

UNISYS

A Series
System Software Utilities
Operations
Reference Manual

July 1992

Printed in U S America
8600 0460-100

Priced Item



Product Information Announcement

New Release Revision Update New Mail Code

Title

A Series System Software Utilities Operations Reference Manual

This revision of the *A Series System Software Utilities Operations Reference Manual* is relative to the Mark 4.0 release of the A Series operating system software. You can use this manual with the Mark 4.0 or higher system software release.

The major changes described in this revision of the manual are

- The section concerning the DUMPALL utility has been revised to include additional information as well as a reorganized physical structure.
- Miscellaneous technical corrections have been added to other sections.

To order additional copies of this document

- United States customers call Unisys Direct at 1-800-448-1424
- All other customers contact your Unisys Subsidiary Librarian
- Unisys personnel use the Electronic Literature Ordering (ELO) system

Announcement only:

Announcement and attachments:
AS148

System: A Series
Release: Mark 4.0.0 July 1992

Part number: 8600 0460-100

UNISYS

A Series
System Software Utilities
Operations
Reference Manual

Copyright © 1992 Unisys Corporation
All rights reserved.
Unisys is a registered trademark of Unisys Corporation.

Release Mark 4.0.0

July 1992

Priced Item

Printed in U S America
8600 0460-100

The names, places, and/or events used in this publication are not intended to correspond to any individual, group, or association existing, living, or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual, living or otherwise, or that of any group or association, is purely coincidental and unintentional.

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THIS DOCUMENT. Any product or related information described herein is only furnished pursuant and subject to the terms and conditions of a duly executed agreement to purchase or lease equipment or to license software. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, special, or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

RESTRICTED RIGHTS LEGEND. Use, reproduction, or disclosure is subject to the restrictions set forth in DFARS 252.227-7013 and FAR 52.227-14 for commercial computer software.

Correspondence regarding this publication may be forwarded using the Product Information card at the back of the manual, or may be addressed directly to Unisys, Technical Publications, 25725 Jeronimo Road, Mission Viejo, CA 92691-2792.

Page Status

Page	Issue
iii	-100
iv	Blank
v through xix	-100
xx	Blank
xxi	-100
xxii	Blank
xxiii	-100
xxiv	Blank
1-1 through 1-4	-100
2-1 through 2-7	-100
2-8	Blank
3-1 through 3-90	-100
4-1 through 4-19	-100
4-20	Blank
5-1 through 5-60	-100
6-1 through 6-39	-100
6-40	Blank
7-1 through 7-36	-100
8-1 through 8-19	-100
8-20	Blank
9-1 through 9-35	-100
9-36	Blank
10-1 through 10-19	-100
10-20	Blank
11-1 through 11-11	-100
11-12	Blank
12-1 through 12-59	-100
12-60	Blank
13-1 through 13-7	-100
13-8	Blank
A-1 through A-8	-100
Glossary-1 through 10	-100
Bibliography-1 through 3	-100
Bibliography-4	Blank
Index-1 through 17	-100
Index-18	Blank

Unisys uses an 11-digit document numbering system. The suffix of the document number (1234 5678-xyz) indicates the document level. The first digit of the suffix (x) designates a revision level; the second digit (y) designates an update level. For example, the first release of a document has a suffix of -000. A suffix of -130 designates the third update to revision 1. The third digit (z) is used to indicate an errata for a particular level and is not reflected in the page status summary.

About This Manual

Purpose

The *A Series System Software Utilities Reference Manual* describes the system software utilities that are available to users of A Series systems. Each utility is used to perform unique functions.

Scope

This manual is designed to provide descriptions and examples of the syntax and explanations of how to run the software utilities.

Audience

This manual is intended to be used by system operators, applications programmers and system analysts.

Prerequisites

You should be familiar with one or more high level programming languages and with A Series operation.

How to Use This Manual

This manual is designed to present an alphabetical listing of the system software utilities and provides easy access to information.

Organization

Each section in this manual describes a different software utility or facility. This manual is divided into the following sections.

Section 1. CARDLINE Utility

This section tells how to use the CARDLINE utility to print an EBCDIC or BINARY data deck.

Section 2. COMPARE Utility

This section tells how to use the COMPARE utility to compare one or more pairs of files.

Section 3. DUMPALL Utility

This section tells how to use the DUMPALL utility to copy files from one medium to another and to generate listings of files.

Section 4. FILECOPY Utility

This section tells how to use the FILECOPY utility to simplify library maintenance by automating the creation of copy decks.

Section 5. FILEDATA Utility

This section tells how to use the FILEDATA utility to produce reports regarding files.

Section 6. INTERACTIVEXREF Utility

This section tells how to use the INTERACTIVEXREF utility interactively to provide information on the identifiers in a program.

Section 7. KEYEDIO Support

This section describes how to use the SYSTEM/KEYEDIO library to provide PLIISAM for COBOL74 and RPG files. This section also includes descriptions of keyed file structure, coarse tables, fine tables, file management, keyed file attributes, and KEYEDIO procedures.

For information about KEYEDIOII, refer to the *A Series KEYEDIOII Programming Reference Manual*.

Section 8. Mathematical Functions

This section describes the mathematical functions for the A Series systems.

Section 9. PATCH Utility

This section tells how to use the PATCH utility to merge one or more patch files into a single patch file on disk or pack.

Section 10. PL/I Indexed Sequential-Access Method (PLIISAM)

This section tells how to use a set of software routines to implement the PL/I Indexed Sequential-Access Method (PLIISAM) of storing and retrieving data records.

Section 11. RLTABLEGEN Utility

This section tells how to use the SYSTEM/RLTABLEGEN program to specify the format of tape labels defined at your installation.

Section 12. SORT Utility

This section tells how to use the SORT utility.

Section 13. XREFANALYZER Utility

This section tells how to use the XREFANALYZER utility to construct detailed information about all identifiers declared in a program.

Appendix A. Understanding Railroad Diagrams

This appendix explains how to read the railroad diagrams used to show the syntax of expressions and commands.

In addition, a glossary, a bibliography, and an index appear at the end of this manual.

Related Product Information

A Series File Attributes Programming Reference Manual (form 8600 0064). Formerly the A Series I/O Subsystem Reference Manual

This manual contains information about each file attribute and each direct I/O buffer attribute. The manual is written for programmers and operations personnel who need to understand the functionality of a given attribute. The *A Series I/O Subsystem Programming Guide* is a companion manual.

A Series KEYEDIOII Programming Reference Manual (form 8600 0684)

This manual describes the KEYEDIOII software. KEYEDIOII is the Unisys indexed sequential access method (ISAM) software for COBOL74 and Report Program Generator (RPG) programming languages. This manual is designed for applications programmers and analysts, and others who are familiar with KEYEDIO.

A Series Mark 4.0 Software Release Capabilities Overview (form 8600 0015)

This overview summarizes the new features available with the Mark 4.0 software release, describes any features deimplemented in that release, and lists manuals that have been added or changed. The overview is written for system administrators, programmers, and others who will be preparing for and using the new software release.

A Series Operating System Installation Guide (form 8600 1021)

This guide describes how to use the UTILoader and Loader programs to install the operating system when a system is first initialized or when the operating system is not functioning. Information about changing your operating system or using Simple Installation are covered in other manuals. This guide is written for system administrators and operators who are responsible for installing the operating system.

About This Manual

A Series Systems Functional Overview (form 8600 0353)

This manual presents an overview of the A Series systems and serves as a central source of information for these systems. This overview is written for both new and experienced users of A Series systems, and for anyone wanting an introduction to these systems.

Contents

About This Manual	v
Section 1. CARDLINE Utility	
Printing a Data Deck	1-1
Punching a Data Deck	1-2
Examples Using the CARDLINE Utility	1-3
Section 2. COMPARE Utility	
Running the COMPARE Utility	2-1
Using MARC to Run COMPARE	2-1
Using a WFL Job to Run COMPARE	2-1
Files Used by the COMPARE Utility	2-3
Output from the COMPARE Utility	2-3
Section 3. DUMPALL Utility	
Understanding Key DUMPALL Concepts	3-1
Using DUMPALL Commands	3-2
Understanding File Records	3-2
Understanding Structural File Attributes	3-3
General Information about DUMPALL File	
Attributes	3-4
Attributes for Files with Fixed-Length Records . .	3-6
Attributes for Files with Variable-Length Records	3-7
Files with BLOCKSTRUCTURE =	
EXTERNAL	3-7
Files with BLOCKSTRUCTURE=VARIABLE,	
VARIABLE2, or VARIABLEOFFSET	3-8
Files with BLOCKSTRUCTURE=LINKED . .	3-8
Specifying Data or Character Set Translations	3-9
Standard Commands	3-12
ATTRIBUTES or FILE Command	3-13
CAT Command	3-14
COPY Command	3-21
DEFINE Command	3-38
DMPMT Command	3-39
HEXDSK Command	3-43
LIBMT Command	3-45
LIST Command	3-46
TEST Command	3-52
Interactive List Routine Commands	3-55
Interactive AGAIN Command	3-56

Contents

Interactive FILE or ATTRIBUTES Command	3-56
Interactive CONTINUE Command	3-56
Interactive LIST Command	3-57
Interactive MODE Command	3-59
Interactive NEXT Command	3-60
Interactive OPEN Command	3-60
Interactive PREVIOUS Command	3-61
Interactive PRINT Command	3-61
Interactive QUIT Command	3-62
Interactive RECORD Command	3-62
Interactive SKIP Command	3-63
Running the DUMPALL Utility	3-64
Parameter Mode	3-64
Card Mode	3-65
Interactive Mode	3-65
Using the MARC Interface	3-66
Controlling I/O Exceptions	3-67
Input to the DUMPALL Utility	3-70
Basic DUMPALL Constructs	3-70
Field Definition	3-73
Format Definition	3-77
Old Specs	3-78
Print Option	3-79
Record Range List	3-80
Skip Specification	3-82
Handling Tape Files	3-84
Description of Tape Formats	3-84
Tape Marks	3-84
Unlabeled Tapes	3-84
Labeled Tapes	3-85
Nonstandard Labeled Tapes	3-87
Output Files	3-87
Input Files from Labeled Tapes	3-88
Input Files from Unlabeled Tapes	3-88
Treating Labeled Tapes as Unlabeled Tapes	3-89

Section 4. FILECOPY Utility

Running the FILECOPY Utility	4-1
Input to the FILECOPY Utility	4-2
Basic FILECOPY Constructs	4-3
FILECOPY Task Requests	4-5
CREATED, ACCESSED, or UPDATED Request	4-5
ADDED or ALLFILES Request	4-6
EXPIRED Request	4-7
FILECOPY Modifiers	4-8
FILECOPY Options	4-12
Sample FILECOPY Runs	4-15
Index Files	4-18

Section 5. FILEDATA Utility

Basic FILEDATA Constructs	5-1
Output Options	5-4
Running the FILEDATA Utility	5-5
FILEDATA Parameter List	5-5
Selecting the Files to Be Reported On	5-6
Effects of Family Substitution	5-6
Error Reporting	5-6
Database Generation and Reuse	5-7
Sample FILEDATA Runs	5-8
Task Requests	5-10
ARCHIVEINFO Request	5-11
AREASUMMARY Request	5-13
ATTRIBUTES Request	5-14
BACKUP Request	5-17
CATALOGINFO Request	5-20
CHECKERBOARD Request	5-22
CODEFILEINFO Request	5-24
COMPATIBILITY Request	5-28
COPYDECK Request	5-30
DEFINEOUTPUT Request	5-31
FILENAMES Request	5-32
HEADERCONTENTS Request	5-35
INCOMPATIBILITY Request	5-37
NOREPORTS Request	5-39
STRUCTUREMAP Request	5-41
TAPEDIR Request	5-43
FILEDATA Modifiers	5-44
ABBREVIATED	5-44
ALL	5-44
ALTERDATE	5-44
ARCHIVE	5-44
ARCHIVEBACKUP	5-45
AREALENGTH	5-45
AREAS	5-45
AREASIZE	5-45
BACKUPSN	5-45
BLOCKSIZE	5-46
BLOCKSTRUCTURE	5-46
CATALOGUE	5-46
CCSVERSION	5-46
CODEVERSION	5-47
CREATIONDATE	5-48
CRUNCHED	5-48
CYCLE	5-48
DATABASE	5-48
DIRECTORY	5-48
DOCUMENTTYPE	5-49
EXTMODE	5-49
FAMILYNAME	5-49
FILEKIND	5-49

FILELENGTH	5-49
FILEORGANIZATION	5-49
FILESTRUCTURE	5-49
FILETYPE	5-50
FRAMESIZE	5-50
GUARDFILE	5-50
IDENTITY	5-50
INTMODE	5-50
LASTACCESSDATE	5-51
LASTRECORD	5-51
LEVEL	5-51
LICENSEKEY	5-51
LINEWIDTH	5-51
LOCKEDFILE	5-51
MAXRECSIZE	5-52
MINRECSIZE	5-52
NAMESONLY	5-52
NEWDATABASE	5-52
NONRESIDENTONLY	5-52
NOTE	5-52
PACKNAME	5-52
PAGESIZE	5-53
PERMITTEDACTIONS	5-53
RAWHEADERS	5-53
RELEASEID	5-53
RESIDENTONLY	5-53
SAVEFACTOR	5-53
SECURITY	5-53
TIMESTAMP	5-54
TITLE	5-54
TOTALSECTORS	5-54
UNITS	5-54
USERINFO	5-54
VERSION	5-54
WARNINGS	5-54
Numeric Report Requests	5-55
Old PACKDIR Syntax	5-56
Using System Commands to Initiate FILEDATA	5-58
DIR (Directory) Command	5-58
TDIR (Tape Directory) Command	5-59

Section 6. INTERACTIVEXREF Utility

INTERACTIVEXREF Operation	6-1
Files Used by the INTERACTIVEXREF Utility	6-1
Running the INTERACTIVEXREF Utility	6-2
Input to the INTERACTIVEXREF Utility	6-3
Basic INTERACTIVEXREF Constructs	6-3
Identifier Specification	6-3
Range Specification	6-7
INTERACTIVEXREF Commands	6-9

DECLARATIONS Command	6-10
EXPAND Command	6-17
HELP Command	6-20
LIST Command	6-21
LOAD Command	6-22
LOCATE Command	6-23
MERGE and COINCIDENCE Commands	6-24
QUALIFY Command	6-28
RANGE Command	6-28
REFERENCE Command	6-29
SET and RESET Commands	6-32
STOP Command	6-33
SUMMARY Command	6-33
SYMBOL Command	6-34
TERMINAL Command	6-35
WHAT Command	6-36
WHATFILES Command	6-36
Using the INTERACTIVEXREF Utility	6-37
Use with Improperly Sequenced Source	6-37
Use with COBOL74	6-37
Use with FORTRAN and FORTRAN77	6-38
Use with PASCAL	6-38
Example INTERACTIVEXREF Program	6-38

Section 7. KEYEDIO Support

Physical Structure of KEYEDIO Files	7-1
Coarse Tables	7-1
Fine Tables	7-2
Data Blocks	7-2
Locating Data	7-2
File and KEYEDIO Library Management	7-2
Removing and Installing a KEYEDIO Library	7-3
KEYEDIO Program Interface	7-3
Indexed KEYEDIO File Attributes	7-4
Setting the FILEORGANIZATION Attribute	7-5
Setting the EXCLUSIVE Attribute	7-5
Setting the Value of the BUFFERS Attribute	7-5
Impact of Number of Buffers on Processor Time	7-5
Impact of Number of Buffers on Save Memory	7-5
Rules for Determining the Number of Buffers	
Used	7-6
Choosing a Value for the BLOCKSIZE Attribute	7-7
Effect of Block Size on Processor Time	7-8
Effect of Block Size on Save Memory	7-8
Calculating Actual Block Size	7-8
Calculating User-Specified Block Size (2 Level)	7-9
Calculating Actual Area Size	7-10
KEYEDIO Procedures	7-10
Key Information	7-12
File Access Information	7-13

Results Returned	7-13
ISMGETKEYSTRUCTURE Procedure	7-14
ISMOPEN Procedure	7-15
ISMCLOSE Procedure	7-16
ISMSTART Procedure	7-17
ISMSEQUENTIALWRITE Procedure	7-18
ISMSEQUENTIALREAD Procedure	7-19
ISMRANDOMWRITE Procedure	7-20
ISMRANDOMREAD Procedure	7-20
ISMREWRITE Procedure	7-21
ISMDELETE Procedure	7-22
ISMSETUPLIMIT Procedure	7-23
The KEYEDIO File Structure	7-24
Segment 0 (Zero) of the File	7-24
Block Information Layout	7-28
Coarse Table Layout	7-29
Fine Table Layout	7-29
Key Information Table Layout	7-30
Logical Layout of a KEYEDIO File	7-30
Inserting Keys	7-32
Recovery Procedures	7-35
Recovery Messages and Warnings	7-35

Section 8. Mathematical Functions

Single-Precision Functions	8-1
ALGAMA Function	8-1
ALOG Function	8-2
ALOG10 Function	8-2
ARCOS Function	8-2
ARSIN Function	8-2
ATAN Function	8-3
ATAN2 Function	8-3
COS Function	8-3
COSH Function	8-4
COTAN Function	8-4
ERF Function	8-4
ERFC Function	8-5
EXP Function	8-5
Single-Precision Exponentiation	8-5
GAMMA Function	8-5
RANDOM Function	8-5
SIN Function	8-6
SINH Function	8-6
SQRT Function	8-7
TAN Function	8-7
TANH Function	8-7
Double-Precision Functions	8-7
DARCOS Function	8-7
DARSIN Function	8-8
DATAN Function	8-8

DATAN2 Function	8-8
DCOS Function	8-8
DCOSH Function	8-8
DERF Function	8-9
DERFC Function	8-9
DEXP Function	8-9
DGAMMA Function	8-9
DLGAMA Function	8-9
DLOG Function	8-10
DLOG10 Function	8-10
DSIN Function	8-10
DSINH Function	8-10
DSQRT Function	8-10
DTAN Function	8-10
DTANH Function	8-11
Double Precision Exponentiation	8-11
Complex Functions	8-11
Definitions Used in Complex Function Descriptions	8-11
CABS Function	8-12
CCOS Function	8-12
CEXP Function	8-13
CLOG Function	8-13
CSIN Function	8-14
CSQRT Function	8-14
Complex Exponentiation	8-15
Common Constants	8-15
Permissible Argument Ranges	8-16

Section 9. PATCH Utility

Running the PATCH Utility	9-1
Using a WFL Job to Run the PATCH Utility	9-1
Using CANDE to Run the PATCH Utility	9-2
Files Used by the PATCH Utility	9-3
Patch Control Records	9-3
Patch Compiler Control Records (\$ Records)	9-4
Patch Literal Compiler Records (\$& Records)	9-5
Patch Delimiter Records (\$# Records)	9-6
Patch Comment Records (\$: Records)	9-6
Patch Patch Records (\$- Records)	9-6
Patch WFL Records (\$* Records)	9-7
Patch Control Records (\$. Records)	9-8
Patch Control Record Options	9-10
\$.BRIEF Option	9-10
\$.COBOL Option	9-11
\$.COBOL74 Option	9-11
\$.COMPARE Option	9-11
\$.COMPILE Option	9-12
\$.CONFLICT Option	9-12
\$.COUNT Option	9-13
\$.CYCLE Option	9-13

\$.DELETE Option	9-13
\$.DELIMOPT Option	9-14
\$.DISK Option	9-15
\$.DISK \$ Option	9-16
\$.DUMP Option	9-16
\$.EOF Option	9-17
\$.ERRLIST Option	9-17
\$.EXECUTE Option	9-17
\$.FILE, \$.DISK \$, and \$.PATCHDECK Options	9-18
\$.FLAG Option	9-18
\$.GUARD Option	9-19
\$.INSERT Option	9-19
\$.LABEL Option	9-21
\$.LIST Option	9-22
\$.LISTD Option	9-22
\$.LISTI Option	9-22
\$.LISTN Option	9-23
\$.LISTP Option	9-23
\$.MARK Option	9-23
\$.MARKBLANK Option	9-24
\$.MOVE Option	9-25
\$.NDLII Option	9-26
\$.NEW Option	9-26
\$.OUT Option	9-26
\$.PASCAL Option	9-27
\$.PATCHDECK Option	9-27
\$.RPG Option	9-27
\$.SINGLE Option	9-27
\$.SQUASH Option	9-28
\$.TOTAL Option	9-28
\$.VERSION and \$.CYCLE Options	9-28
Debug \$. Records	9-29
Debug Options	9-30
\$.BUG Option	9-30
\$.CANDE Option	9-30
\$.DISCARD Option	9-31
\$.END Option	9-31
\$.EQUATE Option	9-32
\$.PDUMP Option	9-32
Examples of PATCH Utility Input	9-33

Section 10. PL/I Indexed Sequential-Access Method (PLIISAM)

Program Interface for Primitive and Standard ISAM	10-1
Primitive ISAM	10-1
Standard ISAM	10-2
Structure of ISAM Files	10-2
Prime Data Area	10-3
Data Overflow Area	10-3
Prime Data Area Overflow Space	10-3
File Overflow Area	10-3

