This function sets the default version to be accessed on relative references to the object. Set_Dversion is included in the KDBS Utility Package, and is visible as part of the virtual interfare. Set_Dversion calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.5.5.1 Inputs

There is one input argument defined for Set_Dversion:

Ovname - The name of the version to be the default in relative references to the object.

3.3.5.5.2 Processing

Set_Dversion locates the object specified by Ovname and sets the area in the abstract object to the version specified to be the default.

Errors Detected:

1. Requesting process does not have "mod" access to the specified object.
2. Object specified does not exist.
3. Version specified is invalid.
4. Object specified is not under version control.

3.3.5.5.3 Outputs

There is one output argument defined for Set_Dversion; it indicates the success of execution or an error condition.

3.3.6 Archive Support

This section describes facilities that enable the user of MAPSE to archive objects. These facilities are included in the KDBS Utility Package and their specifications are made visible through the KAPSE virtual interface. See Figure 3-20 for a logical break down of Archive Support functions.

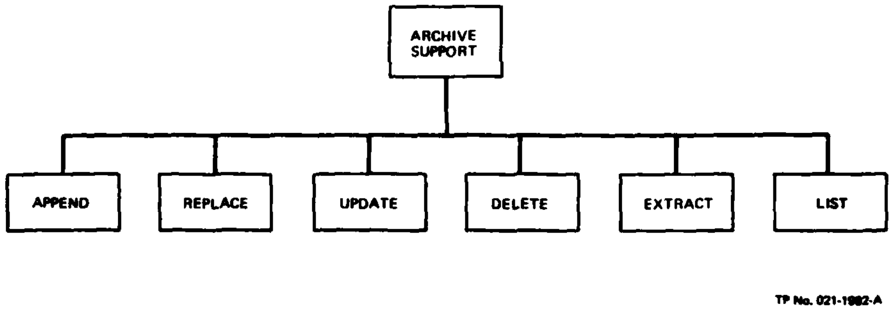


Figure 3-20. Archive Support Functions

3.3.6.1 Archive Append - Aarchive

This function adds an archive member and creates an archive object if one does not exist. Aarchive is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Aarchive calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.6.1.1 Inputs

There are two input arguments defined for Aarchive:

Aoname - The name of the archive object in which an object is to be archived.

Oname - The name of the object to be archived.

3.3.6.1.2 Processing

Aarchive locates the archive object specified by Aoname. An archive object is created if one does not exist. The object specified by Oname is added to the membership of the archive.

Errors Detected:

1. Requesting process does not have "write" access in the specified partition in order to create a partition.
2. Object specified does not exist.
3. Requesting process does not have "write" access to the archive object.
4. Requesting process does not have "read" access to the object to be archived.

3.3.6.1.3 Outputs

There is one output argument defined for Aarchive; it indicates success of execution or an error condition.

3.3.6.2 Archive Replace - Rarchive

This function replaces or adds an object to the archive and creates an archive object if one does not exist. Rarchive is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Rarchive calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.6.2.1 Inputs

There are two input arguments defined for Rarchive:

Aoname - The name of the archive in which a member is to be replaced.
Oname - The name of the object to be replaced in the archive object.

3.3.6.2.2 Processing

Rarchive locates the archive object specified by Aoname. An archive object is created if one does not exist and replaces or adds the object specified by Oname to the archive.

Errors Detected:

1. Requesting process does not have "write" access to the archive object.

2. Requesting process does not have "read" access to the object specified.

3. Requesting process does not have "write" access to create the archive object.

4. Object specified does not exist.

3.3.6.2.3 Outputs

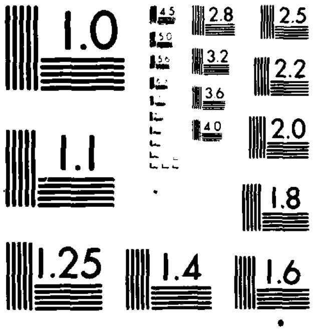
There is one output argument defined for Rarchive; it indicates success of execution or an error condition.

AD-A109 980 COMPUTER SCIENCES CORP FALLS CHURCH VA F/G 9/2
ADA INTEGRATED ENVIRONMENT II COMPUTER PROGRAM DEVELOPMENT SPEC--ETC(U)
DEC 81 F30602-80-C-0292

UNCLASSIFIED RADC-TR-81-364-PT-1 NL

4 OF 4
AD
A109 980

END
DATE
FILMED
3-82
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

3.3.6.3 Archive Update - Uarchive

This function updates a member of the archive object. The update only takes place when the date/time stamp of the object is more recent than those of the associated members. Uarchive is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Uarchive calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.6.3.1 Inputs

There are two input arguments defined for Uarchive:

Aoname - The name of the archive object in which a member is to be updated.