Tables for Locating Data	10-3
Data Record Links	10-4
ISAM Management of Overflow Areas	10-4
Planning for ISAM Files	10-5
Maximum Number of Records	10-5
Coarse Table Size	10-5
Computing Coarse Table Size	10-5
Fine Table Size	10-5
Computing Fine Table Size	10-6
INFO Record Size	10-6
AREAS and AREASIZE Values	10-6
Minimum Record Size (MINRECSIZE)	10-7
Maximum Record Size (MAXRECSIZE)	10-7
BLOCKSIZE Attribute	10-7
EXCLUSIVE USE Attribute	10-7
Fine Table Ratio	10-7
Key Length	10-8
Key Offset	10-8
Practical Considerations	10-8
Implementation for Primitive ISAM Procedures	10-9
ISAM Procedures	10-9
ISOPEN Procedure	10-10
ISCLOSE Procedure	10-12
ISREAD Procedure	10-13
ISWRITE Procedure	10-14
ISREADNEXT Procedure	10-15
ISREWRITE Procedure	10-15
ISKEYWRITE Procedure	10-16
ISDELETE Procedure	10-17
ISAM I/O Result Information	10-17
Primitive ISAM Result Information	10-18

Section 11. RLTABLEGEN Utility

General Information	11-1
Installation Defined Tape Labels	11-1
Running the RLTABLEGEN Utility	11-2
Input to the RLTABLEGEN Utility	11-2
Label Description Format	11-2
RLTABLEGEN Commands	11-3
ID Command	11-3
RECOGNITION Command	11-4
FIELD Command	11-5
RECORD Command	11-6

Section 12. SORT Utility

SORT Parameters	12-2
Input Options	12-2
Output Options	12-3

Contents

Compare Procedure	12-3
Number of Tapes	12-4
Record Size	12-4
Memory Size	12-4
Determining Memory Size for Disk Sorting	12-5
Determining Memory Size for Tape Sorting	12-6
Determining Memory Size for Memory Sorting	12-6
Disk Size	12-6
SORT Operating Modes	12-7
Disk-Only Mode	12-8
Disk-Only Stringing Phase	12-8
Disk-Only Merging Phase	12-9
Tape-Only Mode	12-9
Tape-Only Stringing Phase	12-9
Tape-Only Merging Phase	12-10
ITD Mode	12-10
Memory-Only Mode	12-11
SORT Files	12-11
Control Files	12-11
Work Files	12-12
Tape Files	12-13
Tag Sorting	12-13
Restart Capability	12-15
RESTART Parameter Values	12-17
Restarting during Stringing Phase	12-19
Error Recovery	12-19
Control File Input Errors	12-20
Control File Output Errors	12-20
Work File Input Errors	12-20
Work File Output Errors	12-20
User Output File Errors	12-21
Work File Input Errors during User Output	12-21
Using SORT in Various Languages	12-21
Using SORT in COBOL or COBOL74	12-21
SORT Input/Output (I/O) Procedure Logic Flow	12-21
COBOL SORT Example	12-22
Using SORT in ALGOL	12-25
ALGOL SORT Example	12-25
ALGOL Example Files	12-26
Using SORT in PL/I	12-26
PL/I SORT Example	12-27
PL/I Example Files	12-28
Using a Procedural Interface for SORT	12-29
SORTFILES	12-29
SORTPROCS	12-32
MERGEFILES	12-33
MERGEPROCS	12-35
TTABLE	12-37
Sample Programs	12-39
SORT Error Messages	12-42
SORT Statistical Array	12-46

Section 13. XREFANALYZER Utility

XREFANALYZER Files	13-1
Invoking XREFANALYZER	13-2
Implicit Execution	13-2
Explicit Execution	13-2
Compile Time Options	13-3

Appendix A. Understanding Railroad Diagrams

What Are Railroad Diagrams?	A-1
Constants and Variables	A-2
Constraints	A-2
Following the Paths of a Railroad Diagram	A-5
Railroad Diagram Examples with Sample Input	A-6

Glossary	1
-----------------------	---

Bibliography	1
---------------------------	---

Index	1
--------------------	---

Figures

7-1.	Coarse Table Layout	7-29
7-2.	Fine Table Layout	7-30
7-3.	KEYEDIO File Layout	7-31
7-4.	Inserting a Key	7-33
7-5.	Inserting a Key into a Full Table	7-34
12-1.	Creating a Tag	12-14
12-2.	Tag Sort, Nondisk Input File	12-15
12-3.	Tag Sort, Disk Input File	12-15
A-1.	Railroad Constraints	A-5

Tables

2-1.	COMPARE Utility Files	2-3
3-1.	Standard DUMPALL Commands	3-12
3-2.	Default Field Type	3-74
5-1.	Code File Status Information	5-47
7-1.	Key Word Format	7-12
7-2.	File Access Values	7-13
8-1.	TAN/COTAN Calculation	8-4
8-2.	Common Constants	8-15
8-3.	Permissible Argument Ranges	8-16
8-4.	Function Names	8-17
12-1.	Determining SORT Operating Mode	12-7
12-2.	Fatal Error Messages	12-42
12-3.	Nonfatal Error Messages	12-45
12-4.	SORT Collating Sequence	12-51

Section 1

CARDLINE Utility

The SYSTEM/CARDLINE utility prints or punches an EBCDIC or BINARY data deck. The output includes a printout of the card images, a card count, and a sequence check. Columns 73 through 80 are checked for sequence errors.

In addition to the card-to-print function, you can perform other utility functions by file-equating the input or output files of the program. The input file is named CARD, and the output file is named LINE.

Printing a Data Deck

To list a card deck with EBCDIC data, use the following Work Flow Language (WFL) deck:

```
<i>BEGIN JOB CARDLINE;  
RUN SYSTEM/CARDLINE; VALUE = <integer>;  
EBCDIC  
.  
.  
<data deck>  
.  
.  
<i>END JOB
```

To list a card deck with BINARY data, use the following WFL deck:

```
<i>BEGIN JOB CARDLINE;  
RUN SYSTEM/CARDLINE; VALUE = <integer>;  
BINARY  
.  
.  
<binary data>  
.  
.  
BEND card  
<i>END JOB
```

To list a disk file, FILE CARD is file-equated to disk as follows:

```
<i>BEGIN JOB;  
RUN SYSTEM/CARDLINE; VALUE=<integer>;  
FILE CARD (KIND=DISK, TITLE="MYFILE");  
<i>END JOB
```

CARDLINE Utility

The `<i>` variable specifies an invalid EBCDIC character. When CARDLINE is run from an operator display terminal (ODT) or a remote terminal, the `<i>` variable is the question mark (?). When CARDLINE is run from a card reader, the `<i>` variable can be any invalid punch. An `<i>` variable is optional when CARDLINE is run from an ODT or a remote terminal; however, it is required when CARDLINE is run from a card reader.

The `VALUE = <integer>` clause specifies spacing between output lines. If this clause is not specified, 0 is the assumed value.

For EBCDIC files, the integer value must be within the range of 0 through 9. The values 0 and 1 cause single spacing (no blank lines between the lines of data). The values 2 through 9 cause the specified number of lines to be spaced; for example, if the value is 5, four blank lines are placed between the lines of data.

For BINARY files, the integer value must be within the range of 10 through 19. The first digit (the 1) indicates that the data is BINARY. The second digit is interpreted the same as for EBCDIC files: 0 and 1 cause single spacing; 2 through 9 cause the specified number of lines to be spaced.

The BEND card is a special card that indicates the end of the data and is only used with BINARY data. Refer to the "Job Initiation" section of the *A Series Work Flow Language (WFL) Programming Reference Manual* for information about creating BEND cards.

Punching a Data Deck

To punch a card deck, file-equate FILE LINE to a card punch.

To punch a card deck with EBCDIC data, use the following WFL deck:

```
<i>BEGIN JOB CARDLINE;  
RUN SYSTEM/CARDLINE;  
FILE LINE (KIND=PUNCH);  
EBCDIC  
.  
.  
<data deck>  
.  
.  
<i>END JOB
```

To punch a card deck with binary data, use the following WFL deck:

```

<i>BEGIN JOB CARDLINE;
RUN SYSTEM/CARDLINE;
FILE LINE (KIND=PUNCH);
BINARY CARD
.
.
<data deck>
.
.
BEND card
<i>END JOB
    
```

Examples Using the CARDLINE Utility

The following are examples of how to use the CARDLINE utility.

Example 1

The following WFL job initiates CARDLINE to produce a listing of the given EBCDIC data:

Input

```

?BEGIN JOB CARDLINE;
RUN SYSTEM/CARDLINE;
EBCDIC
THIS IS THE DATA TO BE LISTED.
MORE OF THE DATA.
?END JOB.
    
```

Output

```

THIS IS THE DATA TO BE LISTED.      0000100  CARD  0001
MORE OF THE DATA.                  0000200  CARD  0002
    
```

NO SEQUENCE ERRORS

CARDLINE Utility

Example 2

The following WFL job initiates CARDLINE to produce a listing of the disk file MYFILE:

Input

```
?BEGIN JOB CARDLINE;  
RUN SYSTEM/CARDLINE; VALUE=2;  
FILE CARD (KIND=DISK, TITLE=MYFILE);  
?END JOB.
```

Output

```
CONTENTS OF THE FILE.           00000100  CARD  0001  
MORE OF THE FILE.              00000200  CARD  0002  
END OF THE FILE.               00000300  CARD  0003
```

NO SEQUENCE ERRORS

Section 2

COMPARE Utility

The SYSTEM/COMPARE utility compares one or more pairs of files. This utility performs a bit-by-bit comparison on each record, or on each sequence number and record, for each pair of files. If the records or sequence numbers are not identical, or if one of the specified files is not present, an appropriate error message is printed. The comparison of a pair of files is terminated after a specified number of unsuccessful comparisons have been made, and the utility proceeds to the next pair of files.

Running the COMPARE Utility

The COMPARE utility can be initiated through the Menu-Assisted Resource Control (MARC) System Utilities screen or by means of a Work Flow Language (WFL) job.

Using MARC to Run COMPARE

On the System Utilities screen of the MARC interface, enter *COMP* in the Choice field. The SYSTEM/COMPARE Specification screen is displayed. For menu-specific or field-specific information, consult the help text associated with that menu or field.

To return to the MARC System Utilities screen, enter *BYE* in the Action field.

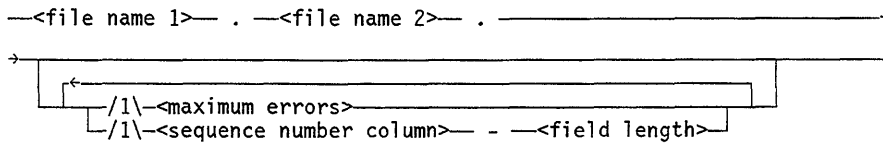
Using a WFL Job to Run COMPARE

The following WFL job initiates the SYSTEM/COMPARE utility:

```
<i>BEGIN JOB;  
RUN SYSTEM/COMPARE;  
DATA  
<compare input>  
<i>END JOB
```


COMPARE Utility

<compare input>



Explanation

<i>

Specifies an invalid character. When COMPARE is run from the operator display terminal (ODT) or a remote terminal, the <i> variable is the question mark (?). When COMPARE is run from the card reader, the <i> variable includes any invalid punch. An <i> variable is optional when COMPARE is run from the ODT or a remote terminal. However, it is required when COMPARE is run from the card reader.

<file name 1>.<file name 2>.

Specifies the two files to be compared. A period (.) must follow each file name. The system assumes that the files are disk files. If each file is not a disk file, they must first be label-equated.

EBCDIC, ASCII, and HEX disk files can be compared. Input specifications are in free-field format.

<maximum errors>

Specifies the number of unsuccessful comparisons that can be made before COMPARE proceeds to the next pair of files. The default value for the maximum errors field is 5.

<sequence number column> - <field length>

Specifies the column in which the sequence numbers of the files begin and the field length of the sequence numbers.

If no sequence information about the files is specified in the COMPARE input, the files are compared record by record. A new record is read from each file for each comparison. If a difference occurs, the record number at which the difference occurred is printed. If a difference occurs in sequenced files, both records are printed. If the sequence numbers agree but the records do not, the contents of both records are printed. If the record sequence number of the first file is greater than the record sequence number of the second file, the record from the second file is printed, and the next record from the second file is compared against the first record from the first file, and vice versa.

Example

The following example initiates COMPARE on three pairs of files. PROGRAM/ONE and PROGRAM/TWO are compared record by record until five unsuccessful

comparisons have been made. PROGRAM/THREE and PROGRAM/FOUR are compared until five unsuccessful comparisons have been made. The sequence information for these two files begins in column 73 for a length of 8. PROGRAM/FIVE and PROGRAM/SIX are compared record by record until 25 unsuccessful comparisons have been made. PROGRAM/SEVEN and PROGRAM/EIGHT are compared until four unsuccessful comparisons have been made. The sequence information for these two files begins in column 1 for a length of 6. PROGRAM/NINE and PROGRAM/TEN are also compared until four unsuccessful comparisons have been made. The sequence information for these two files begins in column 1 for a length of 6.

```
?BEGIN JOB;
RUN SYSTEM/COMPARE;
DATA
PROGRAM/ONE.    PROGRAM/TWO.
PROGRAM/THREE. PROGRAM/FOUR. 73-8
PROGRAM/FIVE.  PROGRAM/SIX. 25
PROGRAM/SEVEN. PROGRAM/EIGHT. 1-6 4
PROGRAM/NINE.  PROGRAM/TEN. 4 1-6
?END JOB
```

Files Used by the COMPARE Utility

Table 2-1 shows the files that are used by COMPARE and that can be file-equated:

Table 2-1. COMPARE Utility Files

File Name	Utility Use
LYNE	The output printer file used for the listing of differences.
TOT	The output printer file used for the summary report.
FILE1	The first file specified for the comparison.
FILE2	The second file specified for the comparison.
FILE3	The input file. This file contains the compare input parameters.

Output from the COMPARE Utility

The output listing contains the following information:

- A description of the two files being compared, which includes the MAXRECORDSIZE, BLOCKSIZE, UNITS, INTMODE, and CREATIONDATE attribute values. If the file is not in the directory, an error message is printed.
- If the files differ in blocking specifications (UNITS, BLOCKSIZE, MAXRECORDSIZE), no comparison is made and a message is printed.
- The maximum number of errors specified is listed.

COMPARE Utility

- If sequence information is specified in the COMPARE input, it is printed; otherwise, a message is printed stating that unsequenced files are assumed.
- If any differences occur, a list of the differences is printed.
- If the comparison of the files is terminated because the maximum number of errors has been reached, a message is printed along with the current record number of each file being compared.
- If an end-of-file (EOF) condition occurs in one file before the other, a message is printed. The message also indicates in which file the end-of-file (EOF) condition occurred.
- The number of differences is listed.
- The number of records in each file is given.
- A section is also printed that provides the information described previously, except for the list of differences.

Example

Input

```
?BEGIN JOB;  
RUN SYSTEM/COMPARE;  
DATA  
FILE/A. FILE/B. 73-8  
MYFILE/1. MYFILE/2.  
?END JOB
```

Output

TOTAL FOR ALL FILES COMPARED

 FILE 1 = FILE/A ON DISK.

MAXRECORDSIZE = 15
 BLOCKSIZE = 420
 UNITS = WORDS
 INTMODE = EBCDIC
 CREATION DATE = 5/27/83

FILE 2 = FILE/B ON DISK.

MAXRECORDSIZE = 15
 BLOCKSIZE = 420
 UNITS = WORDS
 INTMODE = EBCDIC
 CREATION DATE = 5/27/83

----MAXIMUM ERROR DEFAULT = 5
 ----SEQUENCED FILES,
 BEGINNING IN COL. 73 FOR A
 LENGTH OF 8

NUMBER OF DIFFERENCES = 3
 NUMBER OF RECORDS IN FILE 1 = 27
 NUMBER OF RECORDS IN FILE 2 = 27

FILE 1 = MYFILE/1 ON DISK.

MAXRECORDSIZE = 14
 BLOCKSIZE = 420
 UNITS = WORDS
 INTMODE = EBCDIC
 CREATION DATE = 5/27/83

FILE 2 = MYFILE/2 ON DISK.

MAXRECORDSIZE = 14
 BLOCKSIZE = 420
 UNITS = WORDS
 INTMODE = EBCDIC
 CREATION DATE = 11/17/82

---MAXIMUM ERROR DEFAULT = 5---
 ---UNSEQUENCED FILES ASSUMED---

NUMBER OF DIFFERENCES = 1
 NUMBER OF RECORDS IN FILE 1 = 2
 NUMBER OF RECORDS IN FILE 2 = 2

COMPARE Utility

FILE 1 = FILE/A ON DISK.

MAXRECORDSIZE = 15
BLOCKSIZE = 420
UNITS = WORDS
INTMODE = EBCDIC
CREATION DATE = 5/27/83

FILE 2 = FILE/B ON DISK.

MAXRECORDSIZE = 15
BLOCKSIZE = 420
UNITS = WORDS
INTMODE = EBCDIC
CREATION DATE = 5/27/83

-----MAXIMUM ERROR DEFAULT = 5
-----SEQUENCED FILES,
BEGINNING IN COL. 73 FOR A
LENGTH OF 8

*** LISTING OF DIFFERENCES ***

X := COUNTER - 1;	00001500 (ACCTDOC) FILE/A ON ACCOUNT.
Y := COUNTER -1;	00001500 (ACCTDOC) FILE/B ON ACCOUNT.
COUNTER := * + 2;	00004000 (ACCTDOC) FILE/A ON ACCOUNT.
COUNTER := * + 1;	00004000 (ACCTDOC) FILE/B ON ACCOUNT.
TOTAL := COUNTER;	00010000 (ACCTDOC) FILE/A ON ACCOUNT.
TOTAL := * + COUNTER;	00010000 (ACCTDOC) FILE/A ON ACCOUNT.

NUMBER OF DIFFERENCES = 3
NUMBER OF RECORDS IN FILE 1 = 27
NUMBER OF RECORDS IN FILE 2 = 27

FILE 1 = MYFILE/1 ON DISK.

MAXRECORDSIZE = 14
 BLOCKSIZE = 420
 UNITS = WORDS
 INTMODE = EBCDIC
 CREATION DATE = 5/27/83

FILE 2 = MYFILE/2 ON DISK.

MAXRECORDSIZE = 14
 BLOCKSIZE = 420
 UNITS = WORDS
 INTMODE = EBCDIC
 CREATION DATE = 11/17/82

---MAXIMUM ERROR DEFAULT = 5---
 ---UNSEQUENCED FILES ASSUMED---

*** LISTING OF DIFFERENCES ***

***** THE FILES DIFFER IN RECORD NUMBER 2

NUMBER OF DIFFERENCES	=	1
NUMBER OF RECORDS IN FILE 1	=	2
NUMBER OF RECORDS IN FILE 2	=	2

Section 3

DUMPALL Utility

You can use the SYSTEM/DUMPALL utility to print or copy various kinds of files such as disk files, card files, and labeled and unlabeled tape files. The following list provides an idea of what you can accomplish with the DUMPALL utility:

- You can copy a file from one kind of media, such as tape, to another kind of media, such as disk.
- When you copy a file, you can specify that the new copy have file attributes that are different from the original file.
- You can specify that only certain records in a file are to be copied or printed.
- You can copy records from several different input files to a single output file.
- You can add records from one or more input files to the end of an existing disk file.
- You can copy or print multivolume tape files, and you can copy or print files from multifile tape volumes and unlabeled tape volumes.
- You can create multivolume tape files, multifile tape volumes, and unlabeled tape volumes.
- You can copy or list files located on other host systems.

Understanding Key DUMPALL Concepts

An understanding of the following concepts is helpful in your use of DUMPALL.

- You can run DUMPALL in any of the following three modes:
 - Parameter mode
 - Card mode
 - Interactive mode

The mode you choose affects the way you enter the commands that you want DUMPALL to execute.

- DUMPALL prints or copies a file by reading records sequentially from the input file and writing them sequentially to the output file. To use DUMPALL, it is important that you understand what a record is.
- To read and write records, DUMPALL needs to know the organization of the file, the structure of the file, the size of the data records in the file, and the size of the blocks of records in the file. DUMPALL determines this information from the file attributes of each file and from any file attributes that you specify in the command.
- In some cases, DUMPALL translates the data in records from one character set, such as EBCDIC, to another, such as ASCII. The specifications in the commands that you use control the kind of data translations, if any, that DUMPALL performs.

Using DUMPALL Commands

To use DUMPALL, you give the utility commands such as CAT, COPY, or LIST to cause it to copy or print files. Refer to “Standard Commands” later in this section for explanations of all the DUMPALL commands.

When you run DUMPALL, you input the commands in one of the following ways:

- As a quoted parameter
- As card images
- Interactively
- Through Menu-Assisted Resource Control (MARC)

The method you follow for entering commands depends on which mode you choose. The method for entering commands for each mode is explained in “Running the DUMPALL Utility” later in this section.

Understanding File Records

A record or a record in a file is a group of data elements treated as an entity by READ and WRITE statements in programs. A block is a group of one or more records that the logical I/O subsystem transmits to and from I/O units. Files are often structured with multirecord blocks to increase the overall speed of I/O data transfers and to increase the amount of data that can be stored in a given amount of disk or tape space.

DUMPALL reads and writes data records from and to files one record at a time. DUMPALL uses the ordinary logical I/O subsystem of the A Series operating system to read and write files. DUMPALL does not simply read and write bulk masses of data as the library maintenance utility does.

For these reasons, DUMPALL needs to know the organization and structures of the input and output files, and the size of each data record and block of data records in the files. Some files, such as printer backup disk files, have special structures that the DUMPALL utility cannot copy properly. But most files on A Series systems are structured with file attributes that logical I/O supports. DUMPALL can correctly copy and print those kinds of files.

The keys to the success of printing and copying files with DUMPALL are as follows:

- Make sure that DUMPALL uses the correct values for MAXRECSIZE and BLOCKSIZE when it reads the input files.
- Make sure that data translation does not occur unless you want it to.
- In the case of an unlabeled tape input file or a standard labeled input file, use the SKIPTM option to correctly position the input tape to the area of the tape that contains the data records of the input file you want to copy or print.
- In the case of multifile input or output tape files, use the MULTIFILE option correctly.

DUMPALL might be able to process tape files written on other operating systems. Success depends on the following factors:

- The files must be structured in a manner that the A Series logical I/O subsystem supports.
- You must specify the proper values for record size and other file attributes in the commands that you give to DUMPALL.

Understanding Structural File Attributes

To copy or print a file properly, DUMPALL needs to know the following:

- The structure of the input file
- The size of the data records in the file
- The size of the blocks of records in the file

For ordinary disk files, labeled tape files, and card files, DUMPALL can automatically determine these requirements from the file attributes of the files. For such cases, DUMPALL can proceed to copy or print the files without requiring you to specify any file attributes—other than the file titles and kind of I/O units—in your DUMPALL commands.

However, there are other cases where you need to understand how certain file attributes affect DUMPALL so that you can specify the proper values in your DUMPALL commands. These cases include the following:

- You want DUMPALL to copy a file so that the output file has different attributes from the input file.
For this case, you must specify compatible file attributes for the output file, or DUMPALL cannot properly copy the file.
- You want to print or copy files from an unlabeled tape.
For unlabeled input tape files, you must specify the proper values for the file attributes, because DUMPALL has no way to determine them on its own.
- You want to print or copy files from a tape generated by another operating system.
The tape labels written by other systems might not always be compatible with A Series label standards. Therefore, logical I/O and DUMPALL might not be able to determine the correct file attributes of the files on such tapes.
For these cases, as for unlabeled tapes, you must specify the proper value for the file attributes, or DUMPALL cannot correctly process the file.
- You want to print or copy a disk file, but the data records were written by a program that used different file attributes from the program that initially declared and created the file.
For a case like this, specify in the DUMPALL commands the values of the file attributes used by the program that wrote the data records into the file, because the values that DUMPALL obtains automatically from the system might not work properly.

You might sometimes have a combination of these cases, such as when you want to copy a file from an unlabeled tape and give the output file—labeled or unlabeled—some different attribute values, such as a different BLOCKSIZE.

For a complete discussion of all file attributes, refer to the *A Series File Attributes Programming Reference Manual*.

General Information about DUMPALL File Attributes

The file attributes that DUMPALL uses for each input and output file have a large effect on whether DUMPALL prints or copies files correctly. The values for file attributes that DUMPALL uses when it processes a file are determined by the following criteria (in order of precedence):

1. The values you explicitly specify for file attributes in each DUMPALL command.

Note: *Your explicit values have the highest precedence, and they override values obtained from file labels, values that output files inherit from input files, and default values.*

2. For input files and existing output files, the values that the system obtains from file labels or headers.

Note: *The CAT command with a TO <file name> clause writes output to an existing output file instead of creating a new output file.*

3. For output files that DUMPALL creates, the values of the corresponding file attributes of the first input file.
4. The default values established by the system and DUMPALL.

For input files, DUMPALL sets the file attribute DEPENDENTSPECS equal to TRUE unless you perform one or more of the following actions:

- Explicitly set the DEPENDENTSPECS attribute equal to FALSE.
- Explicitly specify a nonzero value for the MAXRECSIZE or BLOCKSIZE file attribute.
Note: *If you set only one of these attributes, the other one is set to the same value by default.*
- Input an unlabeled tape file by using the UL or FR options, or set the file attribute KIND equal to TAPE and the file attribute LABEL equal to OMITTED or OMITTEDEOF.

Refer to the explanation of DEPENDENTSPECS in the *File Attributes Reference Manual* for an explanation of the other file attributes affected by DEPENDENTSPECS.

For each input and output file, DUMPALL prints a report of the file attributes it used for that file. You can use these reports to ensure that the correct attribute values were used. These reports can also aid you in figuring out why DUMPALL did not print or copy records correctly. For example, if the wrong BLOCKSIZE or MAXRECSIZE is used, the data in some of the records that are printed or copied might be missing or shifted.

In the syntax of DUMPALL commands—such as the CAT and COPY commands—you can specify file attributes for the input files and the output files. It is important to put your file attribute specifications in the correct place in the command. For example, if you want to copy a file from an unlabeled tape, make sure you specify the file attributes of the input file in the location of the CAT or COPY command that is reserved for specifying the attributes of the input file, rather than the location reserved for specifying the attributes of the output file.

Use of the following file attributes affect how DUMPALL reads and writes files:

- The structure of records in a file—whether the file has fixed length records or variable length records—and so forth, is determined by the BLOCKSTRUCTURE file attribute.
- The size of records is determined by the FRAMESIZE and MAXRECSIZE file attributes. In the case of files with variable length records, the size is limited by MINRECSIZE.
- The size of blocks is determined by the FRAMESIZE and BLOCKSIZE file attributes.
- The following attributes apply to disk files:
 - The method of accessing a file—whether it is a KEYEDIO file or a NOTRESTRICTED file—is determined by the FILEORGANIZATION file attribute.
 - How the file is placed on disk—whether or not it is a STREAM file—is determined by the FILESTRUCTURE attribute.

***Note:** When DUMPALL is copying to a disk file, the FILESTRUCTURE attribute is not inherited from the input file. The I/O subsystem uses the default value ALIGNED180 unless you explicitly specify otherwise.*

MAXRECSIZE and BLOCKSIZE are measured in FRAMESIZE units. FRAMESIZE can have the following values:

Value	Meaning
8	8 bits or one character or byte
48	48 bits or one word
4	4 bits or one digit

Thus, to say that a file has a record size of 70 is not precise unless you know what the value of the FRAMESIZE file attribute is. If FRAMESIZE is 8, then a record size of 70 means 70 bytes. But if FRAMESIZE is 48, then a record size of 70 means 70 words, or 420 bytes.

Programming note: *Tapes written by other operating systems are generally written with `FRAMESIZE=8`. Unfortunately, the default value for `FRAMESIZE` is 48 when `DUMPALL` is reading from a tape that you specify with the `UL` or `FR` options. So when you specify `UL` or `FR` to read a tape generated by an operating system other than an A Series system, you should explicitly specify `FRAMESIZE=8`.*

The way that `DUMPALL` handles `MAXRECSIZE` and `BLOCKSIZE` depends on whether the file has fixed length records or variable length records.

Attributes for Files with Fixed-Length Records

In a file with fixed-length records, all the records have the same length. The size of the records is specified by the value of the `MAXRECSIZE` file attribute. The size of blocks in a file with fixed-length records should be an integer multiple of the size of the records and is specified by the value of the `BLOCKSIZE` file attribute. For example, a file with fixed-length records with a `MAXRECSIZE` of 80 could have a `BLOCKSIZE` of 80, 160, or 240, and so fourth.

For a file with fixed-length records, the `BLOCKSTRUCTURE` file attribute has the value `FIXED`.

For a file with a `FILESTRUCTURE` of `STREAM`, use the `MAXRECSIZE` attribute rather than the `BLOCKSIZE` attribute.

By default, when you use the `CAT` command with the `GIVING` option, or when you use the `COPY` command, `DUMPALL` sets the attributes for the output file equal to those of the input file. You can also give the output file other attribute values as necessary:

- You can change the block size of the output file by specifying a new `BLOCKSIZE` value, but you should pick a size that is an integer multiple of the record size.
- You can change the record size of the output file by specifying a new value for the `MAXRECSIZE` attribute:
 - If the new value is larger than the size of the input records, `DUMPALL` pads the output records with trailing binary zeros.
 - If the new value is less than the size of the input records, `DUMPALL` truncates the trailing part of each input record when `DUMPALL` writes the record to the output file.
- When you change the value of the `MAXRECSIZE` attribute, you might also need to change the value of the `BLOCKSIZE` attribute so that it is an integer multiple of the new record size.

- You can change the value of the `FRAMESIZE` attribute of the output file. If you do this, you might also want to make corresponding changes in the `MAXRECSIZE` and `BLOCKSIZE` attributes. For example, if you change the `FRAMESIZE` value from 48 to 8 (from words to bytes) you should multiply both the `MAXRECSIZE` and `BLOCKSIZE` values by 6; otherwise the records DUMPALL writes to the output file contain only one-sixth of the data that the input records contain.

Attributes for Files with Variable-Length Records

You can use DUMPALL to copy and print most files with variable-length records and variable-length blocks. The structure of the variable-length blocks and records in file is determined by the `BLOCKSTRUCTURE` file attribute. For files with variable-length records, this attribute has one of the following values: `EXTERNAL`, `LINKED`, `VARIABLE`, `VARIABLEOFFSET`, or `VARIABLE2`. DUMPALL normally opens input files with `DEPENDENTSPECS=TRUE`. In general, when you want to print or copy a labeled A Series file with variable-length records, you should leave `DEPENDENTSPECS=TRUE`, and you should not specify any of the attributes that control variable-length size. It is easier and safer to let logical I/O automatically pick up the proper values for `BLOCKSTRUCTURE`, `BLOCKSIZE`, `MAXRECSIZE`, and `MINRECSIZE`. In addition, for files with `BLOCKSTRUCTURE=VARIABLEOFFSET`, let logical I/O pick up the values for `SIZEMODE`, `SIZEOFFSET`, and `SIZE2`. For unlabeled files and for files generated by operating systems other than the A Series operating system, DUMPALL is able to read the file correctly only if it has one of the structures that logical I/O supports, and if you specify the proper values for the attributes of the input file.

When DUMPALL copies or prints files with variable-length records, the utility proceeds as follows:

1. It issues read requests with lengths of the `MAXRECSIZE` attribute.
2. Logical I/O transfers the number of characters or words that the record has – up to the value of `MAXRECSIZE` – into the input buffer for DUMPALL.
3. DUMPALL checks to see how much data was transferred for the record.
4. DUMPALL then prints or writes that much data to the output file.

In the case of a `COPY` or `CAT` command, if the output file has a different `MAXRECSIZE` or `MINRECSIZE` value than the input file, some records might be truncated or padded when they are written to the output file – except for files with `BLOCKSTRUCTURE=EXTERNAL` or `BLOCKSTRUCTURE=LINKED`.

When you copy a file with variable-length records, you can specify new values for the `BLOCKSIZE` and `MAXRECSIZE` attributes of the output file when you want the new file to have larger or smaller blocks or records.

Files with `BLOCKSTRUCTURE = EXTERNAL`

For input files with `BLOCKSTRUCTURE=EXTERNAL`, if the value of the `MAXRECSIZE` attribute of the input file is equal to the value of the `BLOCKSIZE` attribute of the input file, DUMPALL can successfully copy and print the files.

If the `MAXRECSIZE` value is less than the `BLOCKSIZE` value, DUMPALL cannot determine the proper boundaries between the individual records in each block it reads. DUMPALL assumes that all the records in each block are `MAXRECSIZE` long, except that the last record in a block is shorter if the value of `MAXRECSIZE` is not an integer multiple of the `MAXRECSIZE` value. DUMPALL can copy these files from tape to tape, from disk to disk, and from tape to disk as long as the output files have the same values for `BLOCKSIZE` and `MAXRECSIZE` as the input files. But when you try to copy the files from disk to tape, to print the files, or to change the `BLOCKSIZE` or `MAXRECSIZE` of the output files, the results might not be entirely correct.

For an output file with a `BLOCKSTRUCTURE=EXTERNAL`, if the value of the `MAXRECSIZE` attribute is less than the length of some of the input records, DUMPALL proceeds as follows:

1. DUMPALL writes the first part of an oversized input record to one output record.
2. DUMPALL then writes the next part of the input record to the next output record, and so forth, until the oversized record is completely copied.

Files with `BLOCKSTRUCTURE=VARIABLE`, `VARIABLE2`, or `VARIABLEOFFSET`

DUMPALL can copy and print these files correctly – if you use the proper values for `BLOCKSIZE`, `MAXRECSIZE`, `INTMODE`, and `FRAMESIZE`; and in the case of `BLOCKSTRUCTURE=VARIABLEOFFSET`, if you use the proper values for `SIZEMODE`, `SIZEOFFSET`, and `SIZE2`. When copying these files, you can specify new values for `BLOCKSIZE` and `MAXRECSIZE` for the output file, and you can even specify that the output file have a different `BLOCKSTRUCTURE` value, such as `FIXED` or `EXTERNAL`.

Files with `BLOCKSTRUCTURE=LINKED`

This kind of file is usually created by FORTRAN programs that execute `WRITE` statements without `FORMAT` statements and by ALGOL programs that execute binary `WRITE` statements – that is, `WRITE` statements with an asterisk (*) for the data count. `LINKED` files do not use the concept of record size as described by the `MAXRECSIZE` and `MINRECSIZE` attributes. In many cases DUMPALL cannot print or copy this kind of file correctly.

The information that follows briefly explains the internal structure of files with `BLOCKSTRUCTURE=LINKED` and under what circumstances DUMPALL can successfully print and copy these files.

- When a program writes data to a LINKED file, the logical I/O subsystem puts special control words into each block before it writes the block to the file. One of the features of LINKED files is that the READ and WRITE statements in a program are not limited by the MAXRECSIZE or BLOCKSIZE attributes. A single WRITE statement in a program can transfer more data than can fit into a single block of the file, as determined by the BLOCKSIZE value. Logical I/O uses the control words to link together the data transferred by one WRITE statement into as many blocks as necessary. Subsequently, when a program tries to read that mass of data, logical I/O checks the control words and assembles data from all the blocks necessary. In summary, from the point of view of the program, each READ or WRITE statement transfers one record, but logical I/O might have to gather data from or distribute data to more than one block in the file.
- When DUMPALL reads from a file, whether the file is LINKED or not, DUMPALL executes a READ statement with a length equal to the value of MAXRECSIZE. If the program that wrote the file used WRITE statements that transferred more data than the value of MAXRECSIZE, DUMPALL does not receive all the data when it reads such oversized, linked records. Thus, DUMPALL does not print or copy all the data in such records.
- DUMPALL can correctly print all records in a LINKED file that are not longer than the value of MAXRECSIZE. DUMPALL prints only the first part of records that are longer than the MAXRECSIZE value of the input file.
- DUMPALL can correctly copy all records in a LINKED file that are not longer than the value of MAXRECSIZE. DUMPALL copies only the first part of records that are longer than the MAXRECSIZE of the input file.
- When you want either to add records to a LINKED file (with the CAT command) or to create a new LINKED file, DUMPALL correctly copies all the data that it reads from the input file to the output file even if the MAXRECSIZE or BLOCKSIZE values of the output file are smaller than the sizes of the input records.

Specifying Data or Character Set Translations

Normally DUMPALL does not perform data or character translation when it copies a file while it executes a COPY or a CAT command. The only translations that DUMPALL normally performs when it executes a LIST command are those required by formats or fields you specify and those required by the default print options or the print options you specify.

DUMPALL can perform data or character-set translations if you explicitly specify values for the INTMODE file attribute, the EXTMODE file attribute, or both.

DUMPALL Utility

Character set translations can occur in three places:

- When DUMPALL is reading the input file, logical I/O can perform data translation if the INTMODE and EXTMODE attributes of the input file are not equal to each other.
- When DUMPALL is copying data from the input buffer to the output buffer, DUMPALL can perform data translation if the INTMODE value of the input file is not equal to the INTMODE value of the output file.
- When DUMPALL is writing to the output file, logical I/O can perform data translation if the INTMODE and EXTMODE attributes of the output file are not equal to each other.

You must be careful whenever you specify values for INTMODE or EXTMODE for the input file or the output file. If the records in the input file contain nontext data, such as decimal or integer numbers (COMP), real or floating point numbers (COMP-2), or other binary or numerical information, you probably should not let DUMPALL perform any data translations. If DUMPALL or logical I/O translate numerical fields in the input records, that data might not be usable or correct in the output records.

By default, DUMPALL sets the INTMODE of the input file, the INTMODE of the output file, and the EXTMODE of the output file all equal to the EXTMODE value of the input file. So, by default, DUMPALL does not perform data translations.

You can invoke and control data translations by using the following file attributes:

File Attribute	Function
TRANSLATION	By default, DUMPALL sets TRANSLATE to FULLTRANS for both the input file and the output file. If the INTMODE and the EXTMODE attributes of an input or output file differ, logical I/O can translate the data in the records.
INTMODE	<p>INTMODE is not a permanent file attribute that is saved in the label information for the file. Logical I/O uses the INTMODE values set by you or by DUMPALL as one of the factors in determining whether it should translate the data in the records.</p> <p>By default, DUMPALL sets the INTMODE value of the input file equal to the EXTMODE value of the input file; by default, DUMPALL sets the INTMODE value of the output file equal to the INTMODE value of the input file.</p> <p>For a COPY or CAT command, DUMPALL compares the INTMODE value of the input file with the INTMODE value of the output file. If the values differ, DUMPALL can translate the data in each record when it copies the data from the input buffer to the output buffer.</p>

continued

continued

File Attribute	Function
EXTMODE	<p>EXTMODE is a permanent file attribute that is stored in the label information of labeled files. Logical I/O compares the EXTMODE value of a file with the INTMODE value set by DUMPALL. If the values differ, logical I/O can perform data translations when DUMPALL reads from or writes to that file.</p> <p>DUMPALL uses any value you specify for the EXTMODE attribute of an input file only if the input file is unlabeled.</p> <p>By default, DUMPALL sets the EXTMODE value of the output file equal to the EXTMODE value of the input file.</p>
CCSVERSION	<p>The CCSVERSION file attribute is stored in the label information for disk files only. The CCSVERSION attribute does not affect data translations directly. But you should know that when DUMPALL copies from an input disk file to an output disk file, DUMPALL sets the CCSVERSION value of the output file equal to the value of the input file unless you do one of the following:</p> <ul style="list-style-type: none"> ● Explicitly specify CCSVERSION for the output file. ● Explicitly specify the INTMODE or EXTMODE value of the output file.

The previous descriptions explain that when modes differ, data translations can occur. For each of the three possible translations, whether or not a translation takes place depends on the exact value of the two modes involved and is determined as follows:

- If the two modes are the same, no translations occur.
- If one mode is SINGLE (word) and the other mode is HEX, EBCDIC, or ASCII, no translations occur.
- If one mode is HEX, EBCDIC, or ASCII, and the other mode differs but is also either HEX, EBCDIC, or ASCII, translations occur.
- Otherwise, the translation that occurs is determined by a procedure in the CENTRALSUPPORT library called CCSTOCCS_TRANS_TABLE. If a transliteration table is available for converting characters between the two coded character sets specified by the two modes, translation occurs; if no transliteration table exists, DUMPALL reports an error. For more information, refer to the *A Series MultiLingual System (MLS) Administration, Operations, and Programming Guide*.

Standard Commands

Table 3-1 lists all the commands you can use when you run DUMPALL. You can use these commands in any mode-parameter mode, card mode, or interactive mode-unless otherwise noted.

Table 3-1. Standard DUMPALL Commands

Command	Function
ATTRIBUTES or FILE	Prints or displays the file attributes of disk or tape files.
CAT	Creates a new file that contains records read from one or more input files or adds records read from one or more input files to the end of an existing disk file.
COPY	Creates a new file that contains records read from one or more input files. Creates multifile tapes with records copied from input files. Copies files from multifile input tapes.
DEFINE	Enables you to specify fields and formats that you can reference in subsequent print commands, such as DMPMT, LIST, and TEST commands.
DMPMT	Prints or displays the contents of blocks read from an input tape volume. DMPMT is useful as an aid to help you determine the contents and structure of files on a tape.
HEXDSK	Prints or displays the contents of the sectors of a disk file in hexadecimal format.
LIBMT	Prints an EBCDIC listing of the tape directory and a hexadecimal listing of the disk file headers and disk file data read from an input library maintenance tape.
LIST	Prints or displays the contents of records read from an input file.
TEST	Reads a file to check for parity errors. Prints or displays defective data contained in records that have parity errors.

In Table 3-1, the phrase *prints or displays* indicates that in card mode and parameter mode DUMPALL prints the information on a printer. In interactive mode, DUMPALL displays the information on the remote terminal unless you include a PRINT modifier in the command.

ATTRIBUTES or FILE Command

The ATTRIBUTES or FILE command lists the file attributes of a file or of all the disk files in a disk directory. ATTRIBUTES and FILE are synonyms.

Syntax

```
ATTRIBUTES <file title> [ <directory title> ] [ PRINT ]  
FILE
```

Explanation

If the file title does not include an ON <family name> clause, DUMPALL searches for one of the following matching files:

- For a disk file with a matching file name
- For a tape file with a matching file name
- For a card file with a matching file name

You can specify the file that is to be a tape file by using the family name TAPE as follows:

```
<file name> ON TAPE
```

The <directory title> case applies only to disk files.

PRINT

Used in interactive mode to list file attributes of the specified file or files on the printer instead of on the remote terminal.