Oname - The name of the object to be used in updating a member in the archive.

3.3.6.3.2 Processing

Uarchive locates the archive object specified by Aoname. The member of the archive is updated only when its date/time stamp is later than the object to be archived.

Errors Detected:

1. Requesting process does not have "write" access to the archive object.

2. Requesting process does not have "read" access to the object specified.

3. Requesting process does not have "write" access to create the archive object.

4. Object specified does not exist.

5) A member does not exist corresponding to the object to be archived.

3.3.6.3.3 Outputs

There is one output argument defined for Uarchive; it indicates success of execution or an error condition.

3.3.6.4 Archive Delete - Darchive

This function deletes a member from a specified archive object in the KAPSE data base. This function deletes only a member of the archive, not an entire archive object. Darchive is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Darchive calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.6.4.1 Inputs

There are two input arguments defined for Darchive:

Aoname	- The name of the archive in which a member is to be deleted.
Mname	- The name of the member to be deleted from the archive.

3.3.6.4.2 Processing

Darchive locates the archive object specified by Aoname. The member corresponding to the name specified by Mname is deleted from the archive.

Errors Detected:

1. Requesting process does not have "delete" access to the archive object.
2. Archive specified does not exist.
3. Member specified does not exist.

3.3.6.4.3 Outputs

There is one output argument defined for Darchive; it indicates the success of execution or an error condition.

3.3.6.5 Archive Extract - Earchive

This function copies a member of a specified archive into the KAPSE data base. Earchive is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Earchive calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.6.5.1 Inputs

There are two input arguments defined for Earchive:

Aoname - The name of the archive in which a member is to be extracted.
Mname - The name of the member to be extracted from the archive.

3.3.6.5.2 Processing

Earchive locates the archive object specified by Aoname. A copy of the member specified by Mname is made into the KAPSE data base.

Errors Detected:

1. Requesting process does not have "read" access to the specified archive object.
2. Archive specified does not exist.
3. Member specified does not exist.

3.3.6.5.3 Outputs

There is one output argument defined for Earchive; it indicates the success of execution or an error condition.

3.3.6.6 Archive List - Larchive

This function retrieves the membership list of a specified archive. Larchive is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Larchive calls an entry point of the same name in the KDBS Kernel to perform the privileged operations associated with the function.

3.3.6.6.1 Inputs

There is one argument passed in the call to Larchive:

Aoname - the name of the archive in which a list of members is to be built.

3.3.6.6.2 Processing

Larchive locates the archive object specified by Aoname. A list is retrieved and returend to the requesting process.

Errors Detected:

1) Requesting process does not have "read" access to the specified archive object.

2) Archive specified does not exist.

3.3.6.6.3 Outputs

There is one output argument defined for Larchive, which indicates success of execution or an error condition. If Larchive is successful then a membership list of the archive is returned.

3.3.7 KDB Backup/Restore Support

This section describes facilities that enable the MAPSE user to Backup and Restore selected portion of the KDB. These facilities are included in the KDBS Utility Package and their specifications are made visible through the KAPSE virtual interface. See Figure 3-21 for a logical breakdown of Backup/Restore Support functions.

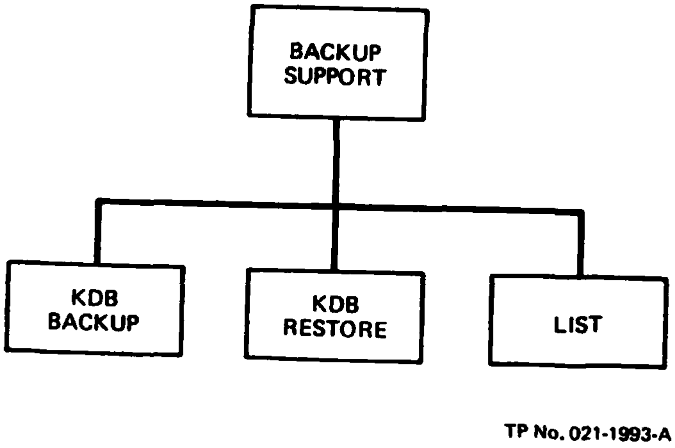


Figure 3-21. Backup/Restore Support Functions

3.3.7.1 KAPSE Data Base Backup Facilities - Backup

This function performs a backup of selected portions of the KAPSE data base. Backup is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Backup calls an entry point of the same name in the KDBS Kernel to perform privileged operations associated with the function.