Example

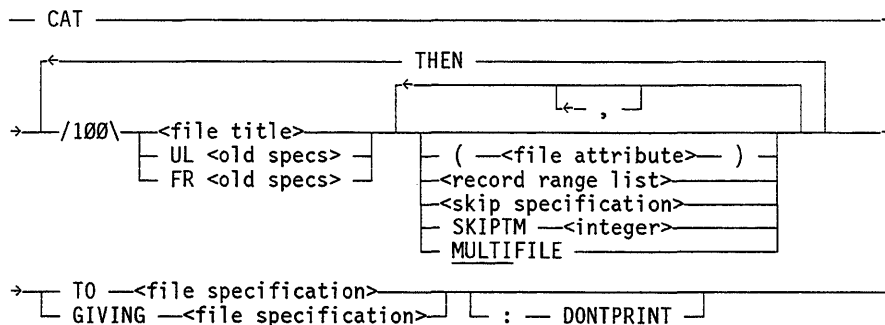
The following example lists the file attributes of the file IN/FILE:

```
FILE IN/FILE;
```

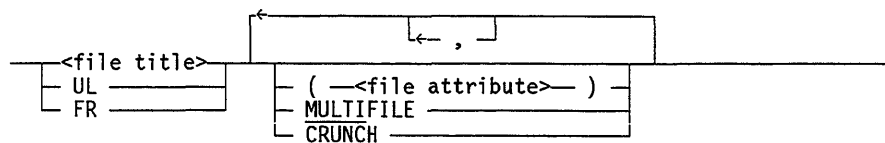
CAT Command

The CAT command extends an existing disk file by copying records to the end of that file from one or more input files. You can also use the CAT command to create a new file with records copied from one or more input files.

Syntax



<file specification>



Explanation

CAT <file title>
 THEN <file title>

Specify the name of an input file. If the file title does not include an ON <family name> clause, DUMPALL searches for a disk file with a matching file name, for a tape file with a matching file name, and for a card file with a matching name. You can specify the file to be a tape file by using the family name TAPE. That is, for a tape file title, specify the following:

<file name> ON TAPE

For each subsequent input file that you specify, DUMPALL assumes that the input file has the same file attributes as the preceding input file unless you specify otherwise. As an example, assume that you have entered the following command:

CAT F1/X ON HLPACK THEN Q/FILE GIVING NEWFILE

DUMPALL searches for the file F1/X on the disk family HLPACK. Later, it also searches for the input file Q/FILE on HLPACK. Because you specified no special attributes for the output file, DUMPALL also creates the new output file on the family HLPACK.

GIVING <file title>

Specifies the name of the new output file that DUMPALL is to create. Unless you use a (<file attribute>) clause for the output file, the output files attributes, which include KIND, MAXRECSIZE, FILEKIND, and FAMILYNAME, become the same as those of the first input file. If the first input file is a card file—(KIND=READER)—you must specify a KIND for the output file either by including an ON <family name> clause in the file title or by specifying KIND explicitly as a file attribute. For example, DUMPALL produces an output card file if you specify KIND=PUNCH as follows:

```
... GIVING C (KIND=PUNCH);
```

TO <file title>

Specifies the name of an existing output file to which DUMPALL is to add the records read from the input file or files. You can use this form of the command only with output disk files. Moreover, the file cannot be a crunched file or a code file because records cannot be added to the end of a crunched file, and only a compiler can write records into a codefile.

UL**UL <old specs>**

Specify that an input or output file has a KIND attribute of TAPE and has no label. You can use UL with a labeled tape; UL merely specifies that the tape be treated as unlabeled.

Specify UL only for magnetic tape files. When you use UL, it is not necessary to specify a KIND attribute value of TAPE. UL is ignored for disk and pack files.

The default INTMODE file attribute value is EBCDIC, and the default MAXRECSIZE and BLOCKSIZE attribute values are 1500. You can specify other values for these attributes with the <old specs> variable or with the (<file attribute>) syntax.

Programming note: *The default FRAMESIZE is 48 (that is, words). If you want to specify the MAXRECSIZE and BLOCKSIZE attributes in bytes, you must include FRAMESIZE=8 as a file attribute specification.*

When you use UL in an output file specification, the system creates an unlabeled tape as described in “Unlabeled Tapes” later in this section. The operating system demands as many tapes as are required to hold the data being copied to the tape. UL and FR perform the same functions for output files.

When you use UL for an input file whose options do not include SKIPTM or MULTI, DUMPALL can read more than one physical tape. In this case, when DUMPALL reads a tape mark from the tape, it closes that tape volume. DUMPALL then attempts to open the next input tape volume. This action causes the system to generate the following RSVP message:

```
NO FILE UL (UNLABELED MT) #nn
```

DUMPALL Utility

You must reply to this message with either the UL (Unlabeled) or the FR (Final Reel) system command.

When you use UL in an input file specification that includes SKIPTM or MULTI, UL is equivalent to FR—that is, DUMPALL uses only one physical tape. In the latter case, when DUMPALL reads a tape mark from the tape, DUMPALL closes the file and does not generate a “NO FILE” RSVP message. If you need to copy a file from a multivolume unlabeled set of tapes, and if you need to use the SKIPTM or MULTI options, use the THEN clause of the CAT command to copy the various parts of the file from the separate tape volumes. For example, suppose the fifth file on the first tape was split into two parts by a volume-switch operation that occurred while the file was being copied to the tapes. Then you could copy the entire file from the tapes by using a command of the following form:

```
CAT UL SKIPTM 4 (<file attribute>)  
THEN UL (<file attribute>) TO <file title>
```

FR
FR <old specs>

For input files, specify an unlabeled tape file that has only one reel. As in the case of UL, the tape used need not be an unlabeled tape.

Specify FR only for magnetic tape files. When you use FR, it is not necessary to specify a KIND attribute value of TAPE. DUMPALL ignores FR for disk and pack files.

The default INTMODE file attribute value is EBCDIC, and the default MAXRECSIZE and BLOCKSIZE attribute values are 1500. You can specify other values for these attributes with old specs or with the (<file attribute>) syntax.

Programming note: *The default FRAMESIZE is 48 (that is, words). If you want to specify the MAXRECSIZE and BLOCKSIZE attributes in bytes, you must include FRAMESIZE=8 as a file attribute specification.*

When used for an output file specification, FR specifies that the output tape is to be unlabeled and formatted as described under “Unlabeled Tapes” later in this section. The operating system uses as many tapes as are required to hold the data to be copied.

The use of FR instead of UL is important for input files only. That is, when you use FR in an input file specification, it indicates that only one physical tape is to be used. When DUMPALL reads a tape mark from the tape, it closes the file and does not generate a “NO FILE” RSVP message. Refer to “Input Files from Unlabeled Tapes” later in this section.

(<file attribute>)

Specifies how the file is to be written or read. Except for the file attributes you specify, DUMPALL obtains the file attribute information for each input file from the disk file header or the tape label. Except for the file attributes you specify and the FILESTRUCTURE, SERIALNO and HOSTNAME attributes, DUMPALL obtains the

file attribute information for each output file from the first input file copied to that output file.

DUMPALL determines the value of the KIND attribute for input files and output files by one of the following processes:

- If you specified either UL or FR, then KIND is TAPE.
- If you specified a file title with an explicit ON family name clause, then KIND is TAPE or DISK depending on whether or not you specified the family name ON TAPE.
- If you explicitly specify a value for KIND in the (<file attribute>) syntax, then DUMPALL uses that value.
- Otherwise DUMPALL determines the value for KIND automatically as follows:
 - For the first input file in the command, DUMPALL searches for the file on disk, tape, and cards.
 - For all subsequent input files in the command for which you have not explicitly or implicitly specified a value for KIND, DUMPALL uses the KIND value from the preceding input file.
 - For the output file in the command, DUMPALL uses the KIND value from the first input file in the command.

If you do not specify a value for SAVEFACTOR for the output file, DUMPALL assigns the output file the same SAVEFACTOR value as the first input file. If that value is 0, DUMPALL uses 2 instead.

Programming Note: *A tape file with a SAVEFACTOR of zero expires the day it is created. The system treats tapes containing expired tape files as scratch tapes unless you remove the write ring from the reel or switch the write-protection knob on the tape cartridge.*

If you specify the HOSTNAME file attribute, DUMPALL accesses the file on a remote host system. In this case, distributed system services (DSS) requires that you also explicitly specify the KIND file attribute.

The file attribute equation specifies how the file is to be written or read. If no file attributes are specified for the input file, DUMPALL opens the input file with the following logic:

- Labeled tape files are opened with DEPENDENTSPECS set to TRUE, INTMODE set to EBCDIC, and EXTMODE set to the physical mode of the file.
- Disk files are opened with DEPENDENTSPECS set to TRUE and with INTMODE and EXTMODE set to the physical mode of the input file.

The logical input file resembles the physical input file for disk files, and resembles the physical file for tape files if the tape file has a physical mode of EBCDIC.

The record and block structures of the output file are the same as those of the first input file if you do not specify any file attributes for either the input file or the output

file, and if the input file is either a tape file with a physical mode of EBCDIC or a disk file. In this case, no data translations occur, so DUMPALL does not alter the records it copies from the input file or files to the output file.

Specifying input file attributes such as DEPENDENTSPECS, EXTMODE, and INTMODE might cause data translation and the alteration of other file attributes (for example, MAXRECSIZE). Specification of such attributes might also avoid data translation if the input file is not an EBCDIC tape file. If data translation does occur, the output file might not directly resemble the input file.

Refer to “Understanding Structural File Attributes” earlier in this section for an explanation of the file structure, record size, block size, the DEPENDENTSPECS attribute, and their effects on DUMPALL.

Refer to “Specifying Data or Character Set Translations” earlier in this section for an explanation of the effects of the EXTMODE and INTMODE file attributes on DUMPALL.

<record range list>
<skip specification>

Cause only the specified records of the file to be copied. If you do not specify a record range list or skip specification, DUMPALL copies the entire file.

SKIPTM <integer>

Causes DUMPALL to skip past the number of tape marks specified and to bypass any records encountered between tape marks. Use SKIPTM to position an input tape at the first record of a file to be read. You can use SKIPTM only with an input file that you specify as UL or FR.

Refer to “Description of Tape Formats” later in this section for an explanation of where tape marks appear on a tape.

MULTIFILE
MULTI

Specify that an input file is part of a multiframe, labeled or unlabeled tape. The effect of its use is that the tape is not rewound, but is positioned so that a subsequent file on the same volume can be read from. In other words, MULTIFILE copies one or more subsequent files from the same tape. DUMPALL ignores the MULTIFILE specification if the file is not a tape file. Use MULTIFILE only if you are using a single command to read multiple files.

You must use MULTIFILE to copy a file from a multiframe tape if the following conditions are met:

- The tape is a standard labeled tape.
- The file is not the first file on the tape.

- The file has a nonstandard name. Normally, a tape file has a two-level name of the following form: volumeid/fileid. A nonstandard tape file name is a name with only one level, such as F1 or MYFILE.

In such a case, when DUMPALL tries to open the file, the system produces a “NO FILE <filename> (MT)” RSVP message. You must reply with the IL (Ignore Label) system command to select the tape unit on which the tape with the requested file is mounted.

Normally, when DUMPALL finishes copying a file to a tape, it closes the tape with the LOCK option. So, depending on the kind of tape unit involved, the system either unloads the tape volume or marks the unit as unloaded. However, if you specify MULTIFILE on the output file that you want copied to tape, DUMPALL does not close the tape with the LOCK option, and the tape simply rewinds. When the rewinding finishes, the system leaves the tape online and ready for use in subsequent DUMPALL commands or subsequent programs.

CRUNCH

When the output disk or pack file is closed, DUMPALL returns the unused portion of the last row of disk space to the system. DUMPALL ignores the CRUNCH option if the file is not a disk or pack file.

DONTPRINT

Suppresses the printed report of the file attributes of the input files and the output file.

Example 1

Example 1 concatenates file A to existing file B:

```
CAT A TO B;
```

Example 2

Example 2 concatenates the files A, B and C to existing file D:

```
CAT A THEN B THEN C TO D;
```

Example 3

Example 3 concatenates the files A and B to create a new file named C. If a file named C already exists, DUMPALL replaces the existing file. The new file named C has the same file attribute values as file A.

```
CAT A THEN B GIVING C;
```

DUMPALL Utility

Example 4

Example 4 concatenates the files A, B, and C to an existing file D. DUMPALL suppresses printed output.

```
CAT A THEN B THEN C TO D : DONTPRINT;
```

Considerations for Use

If you use the CAT command to copy an input code file, the resulting output file has a FILEKIND equal to DATA.

Programming note: *A CAT command with a GIVING clause executes just like a COPY command.*

You cannot use DUMPALL to copy files with a FILEORGANIZATION attribute value equal to PLIISAM, KEYEDIOII, or KEYEDIOISET. To copy KEYEDIOII and KEYEDIOISET files, use the KEYEDIOII Copy Utility as described in the *A Series KEYEDIOII Programming Reference Manual*.

You can use DUMPALL to copy an input file with a FILEORGANIZATION value equal to INDEXED or INDEXEDNOTRESTRICTED. DUMPALL does not copy any records that have been marked as deleted from these files. You can copy these files to output files that are NOTRESTRICTED, INDEXED, or INDEXEDNOTRESTRICTED with the following stipulations:

- You can add records copied from these files to existing disk files that have a FILEORGANIZATION value equal to NOTRESTRICTED.
- You can copy these input files to create a new output file with a FILEORGANIZATION value of INDEXED or INDEXEDNOTRESTRICTED if the system option KEYEDIOII is reset to FALSE.
- You can copy these input files to create a new output file that is not an INDEXED file by specifying that the output file has a FILEORGANIZATION value of NOTRESTRICTED.

An attempt to add records to an existing PLIISAM, INDEXED, or KEYEDIOII file by means of the CAT command results in the following error message:

```
CANNOT CAT TO A FILE THAT DOES NOT HAVE A FILEORGANIZATION =  
NOTRESTRICTED.
```

If the KEYEDIOSUPPORT library is not defined as a support library by means of the SL (Support Library) system command, DUMPALL terminates with the following message when DUMPALL accesses an INDEXED file:

```
FILE <filename> OPEN ERROR: SUPPORT LIBRARY UNAVAILABLE.
```

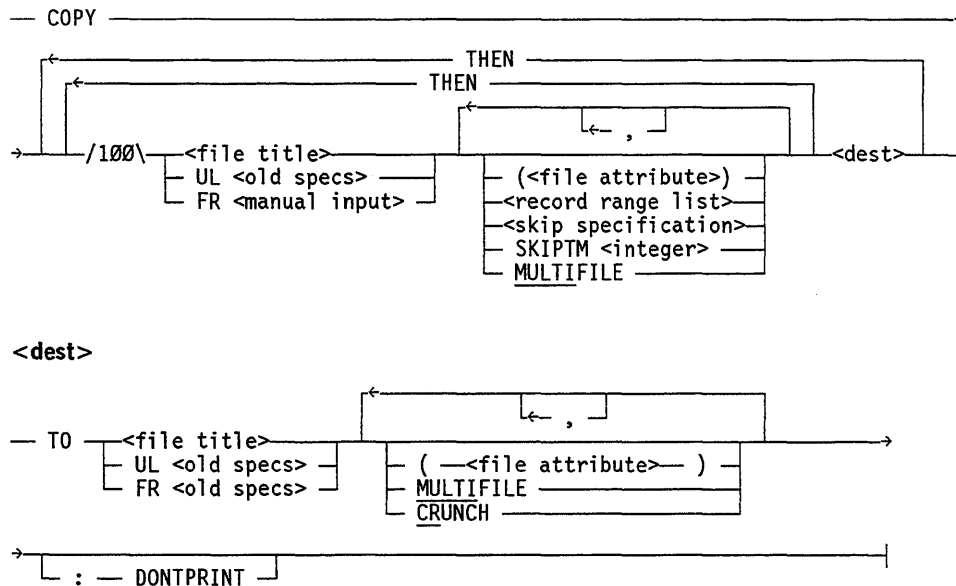
The rules regarding input and output file specifications for labeled and unlabeled tapes are the same as those for the COPY command with one exception. The variation of the CAT command TO <destination file> that causes the input file to be appended to an existing file is not allowed when the output file is a tape file.

COPY Command

The COPY command copies the specified records from one or more input files to an output file. The files can reside on different devices or on multifile tapes.

You can use a single copy command to create several different output files. Each output file is created with copies of records from the input files you specify for it. This multiple output file form of the COPY command is almost equivalent to executing several individual COPY commands in sequence. However, you must use the multiple output file form of the COPY command if you want to create a multifile output tape. No more than 100 input and output files can be specified in one COPY command.

Syntax



Explanation

COPY <file title>

Specifies the first or only input file from which DUMPALL is to copy records.

THEN <file title>

Specifies the name of a subsequent input file from which DUMPALL is to copy records. If you do not specify the KIND or FAMILYNAME (in the case of disk files) for each input file, DUMPALL uses the same KIND and FAMILYNAME as that of the previous input file. Suppose you enter the following command:

```
COPY F/A ON XPACK THEN (UC)ONE TO TFILE
```

DUMPALL searches for the input file (UC)ONE on the disk family XPACK. DUMPALL creates the new output file TFILE on XPACK as well.

DUMPALL Utility

TO <file title>

Specifies the name of the output file that DUMPALL is to create and to which DUMPALL is to copy the records from the preceding input files. If you do not specify file attributes such as **FRAMESIZE**, **MAXRECSIZE**, **BLOCKSIZE**, **KIND**, **FILEKIND**, and **FAMILYNAME** for the output file, DUMPALL uses the attribute values of the first input file that you specified to be copied to the file.

UL

UL <old specs>

Specify that an input or output file has a **KIND** attribute of **TAPE** and has no label. You can use **UL** with a labeled tape; **UL** merely specifies that the tape be treated as unlabeled.

Specify **UL** only for magnetic tape files. When you use **UL**, it is not necessary to specify a **KIND** attribute value of **TAPE**. **UL** is ignored for disk and pack files.

The default **INTMODE** file attribute value is **EBCDIC**, and the default **MAXRECSIZE** and **BLOCKSIZE** attribute values are 1500. You can specify other values for these attributes with the <old specs> variable or with the (<file attribute>) syntax.

Programming note: *The default **FRAMESIZE** is 48 (that is, words). If you want to specify the **MAXRECSIZE** and **BLOCKSIZE** attributes in bytes, you must include **FRAMESIZE=8** as a file attribute specification.*

When you use **UL** in an output file specification, the system creates an unlabeled tape as described in “Unlabeled Tapes” later in this section. The operating system demands as many tapes as are required to hold the data being copied to the tape. **UL** and **FR** perform the same functions for output files.

When you use **UL** for an input file whose options do not include **SKIPTM** or **MULTI**, DUMPALL can read more than one physical tape. When DUMPALL reads a tape mark from the tape, it closes that volume and attempts to open the next volume. This action causes the system to generate the following **RSVP** message:

```
NO FILE UL (UNLABELED MT) #nn
```

You must reply to this message with either the **UL** (Unlabeled) or the **FR** (Final Reel) system command.

When you use **UL** in an input file specification that includes **SKIPTM** or **MULTI**, **UL** is equivalent to **FR**—that is, DUMPALL uses only one physical tape. In the latter case, when DUMPALL reads a tape mark from the tape, DUMPALL does not try to proceed to the next volume.

If you need to copy a file from a multivolume unlabeled set of tapes, and if you need to use the **SKIPTM** or **MULTI** option, use the **THEN** clause of the **COPY** command to copy the various parts of the file from the separate tape volumes. For example, suppose the fifth file on the first tape was split into two parts by a volume switch operation that

occurred while the file was being copied to the tapes. Then you could copy the entire file from the tapes by using a command of the following form:

```
COPY UL SKIPTM 4 (<file attribute>)  
THEN UL (<file attribute>) TO <file title>
```

FR
FR <old specs>

For input files, specify an unlabeled tape file that has only one reel. As in the case of UL, the tape used need not be an unlabeled tape.

Specify FR only for magnetic tape files. When you use FR, it is not necessary to specify a KIND attribute value of TAPE. DUMPALL ignores FR for disk and pack files.

The default INTMODE file attribute value is EBCDIC, and the default MAXRECSIZE and BLOCKSIZE attribute values are 1500. You can specify other values for these attributes with old specs or with the (<file attribute>) syntax.

Programming note: *The default FRAMESIZE is 48 (that is, words). If you want to specify the MAXRECSIZE and BLOCKSIZE attributes in bytes, you must include FRAMESIZE=8 as a file attribute specification.*

When used in an output file specification, FR specifies that the output tape is to be unlabeled and formatted as described under “Unlabeled Tapes” later in this section. The operating system uses as many tapes as are required to hold the data to be copied.

The use of FR instead of UL is important for input files only. That is, when you use FR in an input file specification, it indicates that only one physical tape is to be used. The benefit of using FR is that when DUMPALL reads a tape mark from the tape, it closes the input file and does not need an operator response. Refer to “Input Files from Unlabeled Tapes” later in this section.

(<file attribute>)

Specifies how the file is to be written or read. Except for the file attributes you specify, DUMPALL obtains the file attribute information for each input file from the disk file header or the tape label. Except for the file attributes you specify and the FILESTRUCTURE, SERIALNO and HOSTNAME attributes, DUMPALL obtains the file attribute information for each output file from the first input file copied to that output file.

DUMPALL determines the value of the KIND attribute for input files and output files by one of the following processes:

- If you specified either UL or FR, then KIND is TAPE.
- If you specified a file title with an explicit ON family name clause, then KIND is TAPE or DISK depending on whether or not you specified the family name ON TAPE.

DUMPALL Utility

- If you explicitly specify a value for the KIND in the <file attribute> syntax, then DUMPALL uses that value.
- Otherwise DUMPALL determines the value for KIND automatically as follows:
 - For the first input file in the command, DUMPALL searches for the file on disk, tape, and cards.
 - For all subsequent input files in the command for which you have not explicitly or implicitly specified a value for KIND, DUMPALL uses the same value of KIND as the preceding input file.
 - For the first output file in the command DUMPALL uses the same KIND value as the first input file in the command.
 - For all subsequent output files in the command for which you have not explicitly or implicitly specified a value for KIND, DUMPALL uses the same value of KIND as the preceding output file.

If you do not specify a value for SAVEFACTOR for an output file, DUMPALL assigns the output file the same SAVEFACTOR value as the first input file. If that value is 0, DUMPALL uses 2 instead.

Programming Note: *A tape file with a SAVEFACTOR of zero expires the day it is created. The system treats tapes containing expired tape files as scratch tapes unless you remove the write ring from the reel or switch the write-protection knob on the tape cartridge.*

If you specify the HOSTNAME file attribute, DUMPALL accesses the file on a remote host system. In this case, distributed system services (DSS) requires that you also explicitly specify the KIND file attribute.

The file attribute equation specifies how the file is to be written or read. If no file attributes are specified for the input file, DUMPALL opens the input file with the following logic:

- Labeled tape files are opened with DEPENDENTSPECS set to TRUE, INTMODE set to EBCDIC, and EXTMODE set to the physical mode of the file.
- Disk files are opened with DEPENDENTSPECS set to TRUE and with INTMODE and EXTMODE set to the physical mode of the input file.

The logical input file resembles the physical input file for disk files, and resembles the physical file for tape files if the tape file has a physical mode of EBCDIC.

The record and block structures of the output file are the same as those of the first input file if you do not specify any file attributes for either the input file or the output file, and if the input file is either a tape file with a physical mode of EBCDIC or a disk file. In this case, no data translations occur, so DUMPALL does not alter the records it copies from the input file or files to the output file.

Specifying input file attributes such as DEPENDENTSPECS, EXTMODE, and INTMODE might cause data translation and the alteration of other file attributes (for example, MAXRECSIZE). Specification of such attributes might also avoid data

translation if the input file is a not an EBCDIC tape file. If data translation does occur, the output file might not directly resemble the input file.

Refer to “Understanding Structural File Attributes” earlier in this section for an explanation of the file structure, record size, block size, the DEPENDENTSPECS attribute, and their effects on DUMPALL.

Refer to “Specifying Data or Character Set Translations” earlier in this section for an explanation of the effects of the EXTMODE and INTMODE file attributes on DUMPALL.

<record range list>
<skip specification>

Cause only the specified portion of the file to be copied. If you do not supply a record range list or a skip specification, DUMPALL copies the entire file.

SKIPTM <integer>

Causes DUMPALL to skip past the number of tape marks specified and to bypass any records encountered between tape marks. Use SKIPTM to position an input tape at the first record of a file to be read. You can use SKIPTM only with an input file that you specify as UL or FR.

Refer to “Description of Tape Formats” later in this section for an explanation of where tape marks appear on a tape.

MULTIFILE

MULTI

Specify that an input or output file is part of a multifile labeled or unlabeled tape. After DUMPALL finishes copying the file with the MULTIFILE specification to or from a tape, the tape does not rewind. The operating system positions the tape so that DUMPALL can read or write to a subsequent file on the same volume. In other words, MULTIFILE leaves the tape assigned and positioned so that DUMPALL can copy a subsequent file to or from the tape. DUMPALL ignores the MULTIFILE option if the file is not a tape file.

You must use MULTIFILE to copy a file from a multifile tape if the following conditions are met:

- The file has a nonstandard name. Normally, a tape file has a two-level name of the following form: volumeid/fileid. A nonstandard tape file name is a name with only one level, such as F1 or MYFILE.
- The tape is a standard labeled tape.
- The file is not the first file on the tape.

In such a case, when DUMPALL tries to open the file, the system produces a “NO FILE <filename> (MT)” RSVP message. You must reply with the IL (Ignore Label) system command to select the tape unit on which the tape with the requested file is mounted.

Normally, when DUMPALL finishes copying a file or several files to a tape, it closes the tape with the LOCK option. So, depending on the kind of tape unit involved, the system either unloads the tape volume or marks the unit as locked. However, if you specify MULTIFILE on the last or only output file that you want copied to tape, DUMPALL does not close the tape with the LOCK option, and the tape simply rewinds. When the rewinding finishes, the system leaves the tape online and ready for use in subsequent DUMPALL commands or subsequent programs.

CRUNCH

When the output disk or pack file is closed, DUMPALL returns the unused portion of the last row of disk space to the system. DUMPALL ignores the CRUNCH option if the file is not a disk or pack file.

DONTPRINT

Suppresses the printed report of the file attributes of the input files and output files.

Examples of the COPY Command

This discussion provides three types of examples:

- General examples that illustrate various features of the COPY command.
- Examples that illustrate how to copy files from tapes to disk. These examples include labeled, nonstandard labeled, and unlabeled tapes for single and multifile tapes.
- Examples that illustrate how to copy files to and from tapes.

General Examples

The examples that follow illustrate various features of the COPY command.

Example 1

Example 1 copies the file INFILE to the file OUTFILE. The attributes of OUTFILE are the same as those for INFILE. DUMPALL searches for an online tape file with the name INFILE, for an input card reader file with the name INFILE, and for a disk file named INFILE on the family DISK. If INFILE is on tape, DUMPALL writes OUTFILE to tape. If INFILE is on disk, DUMPALL writes OUTFILE to disk. If INFILE is on a card reader file, DUMPALL receives an "OPEN ERROR" message on OUTFILE, because an output file cannot be written with KIND = READER. For an input card reader file, you must specify a valid output KIND for OUTFILE, such as DISK, TAPE, PRINTER, or PUNCH.

```
COPY INFILE TO OUTFILE;
```

Example 2

Example 2 creates a new file named F3 and copies to it all the records from file F1 followed by all the records from file F2:

```
COPY F1 THEN F2 TO F3;
```

Example 3

Example 3 copies files F1 and F2 to the file F3, closes F3, and then copies files F4 and F5 to the file F6:

```
COPY F1 THEN F2 TO F3
  THEN F4 THEN F5 TO F6;
```

Example 4

Example 4 creates two new files: OUTFILE1 and OUTFILE2. OUTFILE1 has the same file attributes and KIND as the input file FILE1 and receives copies of records 20 through 49 of FILE1. OUTFILE2 is a disk file. Its other file attributes match those of input FILE2. OUTFILE2 receives copies of records 1 through 5 of FILE2 and is closed with CRUNCH.

```
COPY FILE1 REC 20 THRU 49 TO OUTFILE1
  THEN FILE2 REC 1 THRU 5
  TO OUTFILE2 (DISK) CRUNCH;
```

Example 5

Example 5 creates a single-file labeled tape with a standard tape name—that is, a volume identifier of T and a file identifier of FILEONE:

```
COPY FILEONE TO T/FILEONE (KIND=TAPE);
```

Example 6

Example 6 copies three files to a multifile labeled tape. This example generates a tape with nonstandard filenames, because the output filename does not include a volume name; it contains only file identifiers. Notice that it is necessary to specify KIND = TAPE for only for the first output file. Each DUMPALL output file assumes the same KIND as the previous output file unless you specify otherwise. The files are separated by the standard tape header and trailer records (HDR1, HDR2, EOF1, EOF2).

```
COPY F1 TO FONE (KIND = TAPE), MULTIFILE
  THEN F2 TO FTWO, MULTIFILE
  THEN F3 TO FTHREE;
```

Example 7

Example 7 copies file F2 to disk from the tape generated by Example 6. The MULTIFILE option is necessary because the tape does not use standard tape names of

DUMPALL Utility

the form volumeid/fileid and because F2 is not the first file on the tape. The operator must reply to the RSVP message "NO FILE F2 (MT #1)" with the IL (Ignore Label) system command to direct DUMPALL to the tape unit that contains the requested tape volume.

```
COPY FTWO MULTIFILE (KIND=TAPE) TO F2 ON MYPACK
```

Example 8

Example 8 creates a new file named B and copies the records from file A to it. The new file has the same file attributes as file A. DUMPALL does not print a report showing the file attributes.

```
COPY A TO B : DONTPRINT;
```

Examples of Copying Files from Tapes to Disk

The following are examples of copying single and multiple files from labeled tapes, unlabeled tapes, and nonstandard labeled tapes to disk.

Example 1

Example 1 copies a tape file named TX/DATA from a labeled tape to DISK. It gives the copy on DISK the file name NEW/TX/DATA. The disk file automatically inherits file attributes such as MAXRECSIZE, BLOCKSIZE, FRAMESIZE, and EXTMODE from the tape file.

```
COPY TX/DATA (TAPE) TO NEW/TX/DATA (DISK);
```

Example 2

Example 2 copies a file from an unlabeled tape to the disk family named PACK. It gives the file on PACK the file name CD/FILE. You must specify file attributes of the input file such as BLOCKSIZE and EXTMODE, because DUMPALL cannot determine the proper values from the tape itself.

```
COPY UL (FRAMESIZE=8, MAXRECSIZE=80, BLOCKSIZE=240,  
EXTMODE=EBCDIC)  
TO CD/FILE ON PACK;
```

Example 3

Example 3 copies a file from a tape with nonstandard labels to the disk family named DPPACK. The copy of the file on disk is given the name F/XFILE. Because the tape has nonstandard labels, DUMPALL cannot copy the file from tape by its name. Instead, you must use the UL option. Because DUMPALL cannot determine the proper values for file attributes such as record size from the nonstandard tape labels, you must specify them in the command. And finally, use SKIPTM to skip over the tape mark that separates the labels from the first file on the tape.

```

COPY UL SKIPTM 1
  (FRAMESIZE=8, MAXRECSIZE=120, BLOCKSIZE=720,
   EXTMODE=EBCDIC)
  TO F/XFILE ON DPPACK;

```

Example 4

Example 4 copies three named files from a labeled tape to a disk. The name of the tape is XTAPE and the names of the three files to be copied are XTAPE/DATA, XTAPE/SYM, and XTAPE/DOC. These tape files are to be copied to DISK with the file names PROG/DATA, PROG/SYMBOL, and PROG/DOC respectively. Each copy automatically inherits the specific values of file attributes such as record size and EXTMODE from the corresponding tape file. The copy of the symbol is given the FILEKIND of COBOLSYMBOL. The other two files receive the default FILEKIND of DATA. Tape files do not have a FILEKIND attribute, so when you copy from tape, DUMPALL uses the default DATA for disk copies unless you specify an explicit value for the FILEKIND attribute. For efficiency purposes, you should list the files to be copied in the order they appear on the input tape unless you do not know the order. It is not necessary to repeat the kinds TAPE and DISK for each input and output file, because DUMPALL automatically assumes that each input or output file has the same KIND as the first input or output file unless otherwise specified.

```

COPY XTAPE/DATA MULTIFILE (TAPE) TO PROG/DATA (DISK),
  THEN XTAPE/SYM MULTIFILE
  TO PROG/SYMBOL (FILEKIND=COBOLSYMBOL),
  THEN XTAPE/DOC TO PROG/DOC;

```

Example 5

Example 5 copies three files from an unlabeled tape to disk. The input tape actually has four files on it, but only the first, second, and fourth files are to be copied. The input files do not have names, but the disk files receive the names T/F1, T/F2, and T/F4. You must specify file attributes such as record size and EXTMODE for the tape files, because DUMPALL cannot automatically determine the actual values for files on unlabeled tapes. The second and fourth files have the same attributes, so you do not have to repeat the values in the command.

```

COPY UL MULTIFILE
  (FRAMESIZE=8, MAXRECSIZE=100, BLOCKSIZE=900,
   EXTMODE=EBCDIC)
  TO T/F1 ON DISK
  THEN UL MULTIFILE
  (FRAMESIZE, MAXRECSIZE=80, BLOCKSIZE=80,
   EXTMODE=EBCDIC)
  TO T/F2
  THEN UL SKIPTM 1
  TO T/F4;

```

Example 6

Example 6 copies three files from a tape with nonstandard labels. Because the system cannot understand the labels, you must use the UL or FR specification instead of file names in the COPY command to locate the tape files. You must use MULTIFILE, or the COPY command copies the first file from three separate tapes. You must use SKIPTM to skip labels: the first SKIPTM skips over the beginning labels for the first file; the second SKIPTM skips over the ending labels of the first file and the beginning labels of the second file; and the third SKIPTM skips over the ending labels of the second file and the beginning labels of the third file. You must specify file attribute values for the input files, because DUMPALL cannot interpret the file attributes recorded in nonstandard labels. To change the block size of the first file to a larger size on disk, you must also specify this file attribute for the first output file.

```
COPY UL SKIPTM 1, MULTIFILE,  
      (FRAMESIZE=8, MAXRECSIZE=450, BLOCKSIZE=450)  
      TO FT/FILE1 ON DISK (BLOCKSIZE=900)  
      THEN UL SKIPTM 2, MULTIFILE,  
      (FRAMESIZE=8, MAXRECSIZE=70, BLOCKSIZE=770)  
      TO FT/FILE2 ON DISK  
      THEN UL SKIPTM 2,  
      (FRAMESIZE=8, MAXRECSIZE=98, BLOCKSIZE=98)  
      TO FT/FILE3 ON DISK;
```

Examples of Copying Files to and from Tapes

The examples that follow illustrate how to copy files to and from tapes.

Example 1

Example 1 creates a single-file labeled tape.

```
COPY FILEONE TO T/FILEONE (KIND=TAPE)
```

You create a single-file labeled tape by specifying a file title instead of UL or FR in the output file portion of the COPY command and by specifying a KIND value of TAPE in the output file attribute list. Many of the disk file attributes, such as BLOCKSIZE and MAXRECSIZE are written into the tape labels so that they are preserved on tape. This process makes it easier to copy the file back to disk with DUMPALL and makes it easy for programs to read the file directly from tape.

Programming note: *Some programs that read the disk file sequentially from disk could use the tape file T/FILEONE directly. Simply enter label equation statements in those programs to reference the tape file so you do not have to copy the file back to disk to use it.*

DUMPALL reads a single-file labeled tape when you specify a file title in the input file portion of the COPY command corresponding to a file on the tape. KIND must equal TAPE in the input file attribute list.

If multiple physical tape volumes are required to contain the copied file, the I/O subsystem automatically requests the extra volumes as they are needed.

Example 2

Example 2 copies T/FILEONE back to disk:

```
COPY T/FILEONE (KIND=TAPE) TO FILEONE(KIND=PACK, PACKNAME=MYPACK);
```

When the file is copied, some of the file attributes, such as the BLOCKSIZE and MAXRECSIZE, of the original disk file are preserved when the file is copied. Other attributes, such as CREATIONDATE and TIMESTAMP, acquire new values. Some attributes, such as USERINFO, are lost. And some attributes, such as AREASIZE and FILEKIND, are given default values when the file is copied from tape to disk.

Example 3

Example 3 creates a single-file labeled tape:

```
COPY FILEONE THEN FILETWO THEN FILETHREE TO T/FIRSTSET (KIND=TAPE)
```

The file T/FIRSTSET is a single file that contains the records in FILEONE, followed by the records in FILETWO, followed by the records in FILETHREE. To copy the file T/FIRSTSET back to disk, use the following command:

```
COPY T/FIRSTSET (KIND=TAPE) TO FIRSTSET (KIND=PACK,PACKNAME=MYPACK);
```

The separate files used to create the tape can be retrieved only by using a record range list. If the identity of the separate files is to be maintained, create a multifile tape.

Example 4

Example 4 controls the output tape selection by specifying a serial number list.

```
COPY BIG/FILE TO T/FILEONE(KIND=TAPE, SERIALNO=("SN1","SN2","SN3"));
```

When you create a labeled tape and the file that is being written to the tape requires more than one tape, the operating system takes appropriate action as each additional tape is required.

Example 5

Example 5 creates a multifile labeled tape containing the files T/FILEONE, T/FILETWO, and T/FILETHREE:

```
COPY FILEONE TO T/FILEONE (KIND=TAPE) MULTI  
THEN FILETWO TO T/FILETWO (KIND=TAPE) MULTI  
THEN FILETHREE TO T/FILETHREE (KIND=TAPE);
```

You can create a multifile labeled tape by using the word MULTI instead of UL or FR in the specification for each output file that is to be followed by another input file. If

DUMPALL Utility

multiple physical tapes are required to contain the set of files written, tapes are selected automatically.

Similarly, DUMPALL reads a multifile labeled tape if you use the word MULTI in each input file specification that is to be followed by another input file from the same tape.

Example 6

Example 6 copies a selected file back to disk:

```
COPY T/FILETWO (KIND=TAPE) TO FILETWO(KIND=PACK, PACKNAME=MYPACK);
```

Example 7

Example 7 copies multiple files back to disk:

```
COPY T/FILEONE (KIND=TAPE) MULTI  
TO FILEONE (KIND=PACK, PACKNAME=MYPACK)  
THEN T/FILETHREE  
TO FILETHREE (KIND=PACK, PACKNAME=MYPACK);
```

Example 8

Example 8 creates a multifile labeled tape containing the files T/FIRSTSET and T/SECONDSET:

```
COPY FILEONE THEN FILETWO THEN FILETHREE  
TO T/FIRSTSET (KIND=TAPE) MULTI  
THEN FILEFOUR THEN FILEFIVE THEN FILESIX  
TO T/SECONDSET (KIND=TAPE);
```

T/FIRSTSET consists of the records of FILEONE, FILETWO, and FILETHREE. T/SECONDSET consists of the records of FILEFOUR, FILEFIVE, and FILESIX. You can copy either or both files back from the tape by using the following commands:

```
COPY T/SECONDSET (KIND=TAPE) TO SECONDSET(KIND=PACK, PACKNAME=MYPACK);
```

```
COPY T/FIRSTSET (KIND=TAPE) MULTI  
TO FIRSTSET (KIND=PACK, PACKNAME=MYPACK)  
THEN T/SECONDSET (KIND=TAPE)  
TO SECONDSET (KIND=PACK, PACKNAME=MYPACK);
```

You can retrieve individual files that comprise FIRSTSET and SECONDSET only if you specify a record range list.

Example 9

Example 9 creates a single-file unlabeled tape.

```
COPY FILEONE TO FR;
```

You might want to copy files to unlabeled tapes if you need to take those tapes to systems other than A Series systems that might not be able to read the tape labels produced by A Series systems. You can create an unlabeled tape by specifying UL or FR in the output file specification of the COPY command.

If you want DUMPALL to read files from an unlabeled tape, specify UL or FR in the input file portion of the COPY command. FR causes DUMPALL to use only one input tape volume, while UL permits DUMPALL to use more than one input tape volume.

The use of unlabeled input tapes requires greater user involvement, because the only information available to DUMPALL is data separated by tape marks. You cannot use file names to locate the proper input file or files. You might have to use SKIPTM to position the tape to the proper file. You must specify structural file attributes such as FRAMESIZE, MAXRECSIZE, and BLOCKSIZE for each unlabeled input tape file.

If all of FILEONE fits on a single tape volume, use a command in the following format to copy it from tape back to disk:

```
COPY FR (<file attribute>)  
    TO FILEONE (KIND=PACK, PACKNAME=MYPACK);
```

If FILEONE is so large that it requires more than one tape volume, use a command in the following format to copy it from tape back to disk:

```
COPY UL (<file attribute>)  
    TO FILEONE (KIND=PACK, PACKNAME=MYPACK);
```

In the previous two examples, you must specify file attributes such as FRAMESIZE, MAXRECSIZE, and BLOCKSIZE.

Example 10

Example 10 copies the records of three files to one or more unlabeled tapes:

```
COPY FILEONE THEN FILETWO THEN FILETHREE TO FR;
```

When the command has been executed, the file on the tape or tapes contains the records in FILEONE, followed by the records in FILETWO, followed by the records in FILETHREE. To copy the concatenated file back to disk, use the following command:

```
COPY UL TO ONETWOTHREE (KIND=PACK, PACKNAME=MYPACK);
```

Example 11

Example 11 creates a multifile, unlabeled tape containing three files:

DUMPALL Utility

```
COPY FILEONE TO UL MULTI THEN FILETWO  
TO UL MULTI THEN FILETHREE TO UL;
```

Example 12

Label information is not available when you copy files back from unlabeled tapes. Therefore, you must position the tape at the file or files to be copied. You can copy any number of files from the tape in succession by using one of the following commands:

```
COPY UL TO FILEONE (KIND=PACK, PACKNAME=MYPACK);  
  
COPY UL MULTI TO FILEONE (KIND=PACK, PACKNAME=MYPACK)  
THEN UL TO FILETWO (KIND=PACK, PACKNAME=MYPACK);  
  
COPY UL MULTI TO FILEONE (KIND=PACK, PACKNAME=MYPACK)  
THEN UL MULTI TO FILETWO (KIND=PACK, PACKNAME=MYPACK)  
THEN UL TO FILETHREE (KIND=PACK, PACKNAME=MYPACK);
```

To copy a specific file, you must use the `SKIPTM` option to position the tape to the desired file. `DUMPALL` executes the `SKIPTM` option before it reads the file. To copy the second file on the tape for the previous example, use the following command:

```
COPY UL SKIPTM 1 TO FILETWO (KIND=PACK, PACKNAME=MYPACK);
```

At the time `DUMPALL` executes this command, the operating system has positioned the unlabeled tape so that `DUMPALL` can read the first record of the first file on the tape. `DUMPALL` first skips one tape mark. Because the first tape mark is located between the first and second files on the tape, skipping one tape mark positions the tape so that `DUMPALL` can read the first record of the second file on the tape.