3.3.7.1.1 Inputs

There is one input argument defined for Backup:

Pname - The starting partition for the backup.

3.3.7.1.2 Processing

Backup locates the partition specified by Pname. All members of the specified partition are then copied to an external medium for later use.

Error Detected:

1. Requesting process does not have "read" access to the specified partition object.

3.3.7.1.3 Outputs

There is one output argument defined for Backup, it indicates the success of execution or an error condition.

3.3.7.2 KAPSE Data Base Restore Facilities - Restore

This function restores a selected portion of the KAPSE data base. Restore is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Restore calls an entry point of the same name in the KDBS Kernel to perform privileged operations associated with the function.

3.3.7.2.1 Inputs

There is one input argument defined for Restore:

Oname - The object name which is to be restored.

3.3.7.2.2 Processing

Restore locates the object specified by Oname. A search is made of the backup medium for the specified object. If the object is a partition, the entire structure under the partition is also restored.

Errors Detected:

1. Object specified does not exist on the backup medium.
2. Requesting process does not have "write" access to the specified object.

3.3.7.1.3 Outputs

There is one output argument for Restore; it indicates the success of execution or an error condition.

3.3.7.3 List Backup - Lbackup

This function lists the objects on the backup medium. Lbackup is included in the KDBS Utility Package, and its specification is visible as part of the virtual interface. Lbackup calls an entry point of the same name in the KDBS kernel to perform privileged operations associated with the function.

3.3.7.3.1 Inputs

There is one input argument defined for Lbackup:

Bck_Name - The name of the backup medium

3.3.7.3.2 Processing

Lbackup locates the backup medium specified by Bck_Name and lists the resident objects to Standard_Output.

Errors Detected:

1. None.

3.3.7.3.3 Outputs

There is one output argument defined for Lbackup; it indicates the success of execution or an error condition.

3.4 ADAPTATION

This section describes any adaptation that might be required to rehost the KDBs.

3.4.1 General Environment

The mapping performed from the logical to physical representation will probably differ on each implementation of the KDBs. This mapping may result in a change to the object control block buffer used for I/O in order to better utilize more efficient blocking factors of the host storage system.

3.4.2 System Parameters

The object control blocks that the KDBS accesses and controls may be parameterized for ease of portability of the KDBs.

3.4.3 System Capacities

An implementation may place some limitations on the number of host files that a process may have open; however, no logical design limitations exist.

3.5 CAPACITY

None known.

SECTION 4 - QUALITY ASSURANCE PROVISIONS

4.1 INTRODUCTION

This section contains the requirements for verification of the performance of the KAPSE Data Base System (KDBS). The test levels, verification methods, and test requirements for the detailed functional requirements in Section 3 are specified in this section. The verification requirements specified herein shall be the basis for the preparation and validation of detailed test plans and procedures for the KDBS. Testing shall be performed at the subprogram, program (CPCI), system integration, and acceptance test levels. The performance of all tests, and the generation of all reports describing test results, shall be in accordance with the Government approved CPDP and the Computer Program Test Procedures.

The verification methods that shall be used in subprogram and program testing include the methods described below:

1. Inspection - Inspection is the verification method requiring visual examination of printed materials such as source code listings, normal program printouts, and special printouts not requiring modification of the CPCI. This might include inspection of program listings to verify proper program logic flow.

2. Analysis - Analysis is the verification of a performance or design requirement by examination of the constituent elements of a CPCI. For example, a parsing algorithm might be verified by analysis.

3. Demonstration - Performance or design requirements may be verified by visual observation of the system while the CPCI is executing. This includes direct observance of all display, keyboard, and other peripheral devices required for the CPCI.

4. Review of Test Data - Performance or design requirements may be verified by examining the data output when selected input data are processed. For example, a review of hard copy test data might be used to verify that the values of specific parameters are correctly computed.

5. Special Tests - Special tests are verification methods other than those defined above and may include testing one functional capability of the CPCI by observing the correct operation of other capabilities.

These verification methods shall be used at various levels of the testing process. The levels of testing to be performed are described in the paragraphs below. Data obtained from previous testing will be acceptable in lieu of testing at any level when certified by CSC/SEA and found adequate by the RADC representative. Any test performed by CSC/SEA may be observed by RADC representatives whenever deemed necessary by RADC.

Table 4-1 specifies the verification method for each functional requirement given in Section 3 of this specification. The listing in Table 4-1 of a Section 3 paragraph defining a functional requirement implies the listing of any and all subparagraphs. The verification methods required for the subparagraphs are included in the verification methods specified for the functional requirement. Acceptance test requirements are discussed in Paragraph 4.3.

Table 4-1. Test Requirements Matrix

SECTION	TITLE	INSP.	ANAL.	DEMO.	DATA.	SECTION NO.
3.3.1	Attribute Support				X	4.2
3.3.2	Partition Support				X	4.2
3.3.3	Access Support				X	4.2
3.3.4	Ada I/O Support				X	4.2
3.3.5	Version Support				X	4.2
3.3.6	Archive Support				X	4.2
3.3.7	Backup/Restore Support				X	4.2

4.1.1 Subprogram Testing

Following unit testing, individual modules of the KDBS shall be integrated into the evolving CPCI and tested to determine whether software interfaces are operating as specified. This integration testing shall be performed by the development staff in coordination with the test group. The development staff shall ensure that the system is integrated in accordance with the design, and the test personnel shall be responsible for the creation and conduct of integration tests.

4.1.2. Program (CPCI) Testing

This test is a validation of the entire CPCI against the requirements as specified in this specification.

CPCI testing shall be performed on all development software of the KDBS. This specification presents the performance criteria which the developed CPCI must satisfy. The correct performance of the KDBS will be verified by testing its major functions. Successful completion of the program testing that the majority of programming errors have been eliminated and that the program is ready for system integration. The method of verification to be used in CPCI testing shall be review of test data. CPCI testing shall be performed by the independent test team.

4.1.3. System Integration Testing

System integration testing involves verification of the integration of the KDBS with other computer programs and with equipment. The integration tests shall also verify the correctness of man/machine interfaces, and demonstrate functional completeness and satisfaction of performance requirements.

System integration testing shall begin in accordance with the incremental development procedures as stated in the CPDP. Final system integration shall occur subsequent to the completion of all the CPCIs comprising the MAPSE system. Two major system integration tests shall be performed: one for the IBM VM/370 implementation and one for the Interdata 8/32 implementation. The method of verification used for system integration testing shall be the review of test data.

The test team shall be responsible for planning, performing, analyzing monitoring, and reporting the system integration testing.

4.2 TEST REQUIREMENTS

Quality assurance tests shall be conducted to verify that the KDBS performs as required by Section 3 of this specification. Table 4-1 specifies the methods that shall be used to verify each requirement. The last column refers to a brief description of the specified types of verification as given below. Test plans and procedures shall be prepared to provide details regarding the methods and processes to be used to verify that the developed CPCI performs as required by this specification. These test plans and procedures shall contain test formulas, algorithms, techniques, and acceptable tolerance limits, as applicable.

All programs described in Table 4-1 will be tested using driver programs and examining output data. Drivers shall be written to generate input data and to log output data. Test input scripts and expected test output shall be developed by test personnel in accordance with subprogram and program specifications. Testing shall consist of comparing expected output data with test output data.

4.3. ACCEPTANCE TESTING

Acceptance testing shall involve comprehensive testing at the CPCI level and at the system level. The CPCI acceptance tests shall be defined to verify that the KDBS satisfies its performance and design requirements as specified in this specification. System acceptance testing shall test that the MAPSE satisfies its functional requirements as stated in the System Specification. Acceptance testing shall be performed by review of test data.

These tests shall be conducted by the CSC/SEA team and formally witnessed by the government. Satisfactory performance of both CPCI and system acceptance tests shall result in the final delivery and acceptance of the MAPSE system.

SECTION 5 - DOCUMENTATION

5.1 GENERAL

The documents that will be produced during the implementation phase in association with the KDBS are:

1. Computer Program Development Specification (Type B5) - Update
2. Computer Program Product Specification
3. Computer Program Listings
4. Maintenance Manual
5. User`s Manual
6. Rehostability Manual

5.1.1 Computer Program Development Specification

The final KDBS B5 Specification will be prepared in accordance with DI-E-30139 and submitted 30 days after the start of Phase II.

5.1.2 Computer Program Product Specification

A type C5 Specification shall be prepared during the course of Phase II in accordance with DI-E-30140. This document will be used to specify the design of the KDBS and the development approach implementing the B5 specification. This document will provide the detailed description that will be used as the baseline for any Engineering Change Proposals.