The following example copies files one and three:

```
COPY UL MULTI TO FILEONE (KIND=PACK, PACKNAME=MYPACK)  
THEN UL SKIPTM 1 TO FILETHREE (KIND=PACK, PACKNAME=MYPACK);
```

At the time `DUMPALL` executes this command, the operating system has positioned the unlabeled tape so that `DUMPALL` can read the first record of the first file on the tape. `DUMPALL` reads the first input file—that is, the first file on the tape—and creates the first output file, `FILEONE`. `MULTI` indicates that there is a subsequent file to be copied, and so the tape does not rewind. `DUMPALL` then proceeds to the second input file. Currently, the tape is positioned to read the second file on the tape. The `SKIPTM 1` syntax causes `DUMPALL` to bypass this file to skip one tape mark between files two and three. This action positions the tape at the first record of the third file on the tape. `DUMPALL` then creates `FILETHREE`.

Example 13

Example 13 creates a multifile unlabeled tape containing two files:

```
COPY FILEONE THEN FILETWO THEN FILETHREE TO UL MULTI  
THEN FILEFOUR THEN FILEFIVE THEN FILESEX TO UL;
```

The first file is a concatenated file containing the records of FILEONE, followed by those in FILETWO, followed by those in FILETHREE. To copy either or both files back to disk, you can use one of the following three commands:

```
COPY UL TO ONETWOTHREE (KIND=PACK, PACKNAME=MYPACK);

COPY UL TO ONETWOTHREE (KIND=PACK, PACKNAME=MYPACK) MULTI
  THEN UL TO FOURFIVESIX (KIND=PACK, PACKNAME=MYPACK);

COPY UL SKIPTM 1
  TO FOURFIVESIX (KIND=PACK, PACKNAME=MYPACK);
```

You can retrieve the individual files that comprise each of the files on the tape only if you specify a record range list.

Example 14

Example 14 copies the first, fourth, and fifth files from an unlabeled multifile tape, and is limited to a single physical tape:

```
COPY FR MULTI TO FILEONE (KIND=PACK, PACKNAME=MYPACK)
  THEN FR MULTI SKIPTM 2
    TO FILEFOUR (KIND=PACK, PACKNAME=MYPACK)
  THEN FR TO FILEFIVE (KIND=PACK, PACKNAME=MYPACK);
```

Example 15

Example 15 copies the first, fourth, and fifth files from an unlabeled multifile tape. Although UL is used, the tape is limited to a single physical tape, because MULTI and SKIPTM are used.

```
COPY UL MULTI TO FILEONE (KIND=PACK, PACKNAME=MYPACK)
  THEN UL MULTI SKIPTM 2
    TO FILEFOUR (KIND=PACK, PACKNAME=MYPACK)
  THEN UL TO FILEFIVE (KIND=PACK, PACKNAME=MYPACK);
```

Example 16

If you need to copy a file that spans more than one tape volume from unlabeled tapes, and if you need to use SKIPTM to position the input tape to the start of the file, specify the input tape file as though it were two or more input files. Suppose you have copied several files to a two-volume multifile unlabeled tape set. Suppose that while DUMPALL is copying the third file to the tapes, the end of the first volume is reached, and the system switches to the second volume. Then the first part of the third file is on the first volume and the second part of the third file is on the second volume. To copy the entire third file from those tapes to disk, use a command in the following format:

```
COPY UL SKIPTM 2 (<file attribute>)
  THEN UL (<file attribute>) TO FILETHREE (DISK);
```

DUMPALL Utility

Example 17

Example 17 copies a file that is contained on one or more tapes to a disk:

```
COPY UL TO FILEONE (KIND=PACK, PACKNAME=MYPACK);
```

As each end of tape is reached, the system asks the operator if there are more input tapes.

Example 18

Example 18 duplicates one file from a labeled tape containing the file T/FILEONE:

```
COPY T/FILEONE (KIND=TAPE) TO T/FILEONE (KIND=TAPE);
```

To duplicate all or part of a labeled tape, you must specify each input file from the input tape and each output file. The tape can be single-file or multifile.

Example 19

Example 19 duplicates a multifile labeled tape containing the files T/FILEONE, T/FILETWO, and T/FILETHREE:

```
COPY T/FILEONE (KIND=TAPE) MULTI  
  TO T/FILEONE (KIND=TAPE) MULTI  
  THEN T/FILETWO (KIND=TAPE) MULTI  
    TO T/FILETWO (KIND=TAPE) MULTI  
  THEN T/FILETHREE (KIND=TAPE)  
    TO T/FILETHREE (KIND=TAPE);
```

Example 20

Example 20 duplicates the first file of an unlabeled tape:

```
COPY UL (<file attribute>) TO UL
```

File attributes include **FRAMESIZE**, **MAXRECSIZE**, and **BLOCKSIZE**. The use of **UL** for the input file indicates that the input file can be a multivolume file.

To duplicate all or part of an unlabeled tape, you must specify an input file and an output file for each file on the tape to be copied. The tape can be single-file or multifile.

Example 21

Example 21 duplicates the third, fourth, and fifth files of a multifile unlabeled tape:

```
COPY UL SKIPTM 2 MULTI (<file attribute>) TO UL MULTI  
  THEN UL          MULTI (<file attribute>) TO UL MULTI  
  THEN UL          (<file attribute>) TO UL;
```

The use of the SKIPTM 2 option causes the first two files on the tape to be skipped. The file attributes for each input file must include the FRAMESIZE, MAXRECSIZE, and BLOCKSIZE for the file. The use of MULTI in the input file specification causes a subsequent file to be copied from the tape being duplicated; the use of MULTI in the output file specification causes a subsequent file to be copied to the duplicate tape being created. For this example, UL for the input files functions as if it were FR, because MULTI and SKIPTM are being used.

Considerations for Use

Except for file attributes you explicitly specify for the output file, the attributes for the output file are determined as follows:

- Some of the file attributes, such as BLOCKSIZE and MAXRECSIZE, of the output file are set to the same values as those of the input file.
- Other file attributes, such as CREATIONDATE and TIMESTAMP, receive new values.
- Some attributes, such as USERINFO, FILESTRUCTURE, and CRUNCHED, are not preserved in the output file; these attributes get default values.
- Some attributes, such as AREASIZE and FILEKIND of disk files, are given default values unless the input file is being copied from disk.

If you use the COPY command to copy a code file, the resulting file has a FILEKIND attribute equal to DATA.

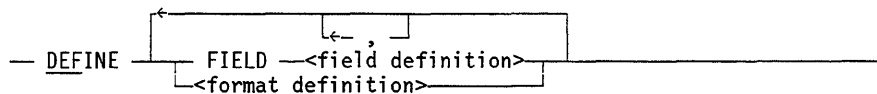
You can use the COPY command to copy certain ISAM files, but many restrictions exist.

- You cannot use DUMPALL to copy files with a FILEORGANIZATION attribute equal to PLIISAM, KEYEDIOII, or KEYEDIOISET. To copy KEYEDIOII and KEYEDIOISET files, use the KEYEDIOII Copy Utility described in the *KEYEDIOII Reference Manual*.
- You can use DUMPALL to copy an input file with a FILEORGANIZATION attribute value equal to INDEXED or INDEXEDNOTRESTRICTED. DUMPALL does not copy any records that have been marked as deleted from these files. You can copy these files to output files that have the FILEORGANIZATION value equal to NOTRESTRICTED, INDEXED, or INDEXEDNOTRESTRICTED.
 - You can copy these input files to an output file with a FILEORGANIZATION attribute value of INDEXED or INDEXEDNOTRESTRICTED if the system option KEYEDIOII is reset to FALSE.
 - You can copy such input files to files that are not ISAM files by either of the following actions:
 - By specifying that the output file has a FILEORGANIZATION attribute value equal to NOTRESTRICTED.
 - By copying the files to media other than disk. For example, you can copy them to tape.

DEFINE Command

The DEFINE command declares and specifies fields and formats for use in subsequent print commands such as LIST and DMPMT. Refer to “Format Definition” and “Field Definition” later in this section.

Syntax



Explanation

You must assign a mnemonic name to each format that you define in this command. You must also assign a mnemonic name to any field that you define that is not part of a format definition.

Example 1

Example 1 declares and names the field F1 for use in subsequent print commands. F1 describes a field that contains the first four bytes of a record in EBCDIC format. EBCDIC is the default when the field offset is specified in bytes.

```
DEFINE FIELD ['F1'] := BYTE 0 FOR 4]
```

Example 2

Example 2 declares and names the format FMT1 for use in subsequent print commands. FMT1 consists of the field F2, which is also entered into the defines file, the field F1, which has been previously defined, and the unnamed field describing byte 80 of the record.

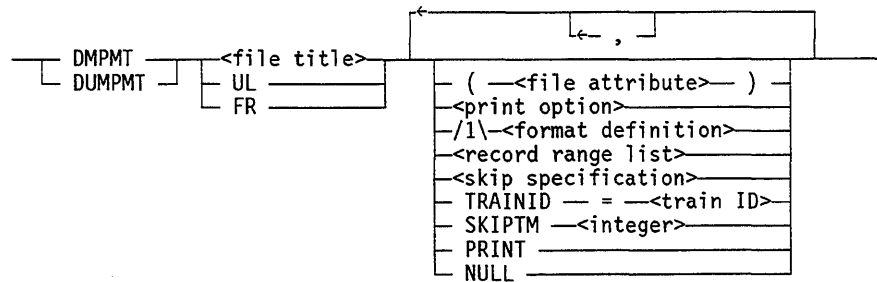
```
DEFINE FORMAT 'FMT1' := ['F2' := WORD 7 DEC] 'F1' [BYTE 80]
```

DMPMT Command

The DMPMT command prints the blocks read from a tape volume. When DUMPALL executes a DMPMT command, it does not use the BLOCKSIZE and MAXRECSIZE attributes of the file or files it reads. DUMPALL simply reads each block of data and prints it as a single entity. You can use the DMPMT command as an aid when you are trying to determine the file structure, such as MAXRECSIZE and BLOCKSIZE, of files on unlabeled tapes or files on tapes with nonstandard labels.

DMPMT and DUMPMT are synonyms.

Syntax



Explanation

<file title>

Specifies the name of the first file on the tape to be printed. DUMPALL prints that file and all subsequent files on the tape.

UL FR

Specifies an unlabeled tape—that is, a tape with a LABELTYPE attribute value of OMITTED.

Data read from the tape is printed until a double tape mark is found. DUMPALL does not attempt to switch to the next volume, if any, when it reaches the double tape mark.

<file attribute>

Causes DUMPALL to use the indicated file attributes when dumping the file. DUMPALL uses direct I/O for reading the input tape in the DMPMT command. DUMPALL does not use the MAXRECSIZE value, and data translations do not occur. If you do not specify the KIND value of the input file, the default is TAPE.

For tapes that do not have standard Large Systems (LS) labels, the default MAXRECSIZE is 10 words unless you specify otherwise in the file attribute option. Refer to the *File Attributes Reference Manual* for information about all the file attributes.

DUMPALL Utility

<print option>

Specifies the format to be used when the file or files are printed. If you do not use a print option or a format specification variable in the command, the files are printed in EBCDIC, hexadecimal, and octal.

You can specify one or more print options. If you specify more than one print option, DUMPALL lists the record in word-sized chunks, one format below the other on a page.

Because DECIMAL is a subset of REAL, if you specify both, only REAL formatting takes place.

If you specify a single print option and it is EBCDIC or ASCII, DUMPALL lists the record as one unit in the format specified in the print option. If you specify a single print option and it is REAL, DECIMAL, HEXADECIMAL, or OCTAL, DUMPALL lists the record in word-sized chunks.

<format definition>

Describes one or more fields of the block and their formats. If you specify any formats in the DMPMT command, DUMPALL prints only the fields defined in those formats. DUMPALL does not print the rest of each block.

Note that the operation of fields is different for the DMPMT command than for other print commands. Because DMPMT handles each block read from the tape as a single entity or record, the offset values in field definitions refer to the beginning of the block, not the beginning of records. For example, suppose the blocks on a tape contained three records each, such as MAXRECSIZE=9, BLOCKSIZE=27. Normally, a field definition would be applied to each of the three records in a block. But for the DMPMT command, a field definition applies to the block. In this case, if you want to display the values of the field for all three records in each block, you would have to specify three different fields with three different offsets.

<record range list>

<skip specification>

Cause only the portion of the file specified by the record range list or the skip specification to be printed. The DMPMT command handles individual records rather than blocks. Therefore, be aware of the following:

- The record numbers in a record range list are treated as block numbers.
- The count of records in a record range list and a skip specification are treated as a count of blocks.

TRAINID = <train ID>

Specifies the train to be used on the printer when the file is listed.

The train ID can be any valid train ID. Refer to the *File Attributes Reference Manual* for a description of the TRAINID attribute and for a list of valid train IDs.

SKIPTM <integer>

Specifies the number of tape marks to be skipped before the file is printed. Refer to "Description of Tape Formats" later in this section for an explanation of where tape marks appear on a tape.

PRINT

Used in interactive mode to send the output to the printer instead of the remote terminal.

NULL

Gives the block size, block number, and I/O result for each block in the file. DUMPALL does not list the actual contents the file.

Considerations for Use

The DMPMT command is used to print the contents of a tape in whole or in part. By default, DUMPALL prints the data on the tape in EBCDIC, hexadecimal, and octal.

The main differences between the DMPMT command and the LIST command are the following:

- DMPMT deals with blocks; LIST deals with records.
- DMPMT prints the file you specify plus all the subsequent files on the tape; LIST prints only one file.
- DMPMT prints tape files; LIST can print different kinds of input files including disk, card, and tape files, and files on other host systems.
- DMPMT reads only one volume; LIST can print multivolume files.

When you use a file title, DUMPALL expects a labeled tape. In this case, DMPMT prints the contents of each file on the tape, starting with the file specified in the command.

Example 1

Example 1 dumps a tape named DATATAPE. The output is printed in word groups (6 bytes or 12 hexades per word).

```
DMPMT DATATAPE EBCDIC, HEX
```

Example 2

Example 2 prints the entire contents of one tape:

```
DMPMT UL
```


DUMPALL Utility

When you specify UL or FR, DUMPALL treats the tape as unlabeled. The entire contents of the tape, including tape marks, starting at the position you specified with the SKIPTM option, are printed until DUMPALL encounters a double tape mark.

Example 3

Example 3 prints the entire contents of one tape starting after the third tape mark:

```
DMPMT UL SKIPTM 3
```

HEXDSK Command

The HEXDSK command lists a file on a sector-by-sector basis no matter what type of file structure the file has. When it executes the HEXDSK command, DUMPALL ignores the file attributes for MAXRECSIZE and BLOCKSIZE. The listing of the file includes those areas of the file that are not normally accessible to programs, such as gaps between blocks and any part of a sector that includes the end-of-file (EOF) character that is past the actual EOF. The HEXDSK command lists a file both in hexadecimal format and in ASCII or EBCDIC. If the EXTMODE value of the file is ASCII, the second list is in ASCII; otherwise, the second list is in EBCDIC.

Syntax

```
— HEXDSK —<file title> —————|
                                   |<record range list>|
                                   |<skip specification>|
```

Explanation

HEXDSK <file title>

Lists the entire file in hexadecimal and EBCDIC formats.

<record range list>

<skip specification>

Cause only the portion of the file specified by the record range list or skip specification to be listed. In this command, the record range list and skip specification refer to sectors instead of records. The first sector of a file is sector number 1.

Example

The following example shows both the input to and the output from a HEXDSK command. The output is of sector 4 of the disk file MYFILE/TEST. DUMPALL writes the text in word-sized chunks in EBCDIC and then in hexadecimal.

```
HEXDSK MYFILE/TEST REC 4

SECTOR NUMBER: 00004

EBCDI      C ARRA      Y ANAM      E [0:2      5];
40C5C2C3C4C9 C340C1D9D9C1 E840C1D5C1D4 C5404AF07AF2 F55A5E404040

404040404040 404040404040 404040404040 404040404040 404040404040

                                000002      30
404040404040 404040404040 F0F0F0F0F0F2 F3F040404040 404040404040
```

DUMPALL Utility

```
TRUTH      SET NU      MBERS      ("0123      456789
40E3D9E4E3C8 E2C5E340D5E4 D4C2C5D9E240 4D7FF0F1F2F3 F4F5F6F7F8F9

");
7F5D5E404040 404040404040 404040404040 404040404040 404040404040

                                000002      35
404040404040 404040404040 F0F0F0F0F0F2 F3F540404040 404040404040
```

LIBMT Command

The LIBMT command lists a library maintenance tape in hexadecimal format. The tape must have been made available by using the UL system command. Refer to the *A Series System Commands Operations Reference Manual* for information about the UL command. The LIBMT command is not available in interactive mode; you cannot use the command with segmented tapes.

Syntax

— LIBMT —————|

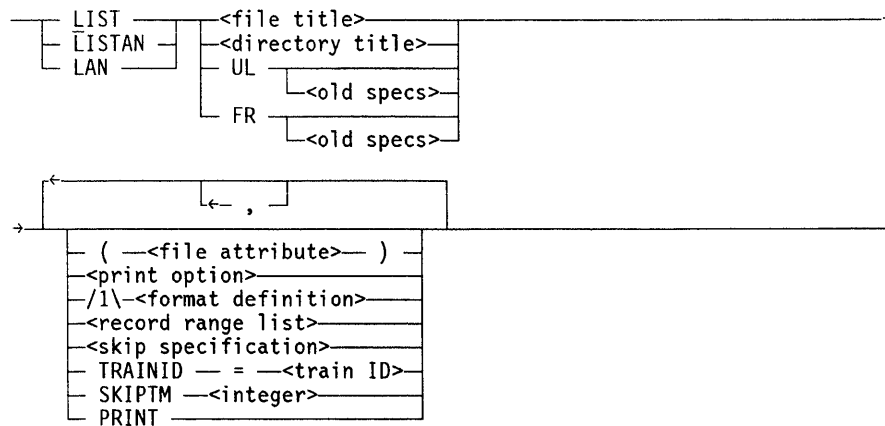
LIBMT produces a listing of a library tape in hexadecimal form. In addition, the listing identifies the position of tape marks.

Library maintenance tapes have a special structure. If you use the LIST command in DUMPALL to print a file on a library maintenance tape, the display does not correspond exactly to a display of the original disk file. If you use the COPY command in DUMPALL to copy a file from a library tape, the copied file does not have the structure of the original disk file, and the copy contains extra data.

LIST Command

You can use the LIST command to print or display records from a card, disk, or tape input file or from a file located on another host system.

Syntax



Explanation

LIST <file title>

Lists the file in EBCDIC format. If the INTMODE attribute value is not EBCDIC, DUMPALL converts all characters to EBCDIC representation for output.

By default, INTMODE is set to the value of the EXTMODE attribute of the file named in the LIST command. But you can explicitly set the values of INTMODE and EXTMODE that you want DUMPALL to use.

LISTAN <file title>

LAN <file title>

List the file in a particular format depending on the INTMODE attribute value of the file as follows:

INTMODE Value	Print Option
SINGLE	EBCDIC, Hexadecimal
HEX	Hexadecimal
EBCDIC	EBCDIC, Hexadecimal
ASCII	ASCII, Hexadecimal

By default, INTMODE is set to the value of the EXTMODE attribute of the file named in the LISTAN command. But you can explicitly set the values of INTMODE and EXTMODE that you want DUMPALL to use.

<directory title>

Causes all files in the specified disk pack directory to be listed.

UL

UL <old specs>

Specify an unlabeled file – that is, a file with a LABELTYPE value of OMITTED.

Specify UL only for tape files. For UL, DUMPALL assigns the tape a default MAXRECSIZE and a BLOCKSIZE value of 1500 words. If you specify file attribute values other than the default values, DUMPALL assigns those values. If the attribute values are incorrect, the list might be incomplete.

UL specifies that an input file has a KIND attribute value of TAPE and has no label. UL does not require an unlabeled tape; it merely specifies that the tape be treated as unlabeled. When you use UL, it is not necessary to specify a KIND attribute value of TAPE. DUMPALL ignores UL for disk and pack files.

When you use UL for an input file whose options do not include SKIPTM or MULTI, DUMPALL can read more than one physical tape. When DUMPALL reads a tape mark from one tape, it closes that volume and attempts to open the next volume. The system then displays the “NO FILE” RSVP message for the next volume. You must respond to the RSVP message with either the FR (Final Reel) or the UL (Unlabeled) system command.

When you use an input file specification that includes SKIPTM or MULTI, UL is equivalent to FR – that is, DUMPALL uses only one physical tape. In this case, when DUMPALL reads a tape mark for the tape, the operating system does not attempt a volume switch.

FR

FR <old specs>

Specify an unlabeled tape file that has only one reel, with reel switching suppressed – that is, the file has a LABELTYPE attribute value of OMITTEDEOF.

FR specifies that an input file has a KIND attribute value of TAPE and has no label. As in the case of UL, the tape used need not be an unlabeled tape. When you use FR, it is not necessary to specify a KIND value of TAPE.