5.1.3 Computer Program Listings

Listings will be delivered that are the result of the final compilation of the accepted KDBS. Each compilation unit listing will contain the corresponding source, cross-reference, and compilation summary. The source listing will contain the source lines from any INCLUDEd source objects.

5.1.4 Maintenance Manual

An KDBS Maintenance Manual will be prepared in accordance with DI-M-30422 to supplement the C5 and compilation listings sufficiently to permit the KDBS to be easily maintained by personnel other than the developers. The documentation will be structured to relate quickly to program source. The procedures required for maintaining the KDBS, will be described and illustrated. Sample scripts for compiling KDBS components, for relinking the KDBS in parts or as a whole, and for installing new releases will be supplied.

5.1.5 User`s Manual

A User`s Manual shall be prepared in accordance with DI-M-30421, which will contain all information necessary for the operation of the KDBS. Because of the virtual user interface presented by the KAPSE, a single manual is sufficient for all host computers. Information relevant to specific hosts will be contained in an appendix.

5.1.6 Rehostability Manual

In accordance with R&D-137-RADC and R&D-138-RADC, a manual will be prepared to describe step-by-step procedures for rehosting the KDBS on a different computer.

300

APPENDIX A - KDBS UTILITY PACKAGE DEFINITION

```
package KDBS_UTILITY is

    -- package to support attribute manipulations

    package ATTRIBUTE_SUPPORT is

         function Adda (Oname  : OBJECT_NAME;
                        Aname  : ATTR_NAME;
                        Avalue : ATTR_VALUE) return COND_TYPE;

         function Dela (Oname : OBJECT_NAME;
                        Aname : ATTR_VALUE) return COND_TYPE;

         procedure Finda (Avstring       :     ATTR_VALUE_STRING;
                          Olist          : out REF_OLIST;
                          Condition_Code : out COND_TYPE);

         procedure Lista (Oname          :     OBJECT_NAME;
                          Alist          : out REF_ALIST;
                          Condition_Code : out COND_TYPE);

         function Addv (Oname  : OBJECT_NAME;
                        Aname  : ATTR_NAME,
                        Avalue : ATTR_VALUE) return COND_TYPE;

         function Chgv (Oname  : OBJECT_NAME;
                        Aname  : ATTR_NAME;
                        Avalue : ATTR_VALUE) return COND_TYPE;

         function Delv (Oname  : OBJECT_NAME;
                        Aname  : ATTR_NAME;
                        Avalue : ATTR_VALUE) return COND_TYPE;

         procedure Readv (Oname          :     OBJECT_NAME;
                         Aname          :     ATTR_NAME;
                         Avalue         : out ATTR_VALUE;
                         Condition_Code : out COND_TYPE),

    end ATTRIBUTE_SUPPORT;

    -- package to manipulate partitions

    package PARTITION_SUPPORT is

         function Createp (Pname : PART_NAME) return COND_TYPE;

         function Deletep (Pname : PART_NAME) return COND_TYPE;
```

```
     procedure Listp (Pname          :     PART_NAME;
                      Plist          : out REF_PLIST;
                      Condition_Code : out COND_TYPE);

     function Linkc (Pname : PART_NAME;
                     Oname : OBJECT_NAME) return COND_TYPE;

     function Linkd (Pname : PART_NAME;
                     Oname : OBJECT_NAME) return COND_TYPE;

     function Findpe (Oname : OBJECT_NAME) return COND_TYPE;

end PARTITION_SUPPORT;

-- package to support access control

package ACCESS_SUPPORT is

     procedure Laccess (Oname          :     OBJECT_NAME;
                        Access_List    : out REF_ALIST;
                        Condition_Code : out COND_TYPE);

     procedure Raccess (Oname          :     OBJECT_NAME;
                        Access_Value   : out REF_AVALUE;
                        Condition_Code : out COND_TYPE);

     function Saccess (Oname  : OBJECT_NAME;
                       Ugname : USER_NAME;
                       Accval : ACCESS_VALUE) return COND_TYPE;

     function Addgm (Gname  : GROUP_NAME;
                     Uname  : USER_NAME) return COND_TYPE;

     function Createg (Gname : GROUP_NAME) return COND_TYPE;

     function Deleteg ( Gname : GROUP_NAME) return COND_TYPE;

     function Deletegm (Gname : GROUP_NAME;
                        Uname : USER_NAME) return COND_TYPE;

     function Findgm (Gname : GROUP_NAME;
                      Uname : USER_NAME) return COND_TYPE;

     procedure Listg (Glist          : out GROUP_LIST;
                      Condition_Code : out COND_TYPE);

     procedure Listgm (Gname          :     GROUP_NAME;
                       Glist          : out GROUP_LIST;
                       Condition_Code : out COND_TYPE);

end ACCESS_SUPPORT;
```

```
-- package to support version control

package VERSION_SUPPORT is

     procedure Listv (Oname          :     OBJECT_NAME;
                      Vlist          : out VERSION_LIST;
                      Condition_Code : out COND_TYPE);

     function Cbranch_Access (Obname : OBJECT_BRANCH;
                              Uname  : USER_NAME;
                              Atype  : TYPE_FUNC) return COND_TYPE;

     function Wbranch_Access (Obname : OBJECT_BRANCH;
                              Uname  : USER_NAME;
                              Atype  : TYPE_FUNC) return COND_TYPE;

     function Cbranch (Ovname : OBJECT_VERSION;
                       Bname  : BRANCH_NAME) return COND_TYPE;

     function Set_Dversion (Ovname : OBJECT_VERSION) return COND_TYPE;

end VERSION_SUPPORT;

-- package to support achiving facilities

package ARCHIVE_SUPPORT is

     function Aarchive (Aoname : ARCHIVE_NAME,
                        Oname  : OBJECT_NAME) return COND_TYPE;

     function Rarchive (Aoname : ARCHIVE_NAME;
                        Oname  : OBJECT_NAME) return COND_TYPE;

     function Uarchive (Aoname : ARCHIVE_NAME;
                        Oname  : OBJECT_NAME) return COND_TYPE;

     function Darchive (Aoname : ARCHIVE_NAME;
                        Mname  : MEMBER_NAME) return COND_TYPE;

     function Earchive (Aoname : ARCHIVE_NAME;
                        Mname  : MEMBER_NAME) return COND_TYPE;

     procedure Larchive (Aoname         :     ARCHIVE_NAME;
                         Aclist         : out ARCHIVE_LIST;
                         Condition_Code : out COND_TYPE);

end ARCHIVE_SUPPORT;
```

```
     -- package to support backup and restore facilities

     package BCKRST_SUPPORT is

          function Backup (Pname : PARTITION_NAME) return COND_TYPE;

          function Restore (Oname : OBJECT_NAME) return COND_TYPE;

          procedure Lbackup (Bck_Name       :     BACKUP_NAME;
                             Blist          : out REF_BLIST;
                             Condition_Code : out COND_TYPE);

     end BCKRST_SUPPORT;

end KDBS_UTILITY;
```