The difference between the FR and the UL DUMPALL options is that when you use UL, DUMPALL reads a tape mark from the tape, closes the file, and does not generate a “NO FILE” RSVP message. Refer to “Handling Tape Files” later in this section.

(<file attribute>)

Causes DUMPALL to use the specified attributes when listing the file. If you specify the HOSTNAME file attribute, DUMPALL requests access to a file on a remote host system; in this case, Distributed Systems Services (DSS) requires that you also specify the KIND attribute. Refer to the *File Attributes Reference Manual* for more information about the KIND attribute.

DUMPALL Utility

Refer to “Understanding Structural File Attributes” earlier in this section for an explanation of the file structure, record size, block size, DEPENDENTSPECS attribute, and their effects on DUMPALL.

Refer to “Specifying Data or Character Set Translations” earlier in this section for an explanation of the effects of the EXTMODE and INTMODE file attributes on DUMPALL.

<print option>

Specifies the format to be used when the file is listed. You can specify one or more print options.

If you specify a single print option and it is EBCDIC or ASCII, DUMPALL lists the record as one unit in the format specified in the print option. If you specify a single print option and it is REAL, DECIMAL, HEXADECIMAL, or OCTAL, DUMPALL lists the record in word-sized chunks.

If you specify more than one print option, DUMPALL lists the record in word-sized chunks, one format below the other on a page. Because DECIMAL is a subset of REAL, if you specify both, only REAL formatting takes place.

<format definition>

Describes one or more fields of the record and their formats. If you do not specify any fields or formats in a print command, DUMPALL prints each record in full. If you specify any fields or formats in a print command, DUMPALL prints only the parts of the record you request.

<record range list>

<skip specification>

Cause only the specified records of the file to be listed.

TRAINID = <train ID>

Specifies the train to be used on the printer when the file is listed.

The train ID can be any valid train ID. Refer to the *File Attributes Reference Manual* for a description of the TRAINID attribute and for a list of valid train IDs.

SKIPTM <integer>

Causes DUMPALL to skip past the number of tape marks specified, bypassing any records encountered between tape marks. Use SKIPTM to position an input tape at the first record of a file to be read. You can use SKIPTM only with an input file that you specify as UL or FR.

Refer to “Description of Tape Formats” later in this section for an explanation of where tape marks appear on a tape.

PRINT

Used in interactive mode to cause the file to be listed on the printer instead of the terminal.

Example 1

Example 1 lists all files in the directory *SOU/999. The output goes to the printer.

```
LIST *SOU/999/= PRINT
```

Example 2

Example 2 lists an unlabeled EBCDIC tape file with 80-byte records, 10 records per block, in EBCDIC and hexadecimal formats after skipping two tape marks:

```
LAN UL EBC 80 800 CHAR SKIPTM 2
```

Example 3

Example 3 lists records 4, 5, 6, 7, 8, 13, 14, 15, and 100 through the end of the file in decimal format from a tape file named X:

```
L X(KIND=TAPE) DEC REC 4 THRU 8, SKIP 4 3, REC 100 THRU END
```

Example 4

Example 4 shows both a LIST command and the output from that command. The first number on the first output record is the record number (one-relative) of the record being listed. The letter E that follows the number specifies that the print option is EBCDIC. A vertical bar (|) separates the record number from the text; one space always follows the bar. A number appears at the end of the last line to indicate the total size of the record in print option units.

This example converts all characters in the file FILE1 to EBCDIC for printing. In this case, FILE1 has data.

```
LIST FILE1
```

```
1E| ABCDEFGHIJKLMNOPQRSTUVWXYZ013456789...36
```

Example 5

Example 5 lists record 5 of the file MYFILE in EBCDIC format. The second line begins with byte 120 of the record. Bytes are the units used to specify offset when the print option value is EBCDIC. The record is 143 bytes long. A number in parentheses appears at the beginning of the second and all following lines to indicate the offset (zero-relative), in print option units, of the data listed on that line.

DUMPALL Utility

```
LIST MYFILE EBCDIC REC 5
```

```
5E| THIS IS THE DATA IN THE RECORD  
(00000120)E| MORE OF THE SAME RECORD...143
```

Example 6

Example 6 lists record 17 of the file ABC in hexadecimal format. The record is 60 digits (5 words) long. The second line begins with digit 48 of the record.

```
LIST ABC HEX REC 17
```

```
17H| 0123456789AB | 00000000000000 | 12345AFE5400 | 333333333333 |  
(0000048)H| 0000000000DFD | ...60
```

Example 7

Example 7 lists record 120 of the file XYZ in decimal and EBCDIC formats. The record is 5 words (30 bytes) long. The ALPHA on line 3 indicates that the word has bit 47 ON and is not considered to be a numeric field. The REAL on line 3 indicates that the word is not an integer but a real number.

```
LIST XYZ DECIMAL EBCDIC REC 120
```

```
120D|          0 |          1 |          2 |  
E| ?????? | ?????? | ?????? |  
-----  
(0000003)D| ALPHA      | REAL      | ...5  
(0000018)E| RECORD     | 120      | ...30  
-----
```

Example 8

Example 8 lists the contents of the file T/FILETHREE, which resides on one or more labeled tapes.

```
LIST T/FILETHREE (KIND=TAPE)
```

Example 9

Example 9 lists the third file of a multifile tape, which either is unlabeled or is to be treated as an unlabeled tape. Because SKIPTM is specified, only one input tape is used.

```
LIST UL SKIPTM 2
```

Considerations for Use

The LIST command can handle INDEXED, KEYEDIOII, and PLIISAM files. Since DUMPALL uses the KEYEDIOSUPPORT, the KEYEDIOIISUPPORT, and the PLISUPPORT libraries respectively to access these files, deleted records are not listed.

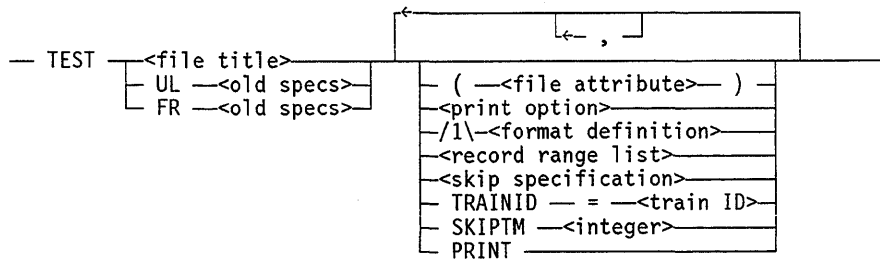
If the KEYEDIOSUPPORT, KEYEDIOIISUPPORT, or PLISUPPORT libraries are not installed with the SL (Support Library) system command, when an INDEXED, KEYEDIOII or PLIISAM file is accessed by DUMPALL, DUMPALL terminates with the following message:

```
FILE <filename> OPEN ERROR: SUPPORT LIBRARY UNAVAILABLE
```

TEST Command

The TEST command tests a file for parity errors. DUMPALL reads the file and prints only those records that contain parity errors.

Syntax



Explanation

<file title>

Specifies the file to be tested.

UL <old specs>

Specify an unlabeled file—that is, a file with a LABELTYPE value of OMITTED.

Specify UL only for tape files. For UL, DUMPALL assigns the tape default MAXRECSIZE and BLOCKSIZE values of 1500 words. If you specify file attribute values other than the default values, DUMPALL assigns those values. If the attribute values are incorrect, the list might be incomplete.

UL specifies that an input file has a KIND attribute value of TAPE and has no label. UL does not require an unlabeled tape; it merely specifies that the tape be treated as unlabeled. When you use UL, it is not necessary to specify a KIND attribute value of TAPE. DUMPALL ignores UL for disk and pack files.

When you use UL for an input file whose options do not include SKIPTM or MULTI, DUMPALL can read more than one physical tape. When DUMPALL reads a tape mark from one tape, it closes that volume and attempts to open the next volume. The system then displays the “NO FILE” RSVP message for the next volume. You must respond to the RSVP message with either the FR (Final Reel) or the UL (Unlabeled) system command.

When you use an input file specification that includes SKIPTM or MULTI, UL is equivalent to FR—that is, DUMPALL uses only one physical tape. In this case, when DUMPALL reads a tape mark for the tape, the operating system does not attempt a volume switch.

FR <old specs>

Specify an unlabeled tape file that has only one reel, with reel switching suppressed – that is, the file has a LABELTYPE attribute value of OMITTEDEF.

FR specifies that an input file has a KIND attribute value of TAPE and has no label. As in the case of UL, the tape used need not be an unlabeled tape. When you use FR, it is not necessary to specify a KIND value of TAPE.

The difference between the FR and the UL DUMPALL options is that when you use UL, DUMPALL reads a tape mark from the tape, closes the file, and does not generate a “NO FILE” RSVP message. Refer to “Handling Tape Files” later in this section.

(<file attribute>)

Causes DUMPALL to use the specified attributes when listing the file. If you specify the HOSTNAME file attribute, DUMPALL requests access to a file on a remote host system; in this case, distributed systems services (DSS) requires that you also specify the KIND attribute. Refer to the *File Attributes Reference Manual* for more information about the KIND attribute.

Refer to “Understanding Structural File Attributes” earlier in this section for an explanation of the file structure, record size, block size, DEPENDENTSPECS attribute, and their effects on DUMPALL.

Refer to “Specifying Data or Character Set Translations” earlier in this section for an explanation of the effects of the EXTMODE and INTMODE file attributes on DUMPALL.

<print option>

Specifies the format to be used to print the records. You can specify one or more print options.

If you specify a single print option and it is EBCDIC or ASCII, DUMPALL lists the record as one unit in the format specified in the print option. If you specify a single print option and it is REAL, DECIMAL, HEXADECIMAL, or OCTAL, DUMPALL lists the record in groups the size of words.

If you specify multiple print options, DUMPALL lists the record in groups the size of words, one format below the other on a page. Because DECIMAL is a subset of REAL, if you specify both, only REAL formatting takes place.

<format definition>

Describes a particular field within a record and its format. DUMPALL prints only the specified field of a record if a REAL error occurs.

DUMPALL Utility

<record range list>
<skip specification>

Cause only the specified range of the file to be tested.

SKIPTM <integer>

Causes the specified number of tape marks to be skipped before the file is tested.

PRINT

Used in interactive mode to send the output to the printer instead of the remote terminal.

TRAINID = <train ID>

Tests the file and writes the records that contain parity errors to the printer using the specified train. DUMPALL generates a separate print file for each use of the train ID option.

The train ID can be any valid train ID. Refer to the *File Attributes Reference Manual* for a description of the TRAINID attribute and for a list of valid train IDs.

The rules regarding tape usage for the TEST command are the same as those for the LIST command. The purpose of the command is to test a file for parity errors. DUMPALL prints only those records that have parity errors.

Example

The following example tests the file F1 for parity errors. In the output, IOCW is the I/O control word issued to the I/O subsystem, IORD is the I/O result descriptor returned by the I/O subsystem, IOET is the direct I/O error type, and BLOCK is the block number in the file where a parity error occurred. Refer to the *File Attributes Reference Manual* regarding the use of IOCW, IORD, and the IOERRORTYPE file attribute.

```
TEST F1
```

```
IO EXCEPTION IOCW=NNNNNNNNNNN IORD=MMMMMMMMMMM IOET=XXX  
BLOCK=YYYY
```

Interactive List Routine Commands

When DUMPALL is run in interactive mode, you can initiate all the previously described commands, one after the other, with the exception of the LIBMT command. In addition, while in interactive mode, you can set up an interactive list routine by using the OPEN command or by using a standard LIST command to open a file.

During a list routine, you can use the interactive list routine commands to switch back and forth between pages of a record display, to change the display mode, and to reposition the listing to any record in the file.

You can use only the interactive list routine commands during the list routine. The list routine must end before you can use any standard DUMPALL command. You can end the list routine with the interactive QUIT command. The following are the interactive list routine commands:

- AGAIN
- FILE or ATTRIBUTES
- CONTINUE
- LIST
- MODE
- NEXT
- OPEN
- PREVIOUS
- PRINT
- QUIT
- RECORD
- SKIP

Interactive AGAIN Command

The interactive AGAIN command relists the current record on the screen.

Syntax

— AGAIN —————|

Interactive FILE or ATTRIBUTES Command

The interactive FILE or ATTRIBUTES command displays the attributes of the file that is being listed. This command can be entered at any time during the listing. ATTRIBUTES and FILE are synonyms.

Syntax

— FILE —————|
 └─ ATTRIBUTES ─┘ └─ PRINT ─┘

Explanation

PRINT

Lists the attributes of the file on the printer.

Interactive CONTINUE Command

The interactive CONTINUE command displays the next record of the file. When the last page of a record display is listed on the screen, the word CONT is displayed in the upper left corner of the screen. You can either transmit the CONT command, or you can enter another interactive command.

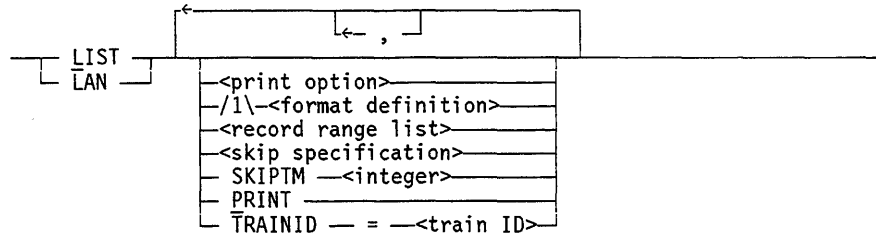
Syntax

— CONTINUE —————|

Interactive LIST Command

The interactive LIST command lists the file opened by the preceding OPEN command. This command differs from the standard LIST command in that you do not specify a file name.

Syntax



Explanation

LIST

Lists the file in the format specified by the INTMODE attribute of the file.

LAN

Lists the file in a print option depending on the INTMODE attribute value of the file as follows:

INTMODE	Print Option
SINGLE	EBCDIC, Hex
HEX	Hex
EBCDIC	EBCDIC, Hex
ASCII	ASCII, Hex

<print option>

Specifies the print option to be used.

If you specify a single print option and it is EBCDIC or ASCII, DUMPALL lists the record as one unit in the format specified in the print option. If you specify a single print option and it is REAL, DECIMAL, HEXADECIMAL, or OCTAL, DUMPALL lists the record in word-sized chunks.

If you specify more than one print option, DUMPALL lists the record in word-sized chunks, one format below the other on a page. Because DECIMAL is a subset of REAL, if you specify both, only REAL formatting takes place.

DUMPALL Utility

<format definition>

Describes one or more particular fields within a record and their formats. DUMPALL lists only the specified fields.

<record range list>

<skip specification>

List the part of the file specified by the record range list or the skip specification.

SKIPTM <integer>

Causes the specified number of tape marks to be skipped before the file is listed.

PRINT

Lists the file on the printer.

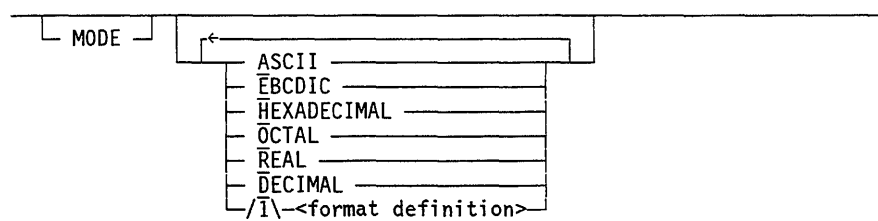
TRAINID = <train ID>

Lists the file on the printer using the specified train.

Interactive MODE Command

The MODE command lists the current record in a specific mode or format. You can also use the MODE command to alter the mode (print option) or format in which the remaining records are to be listed.

Syntax



Explanation

MODE

Lists the current record in the default mode.

MODE ASCII
MODE EBCDIC
MODE HEXADECIMAL
MODE OCTAL
MODE REAL
MODE DECIMAL

List all subsequent records with the new print option or format until another MODE command is entered. If you omit the prefix MODE, DUMPALL lists only the current record in the new mode, and formatting reverts to the old mode when DUMPALL reaches the next record. The various print options are explained under “Print Options” in “Input to the DUMPALL Utility” earlier in this section.

<format definition>

Describes one or more fields within a record and their formats or print options. DUMPALL lists only the fields defined in the format definition.

Interactive NEXT Command

The interactive NEXT command displays the next page of a record. When DUMPALL lists any page of a record other than the last one on the screen, the word NEXT appears in the upper left corner of the screen. You can retransmit NEXT or enter another interactive command.

Syntax

— NEXT —————|

Interactive OPEN Command

The interactive OPEN command opens a file for listing and initiates the interactive list routine. A list of the file attributes is given in response to the command. After the OPEN command completes, you can use only the interactive list routine commands. The list routine must end— with a QUIT command, for example— before you can enter the standard DUMPALL commands.

Syntax

— OPEN —<file title>—————|
 | UL —<old specs>——| (—<file attribute>—) |
 | FR —<old specs>——|

Explanation

OPEN <file title>

Initiates an interactive list routine session for the specified file. After the file attributes are listed, you can enter any interactive list routine command.

UL <old specs>

Specifies an unlabeled file. Use old specs to assign values to the INTMODE, MAXRECSIZE, and BLOCKSIZE attributes for the file.

FR <old specs>

Specifies an unlabeled tape file that has only one reel, with reel switching suppressed— that is, the file has a LABEL attribute value of OMITTEDEOF. You can use old specs to assign values to the INTMODE, MAXRECSIZE, and BLOCKSIZE attributes for the file.

(<file attribute>)

Causes DUMPALL to use the specified file attributes when it opens the file.

Interactive PREVIOUS Command

The interactive PREVIOUS command displays the previous page of the record on the screen. If the current page is the first or only page of the record, the PREVIOUS command displays that page again.

Syntax

```
— PREVIOUS —————|
```

Interactive PRINT Command

The interactive PRINT command lists the current record on the printer in the specified mode. If no mode information is specified in the PRINT command, DUMPALL uses the current default mode.

Syntax

```
— PRINT ———|
|
| ASCII ———|
| EBCDIC ———|
| HEXADECIMAL ———|
| OCTAL ———|
| REAL ———|
| DECIMAL ———|
| /I\<format definition>|
```

Explanation

```
PRINT ASCII
PRINT EBCDIC
PRINT HEXADECIMAL
PRINT OCTAL
PRINT REAL
PRINT DECIMAL
```

Print the file in the specified mode. The specific modes are described under “Print Option” earlier in this section.

<format definition>

Describes a particular field within a record and its format. DUMPALL lists only the specified field.

Interactive QUIT Command

The interactive QUIT command has three uses:

- While displaying records, the list routine displays the word CONT in the upper left corner of the screen. If you enter QUIT in response, the display of records ends, and the list routine asks for your next interactive list routine command.
- If you enter QUIT while the list routine is waiting for your next interactive list routine command, the list routine terminates giving the reply "END OF LIST ROUTINE." Then DUMPALL asks you to enter your next standard DUMPALL command.
- If you enter QUIT while DUMPALL is waiting for your next standard DUMPALL command, DUMPALL goes to end of task (EOT).

Syntax

— QUIT _____|

Interactive RECORD Command

The interactive RECORD command repositions a file that is being listed to the specified record in that file. A record number beyond the end of the file causes termination of the list routine. Records are 1-relative – that is, the first record is record 1. The values specified for the record range list and skip specification parameters in the LIST command override the interactive SKIP and RECORD commands. Therefore, after you enter an interactive SKIP command, the specifications in the LIST command can cause subsequent repositioning of the file or termination of the list routine.

Syntax

— RECORD —<number>_____|

Example

The following example repositions the file to the fourth record:

```
RECORD 4
```

Interactive SKIP Command

The interactive SKIP command repositions the file that is being listed in interactive mode. The operating system can position the file either forward or backward, but positioning beyond the end of the file causes termination of the list routine. The values that you specify for the record range list and skip specification parameters in the LIST command override the interactive SKIP and RECORD commands. Therefore, after you enter an interactive SKIP command, the specifications in the LIST command can cause subsequent repositioning of the file or termination of the list routine.

Syntax

— SKIP $\left[\begin{array}{c} + \\ - \end{array} \right] \langle \text{number} \rangle$

Example

The following example repositions the file backward three records from the current record:

```
SKIP - 3
```

Running the DUMPALL Utility

You can execute the DUMPALL Utility by using the RUN statement or through the Menu-Assisted Resource Control (MARC) System Utilities screen.

If an error occurs while DUMPALL is either reading from or writing to a file, DUMPALL sets the TASKVALUE attribute to a value of 1. If no error occurs, TASKVALUE has a value of 0. WFL jobs can detect whether a DUMPALL task completed successfully by attaching a task variable to the DUMPALL execution. Following the completion of the process, the WFL job can check to see if DUMPALL was successful or not by testing the value of the TASKVALUE attribute of the task variable.

Through the RUN statement, you can execute DUMPALL in parameter mode, card mode, or interactive mode.

Parameter Mode

In parameter mode, specify the input to DUMPALL at the same time that the program is executed. You can enter several commands separated by semicolons (;) in the same statement.

To run DUMPALL in parameter mode, use the following syntax:

```
— RUN        SYSTEM/DUMPALL — ( — " —<command list>— " — ) —|  
      | $ |
```

Explanation

\$

Must be used when running DUMPALL through Command and Edit (CANDE).

<command list>

Lists the commands DUMPALL is to use. The list of commands must appear within double quotation marks ("). The command list consists of one or more DUMPALL commands separated by semicolons (;). Work Flow Language (WFL) permits a maximum of 256 characters to be entered in the command list. Refer to the descriptions in "Standard Commands" earlier in this section.