INPUT_OUTPUT PACKAGE DEFINITION

```
package INPUT_OUTPUT is
    type IN_FILE        is limited private;
    type OUT_FILE       is limited private;
    type INOUT_FILE     is limited private;
    type FILE_INDEX     is range 0 . . . implementation defined;

    --  general operations for file manipulation

    procedure Create (File : in out OUT_FILE;   Name : in STRING);
    procedure Create (File : in out INOUT_FILE; Name : in STRING);

    procedure Open (File : in out IN_FILE;    Name : in STRING);
    procedure Open (File : in out OUT_FILE;   Name : in STRING);
    procedure Open (File : in out INOUT_FILE; Name : in STRING);

    procedure Close (File : in out IN_FILE);
    procedure Close (File : in out OUT_FILE);
    procedure Close (File : in out INOUT_FILE);

    function Is_Open (File : in IN_FILE)    return Boolean;
    function Is_Open (File : in OUT_FILE)   return Boolean;
    function Is_Open (File : in INOUT_FILE) return Boolean;

    function Name (File : in IN_FILE)    return String;
    function Name (File : in OUT_FILE)   return String;
    function Name (File : in INOUT_FILE) return String;

    procedure Delete (Name : in STRING);

    function Size (File : in IN_FILE)    return File_Index;
    function Size (File : in OUT_FILE)   return File_Index;
    function Size (File : in INOUT_FILE) return File_Index;

    function Last (File : in IN_FILE)    return File_Index;
    function Last (File : in OUT_FILE)   return File_Index;
    function Last (File : in INOUT_FILE) return File_Index;

    procedure Truncate (File : in OUT_FILE;   To : in File_Index);
    procedure Truncate (File : in INOUT_FILE; To : in File_Index);

    -- input and output operations

    procedure Read (File : in IN_FILE;    Item : out ELEMENT_TYPE);
    procedure Read (File : in INOUT_FILE; Item : out ELEMENT_TYPE);

    function Next_Read (File : in IN_FILE)    return File_Index;
    function Next_Read (File : in INOUT_FILE) return File_Index;
    procedure Set_Read (File : in IN_FILE;    To : in FILE_INDEX);
    procedure Set_Read (File : in INOUT_FILE; To : in FILE_INDEX);

    procedure Reset_Read (File : in IN_FILE);
    procedure Reset_Read (File : in INOUT_FILE);

    procedure Write (File : in OUT_FILE;   Item : in ELEMENT_TYPE);
    procedure Write (File : in INPUT_FILE; Item : in ELEMENT_TYPE);

    function Next_Write (File : in OUT_FILE)   return File_Index;
    function Next_Write (File : in INOUT_FILE) return File_Index;

    procedure Set_Write (File : in OUT_FILE;   To : in FILE_INDEX);
    procedure Set_Write (File : in INOUT_FILE; To : in FILE_INDEX);

    procedure Reset_Write (File : in OUT_FILE);
    procedure Reset_Write (File : in INOUT_FILE);

    function End_Of_File (File : in IN_FILE)    return Boolean;
    function End_Of_File (File : in INOUT_FILE) return Boolean;

    procedure Pipe (Relocbi        :     REF_OCB;
                    Relocbo        :     REF_OCB;
                    Newocbi        : out REF_OCB;
                    Newocbo        : out REF_OCB;
                    Condition_Code : out COND_TYPE);

    --  exceptions that can be raised

    NAME_ERROR     :exception;
    USE_ERROR      :exception;
    STATUS_ERROR   :exception;
    DATA_ERROR     :exception;
    DEVICE_ERROR   :exception;
    END_ERROR      :exception;

end INPUT_OUTPUT;
```

TEXT_IO PACKAGE DEFINITION

```
package TEXT_IO is
    package CHARACTER_IO is new INPUT_OUTPUT (CHARACTER);

    type IN_FILE  is new CHARACTER_IO.IN_FILE;
    type OUT_FILE is new CHARACTER_IO.OUT_FILE;

    -- Character Input-Output

    procedure Get (File : in  IN_FILE;  Item : out CHARACTER);
    procedure Get (Item : out CHARACTER);
    procedure Put (File : in  OUT_FILE; Item : in CHARACTER);
    procedure Put (Item : in  CHARACTER);

    -- String Input-Output

    procedure Get (File : in  IN_FILE;  Item : out STRING);
    procedure Get (Item : out STRING);
    procedure Put (File : in  OUT_FILE; Item : in STRING);
    procedure Put (Item : in  STRING);

    function Get_String (File : in IN_FILE) return STRING;
    function Get_String return STRING;

    function  Get_Line (File : in IN_FILE) return STRING;
    function  Get_Line return STRING;
    procedure Put_Line (File : in OUT_FILE; Item : in STRING);
    procedure Put_Line (Item : in STRING);
    -- Generic package for Integer Input-Output

    generic
        type NUM is range <>;
        with function Image (X : NUM)    return String is NUM`IMAGE;
        with function Value (X : STRING) return NUM is NUM`VALUE;
    package INTEGER_IO is
        procedure Get (File  : in  IN_FILE; Item : out NUM);
        procedure Get (Item  : out NUM);
        procedure Put (File  : in  OUT_FILE;
                       Item  : in  NUM;
                       Width : in  INTEGER := 0;
                       Base  : in  INTEGER range 2 .. 16 := 10);

        procedure Put (Item  : in NUM;
                       Width : in INTEGER := 0;
                       Base  : in INTEGER range 2 .. 16 := 10);
    end INTEGER_IO;
```

```
-- Generic package for Floating Point Input_Output

generic
     type NUM is digits <>;
     with function Image (X : NUM)    return STRING is NUM`IMAGE;
     with function Value (X : STRING) return NUM is NUM`VALUE;
package FLOAT_IO is
     procedure Get (File : in  IN_FILE; Item : out NUM);
     procedure Get (Item : out NUM);

     procedure Put (File     : in OUT_FILE; Item : in NUM;
                    Width    : in INTEGER := 0;
                    Mantissa : in INTEGER := NUM`DIGITS;
                    Exponent : in INTEGER := 2);

     procedure Put (Item : in NUM; Width : INTEGER := 0;
                    Mantissa : in INTEGER := NUM`DIGITS;
                    Exponent : in INTEGER := 2);
end FLOAT_IO;

-- Generic package for Fixed Point Input_Output

generic
     type NUM is delta <>;
     with function Image (X : NUM)    return STRING is NUM`IMAGE;
     with function Value (X : STRING) return NUM is NUM`VALUE;
package FIXED_IO is
     Delta_Image      : constant STRING := IMAGE(NUM`DELTA -
                                                INTEGER(NUM`DELTA));
     Default_Decimals : constant INTEGER := DELTA_IMAGE`LENGTH - 2;

     procedure Get (File : in  IN_FILE; Item : out NUM);
     procedure Get (File : out NUM);

     procedure Put (File  : in OUT_FILE;
                    Item  : in NUM;
                    Width : in INTEGER := 0;
                    Fract : in INTEGER := Default_Decimals);

     procedure Put (Item  : in NUM;
                    Width : in INTEGER := 0;
                    Fract : in INTEGER := Default_Decimals);

end FIXED_IO;

-- Input_Output for Boolean

procedure Get (File : in  IN_FILE ; Item : out BOOLEAN);
procedure Get (Item : out BOOLEAN);
procedure Put (File       : in OUT_FILE;
               Item       : in BOOLEAN;
               Width      : in INTEGER := 0;
               Lower_Case : in BOOLEAN := FALSE);

procedure Put (Item       : in BOOLEAN;
               Width      : in INTEGER := 0;
               Lower_Case : in BOOLEAN := FALSE);

-- Generic package for Enumeration Types

generic
     type ENUM is (<>);
     with function Image (X : ENUM)   return STRING is ENUM`IMAGE;
     with function Value (X : STRING) return ENUM is ENUM`VALUE;
package ENUMERATION_IO is
     procedure Get (File : in  IN_FILE; Item : out ENUM);
     procedure Get (Item : out ENUM);

     procedure Put (File       : in OUT_FILE;
                    Item       : in ENUM;
                    Width      : in INTEGER := 0;
                    Lower_Case : in BOOLEAN := FALSE);

     procedure Put (Item       : in ENUM;
                    Width      : in INTEGER := 0;
                    Lower_Case : in BOOLEAN := FALSE);

end ENUMERATION_IO;

-- Layout control

function Line (File : in IN_FILE)  return NATURAL;
function Line (File : in OUT_FILE) return NATURAL;
function Line return NATURAL;

function Col (File : in IN_FILE)  return NATURAL;
function Col (File : in OUT_FILE) return NATURAL;

function Set_Col (File : in IN_FILE;  To : in NATURAL);
function Set_Col (File : in OUT_FILE; To : in NATURAL);
function Set_Col (To : in NATURAL);

procedure New_Line (File : in OUT_FILE; N : in NATURAL := 1);
procedure New_Line (N : in NATURAL := 1);

procedure Skip_Line (File : in OUT_FILE; N : in NATURAL := 1);
procedure Skip_Line (N : in NATURAL := 1);
```

```
    function End_Of_Line (File : in IN_FILE) return BOOLEAN;
    function End_Of_Line return BOOLEAN;

    procedure Set_Line_Length (File : in IN_FILE;  N : in INTEGER);
    procedure Set_Line_Length (File : in OUT_FILE; N : in INTEGER);
    procedure Set_Line_Length (N : in INTEGER);

    function Line_Length (File : in IN_FILE)  return INTEGER);
    function Line_Length (File : in OUT_FILE) return INTEGER);
    function Line_Length return INTEGER;

    -- Default input and output manipulation

    function Standard_Input  return IN_FILE;
    function Standard_Output return OUT_FILE;

    function Current_Input  return IN_FILE;
    function Current_Output return OUT_FILE;

    procedure Set_Input  (File : in IN_FILE);
    procedure Set_Output (File : in OUT_FILE);

    -- Exceptions

    NAME_ERROR          : exception renames CHARACTER_IO.NAME_ERROR;
    USE_ERROR           : exception renames CHARACTER_IO.USE_ERROR;
    STATUS_ERROR        : exception renames CHARACTER_IO.STATUS_ERROR;
    DATA_ERROR          : exception renames CHARACTER_IO.DATA_ERROR;
    DEVICE_ERROR        : exception renames CHARACTER_IO.DEVICE_ERROR;
    END_ERROR           : exception renames CHARACTER_IO.END_ERROR;
    LAYOUT_ERROR        : exception;

end TEXT_IO;
```

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.