Example 1

Example 1 produces a listing of the file TESTFILE.

```
RUN $SYSTEM/DUMPALL("LIST TESTFILE");
```

Example 2

Example 2 produces a listing of the files TESTFILE and NEWFILE.

```
RUN $SYSTEM/DUMPALL("LIST TESTFILE; LIST NEWFILE");
```

Card Mode

In card mode, you put one or more DUMPALL commands separated by semicolons (;) onto one or more card images. DUMPALL reads these records and acts on your commands. DUMPALL uses only columns 1 through 72 of each card image. You can use the remainder of the record for a sequence number or comments. If you put a percent sign (%) on a card image, DUMPALL ignores all the text after the percent sign. You can use percent signs to put comments into your DUMPALL input card file.

To run DUMPALL from a WFL job in card mode, put together your card deck or job file in the following form:

```
RUN SYSTEM/DUMPALL ("CARD");
DATA

      card images with DUMPALL commands

?
```

The card or record with the question mark (?) in column one terminates the card images to be read by DUMPALL.

You can also store your DUMPALL commands in a disk file, such as a CANDE TEXT file or ALGOLSYMBOL file. To input commands from a disk file to DUMPALL, use the following WFL or CANDE statement:

```
RUN *SYSTEM/DUMPALL ("CARD");
FILE CARD = <file name> ON <family name>;
```

Example

In the following example, DUMPALL executes the commands contained in the file MYCARD on the pack MYPACK:

```
RUN SYSTEM/DUMPALL("CARD"); FILE CARD(TITLE=MYCARD ON MYPACK);
```

Interactive Mode

In interactive mode, DUMPALL commands are entered and processed one at a time from a remote device. You must enter the commands in all uppercase letters. You can use all commands interactively except for the LIBMT command. Additional commands that you can use only during an interactive DUMPALL list routine are included in "Interactive List Routine Commands" earlier in this section.

DUMPALL Utility

To run DUMPALL in interactive mode, use the following syntax:

```
-- RUN [ $ ] SYSTEM/DUMPALL ( "INTER" )
```

Explanation

\$

Must be used when running DUMPALL through CANDE.

("INTER")

Specifies that DUMPALL is to be run interactively.

Example

The following example initiates DUMPALL in interactive mode through CANDE:

```
RUN $SYSTEM/DUMPALL("INTER")
```

Using the MARC Interface

Through the MARC interface, you can execute SYSTEM/DUMPALL by selecting options from menus. To run DUMPALL, enter *DALL* in the Choice field of the System Utilities screen. The DUMPALL Choice Menu appears. Enter one of the following options in the Choice field:

Command	Function
COPY	Displays the Copy option menu, which enables you to copy some or all of a file with optional file changes. Refer to "COPY Command" earlier in this section for more information.
CAT	Displays the Concatenate INPUT file option menu, which enables you to copy multiple files concatenated into a new or an old file. Refer to "CAT Command" earlier in this section for more information.
LIST	Displays the List option menu, which enables you to list some or all of a file or directory in one or more formats. Refer to "LIST Command" earlier in this section for more information.
HEXDSK	Displays the List Disk Sectors in Hex option menu, which enables you to list disk sectors in EBCDIC or hexadecimal format. Refer to "HEXDSK Command" earlier in this section for more information.
INTER	Runs SYSTEM/DUMPALL in interactive mode. Refer to "Interactive Mode" earlier in this section for more information.

For menu-specific or field-specific information on any of the DUMPALL option menus, consult the help text associated with that menu or field.

To return to the MARC System Utilities screen, enter *BYE* in the Action field.

Controlling I/O Exceptions

DUMPALL gives you an opportunity to react to I/O exceptions when it executes a LIST, COPY, CAT, or TEST command. If DUMPALL is in interactive mode, you can respond through the remote file that is open at the remote device. If DUMPALL is not in interactive mode, you can respond by using the AX system command. Depending on the type of exception and the command being executed, DUMPALL displays an appropriate set of options.

When a response is required and DUMPALL is not in interactive mode, DUMPALL displays the nature of the error and waits for you to return AX input. The DUMPALL task appears in the waiting mix entry list as shown in the following example:

```
6757/7118 5Ø *SYSTEM/DUMPALL
ACCEPT:PLEASE ENTER SHOW, CONT, OR QUIT
```

The following is a valid response at the operator display terminal (ODT):

```
7118 AX CONT
```

Example 1

If the Y system command is used on the waiting entry, the RSVP message portion of the response contains the entire message. Example 1 shows an example input and response:

```
7118 Y

STATUS OF TASK 7118 AT 9:34:58
PRIORITY = 5Ø
ORIGINATION: LSN 5Ø9
USERCODE: USER
STACK STATE: WAITING ON AN EVENT
PROGRAM NAME: *SYSTEM/DUMPALL
RSVP: ACCEPT:PLEASE ENTER SHOW, OR QUIT.
ERROR ON INPUT FILE: (USER)A/FILE ON MYPACK
PARITY ERROR (RECORD = 9675 BLOCK = 387 FILE ;1)
SHOW WILL PRINT THE CONTENTS OF THE BUFFER AND CONTINUE.
CONT WILL SKIP THE RECORD AND CONTINUE.
QUIT WILL TERMINATE THE CURRENT COMMAND.
```

Example 2

If DUMPALL is executed from a data comm terminal, the same message as shown in Example 1 is displayed on the terminal. Example 2 shows this kind of message display:

```
7118 ACCEPT:PLEASE ENTER SHOW, CONT, OR QUIT
ERROR ON INPUT FILE: (USER)A/FILE ON MYPACK
PARITY ERROR (RECORD = 9675 BLOCK = 387 FILE ;1)
SHOW WILL PRINT THE CONTENTS OF THE BUFFER AND CONTINUE.
CONT WILL SKIP THE RECORD AND CONTINUE.
QUIT WILL TERMINATE THE CURRENT COMMAND.
```

Example 3

Example 3 shows a valid response at the data comm terminal:

```
? AX CONT
```

Example 4

If DUMPALL is running in interactive mode, the system displays the same message as shown in Example 1 at the remote device. The task then waits for your response. The display and response are presented and received by means of the remote file as shown in Example 4:

```
PLEASE ENTER SHOW, CONT, OR QUIT
ERROR ON INPUT FILE: (USER)A/FILE ON MYPACK
PARITY ERROR (RECORD = 9675 BLOCK = 387 FILE ;1)
SHOW WILL PRINT THE CONTENTS OF THE BUFFER AND CONTINUE.
CONT WILL SKIP THE RECORD AND CONTINUE.
QUIT WILL TERMINATE THE CURRENT COMMAND.
```

Example 5

Example 5 shows a valid response at the remote device:

```
CONT
```

Depending on the command, one or more of the following options are allowed:

Option	Purpose
SHOW	Prints the contents of the buffer. DUMPALL continues processing the command.
CONT	Skips this record and continues processing the command.
NEXT	Switches to the next reel and continues processing the command. This option is provided only on files whose KIND attribute has the value TAPE, where an END OF TAPE exception has occurred.
QUIT	Terminates the current command. If an output file is being created, it is saved. If DUMPALL is in interactive mode, the next command is requested. If it is not in interactive mode, DUMPALL terminates.

The following are recommendations regarding exceptions:

Exceptions	Recommendations
For LIST and TEST commands	PARITY, DATA ERROR, and END OF TAPE exceptions defer to you for response. All other exceptions display an error message, set the DUMPALL TASKVALUE attribute to an error value (1), and terminate the current command. If it is not in interactive mode, DUMPALL terminates. The available options for an END OF TAPE exception are HELP, NEXT, and QUIT. In all other cases, the options are HELP, SHOW, CONT, and QUIT.
For input files on CAT and COPY commands	PARITY, DATA ERROR and END OF TAPE exceptions defer to you for direction. All other exceptions except SHORTBLOCK display an error message, set the DUMPALL TASKVALUE attribute to an error value (1), and terminate the current command. If it is not in interactive mode, DUMPALL terminates. The available options for an END OF TAPE exception are HELP, NEXT, and QUIT. In all other cases, the options are HELP, CONT, and QUIT.
For input files on CAT, COPY, LIST, and TEST commands	<p>A SHORTBLOCK exception occurs in certain cases when the size of a data block read from an input tape is less than the BLOCKSIZE of the file. The SHORTBLOCK exception displays an error message. You also see messages about the options you can take, such as SHOW, CONT, and QUIT.</p> <p>For example, if you enter the command COPY UL E 95 380 to NEWFILE, and the tape file contains 378 characters per block, you receive a SHORTBLOCK error. This message occurs because 378 is 2 bytes less than the 380 specified in the command.</p>
For output files on CAT and COPY commands	All exceptions defer to you. The available options for an END OF TAPE exception are HELP, NEXT, and QUIT. In all other cases, the options are HELP, CONT, and QUIT.

Input to the DUMPALL Utility

The following text describes the basic constructs, range lists, and format definitions used in DUMPALL commands.

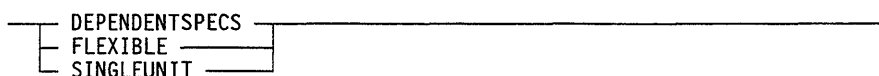
Basic DUMPALL Constructs

The following items commonly appear as syntactic variables in the syntax diagrams featured in this section.

<alphanumeric character>

Any one of the characters A through Z or 0 through 9.

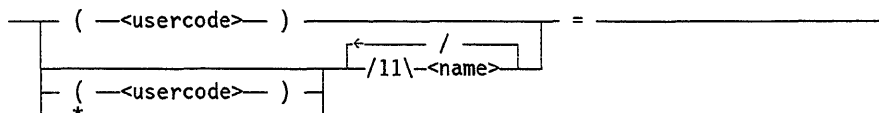
<Boolean-valued attribute>



<digit>

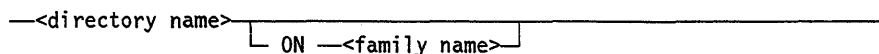
Any one of the decimal digits 0 through 9, inclusive.

<directory name>



Specifies the name of a directory.

<directory title>

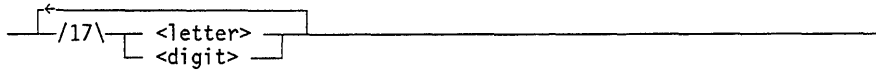


Specifies a particular directory.

<EBCDIC string character>

Any one of the 256 EBCDIC characters except the double quotation mark (").

<family name>

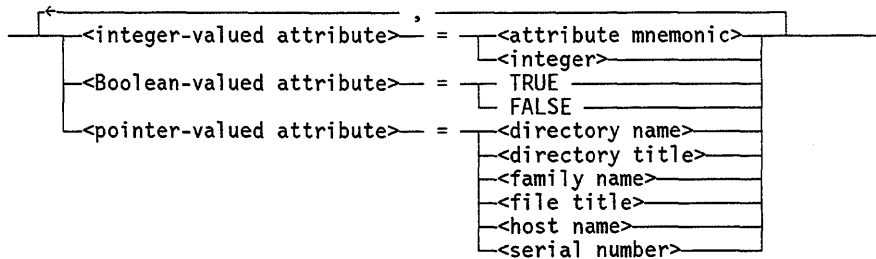


Identifies a disk family.

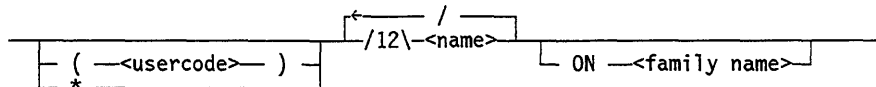
A family is a logical means of grouping mass storage devices together so that they function as one logical unit. Specification of the family name indicates a named native-mode disk pack, and any continuation packs, whose name is the specified family name. A file with a KIND attribute value of DISK or PACK refers, by default, to the family with the family name DISK.

The following diagrams identify the file attributes that can be assigned specific values for certain commands. Refer to the *File Attributes Reference Manual* for explanations of these attributes.

<file attribute assignment>



<file title>



Specifies a particular file. File titles containing the suffix ON <family name> can be used to specify a family other than the default family (DISK).

<host name>



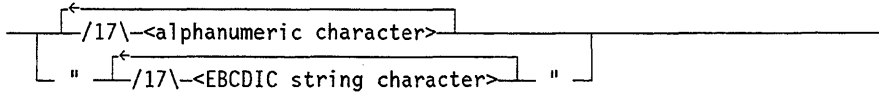
Identifies a host computer system.

<hyphen>

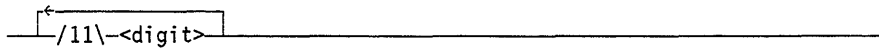
The single hyphen (-) character.

DUMPALL Utility

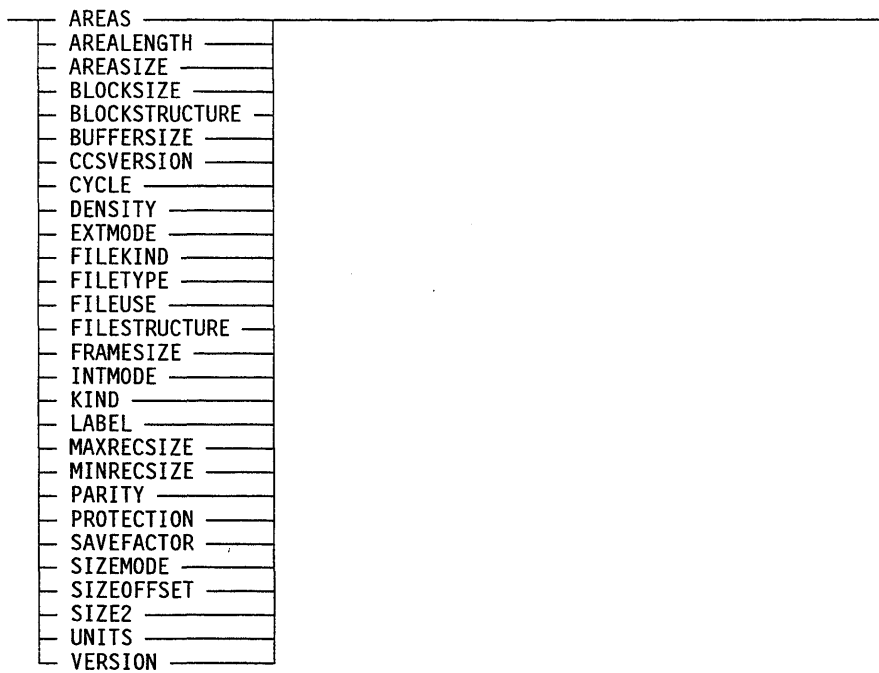
<identifier>



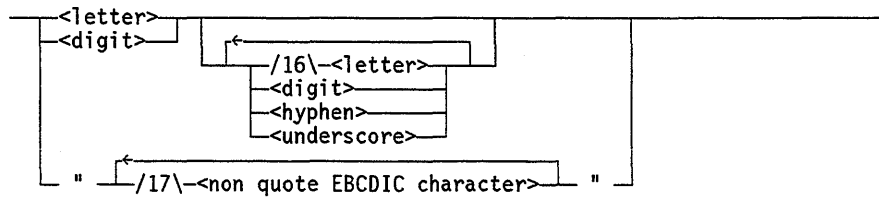
<integer>



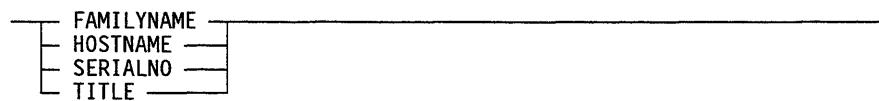
<integer-valued attribute>



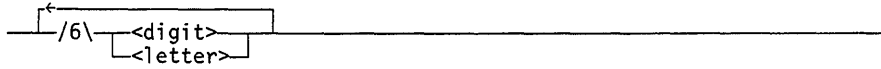
<name>



<pointer-valued attribute>



<serial number>



Identifies a tape or disk volume.

<underscore>

The single underscore (_) character.

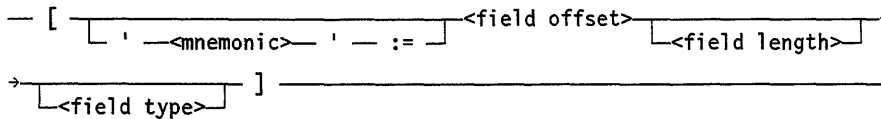
<usercode>



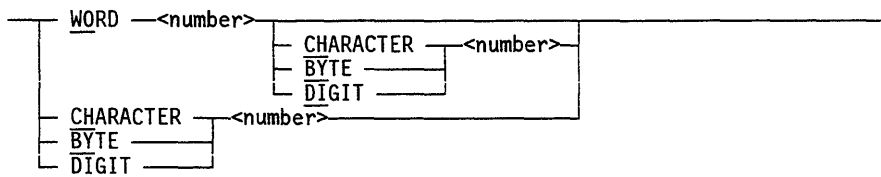
Field Definition

A field definition describes a field within a record and its format. It is used alone or with other field definitions to form a format definition. You can use fields and formats in various print commands to cause DUMPALL to process only the specified fields from each record and to specify how each field is to be printed.

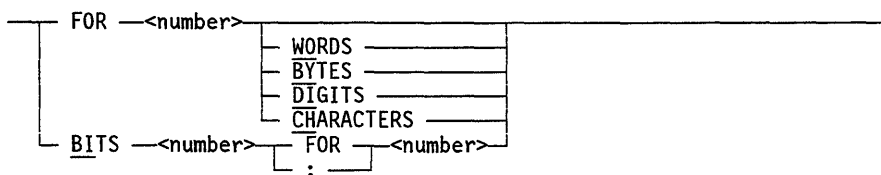
<field definition>



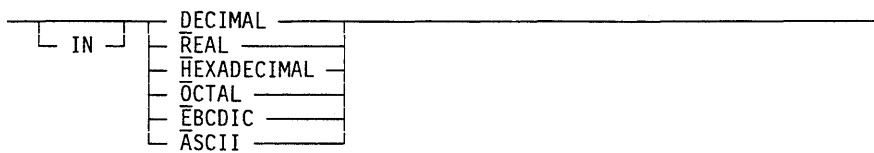
<field offset>



<field length>



<field type>



Explanation

'<mnemonic>' :=

Names the field. The mnemonic is entered into the defines file and can be used in subsequent format definitions to represent the field mnemonic. The mnemonic can be any valid identifier from 1 to 63 characters in length. If the field definition is part of a define command, this option must be present. Refer to the DEFINE command description in this section.

<field offset>

Specifies the offset of the field within the record. This offset can be specified in words, 8-bit characters or bytes, 4-bit digits, words and characters, words and bytes, or words and digits. WORD specifies 48-bit or 6-byte units. If you specify words and characters or words and bytes, the number of characters or bytes must be 5 or less. If you specify words and digits, the number of digits must be 11 or less. CHARACTER and BYTE specify 8-bit units. DIGIT specifies 4-bit units. The offset value is zero relative. The first word, character, byte, or digit of a record is at offset 0, the second is at offset 1, and so forth.

<field length>

Specifies the length of the field. If you do not specify the units for the field length, such as words or bytes, DUMPALL uses the units that you specified for the field offset. If you do not specify a field length, DUMPALL assumes the length of the field is 1.

<field type>

Specifies the type of formatting to be used. If you do not specify a field type, DUMPALL determines the type from the field length and units according to Table 3-2.

Table 3-2. Default Field Type

Units	Length	Type
WORDS	2 or less words	Real
WORDS	more than 2 words	Hex
BYTES		EBCDIC

continued

Table 3-2. Default Field Type (cont.)

Units	Length	Type
CHARACTERS		EBCDIC
DIGITS		Hex
BITS		Real

WORD <number>

Specifies the number of the word in the field where the field definition begins. Word numbers are relative to 0. The first word of a record is word 0, the second word is word 1, and so forth.

CHARACTER <number>**BYTE <number>****DIGIT <number>**

Following the **WORD <number>** option, these options specify the character, byte, or digit in the word where the field definition begins and is zero-relative. The first character, byte, or digit of a word is number 0, the second is number 1, and so forth. If no **WORD <number>** option appears, these options specify a particular character, byte, or digit where the field definition begins and is zero-relative. The first character, byte, or digit of a record is number 0, the second is number 1, and so forth. **CHARACTER** and **BYTE** define 8-bit (EBCDIC) offsets or lengths, and **DIGIT** defines 4-bit (HEXADECIMAL) offsets or lengths.

FOR <number> WORDS**FOR <number> BYTES****FOR <number> DIGITS****FOR <number> CHARACTERS**

Specify the field length in words, bytes, digits, or characters. If no unit (**WORDS**, **BYTES**, **DIGITS**, or **CHARACTERS**) is specified, the length is in the units used to specify the field offset.

If you specified the field offset in digits or in words and digits, you cannot specify the field length in bytes or characters.

BITS <number> FOR <number>

Specifies a partial word field with a length of one word. The first number specifies the left bit of the field and must be between 0 and 47, inclusive. The second number specifies the number of bits in the field and must range from 1 through 48, inclusive.

If you specify less than 48 bits, DUMPALL fills the missing leading bits with binary zeros to obtain a full 48-bit entity. If you specify the field type as **HEXADECIMAL**,

DUMPALL prints the field bit as 12 hexadecimal characters. If you specify the field type as EBCDIC or ASCII, DUMPALL prints the field bit as 6 characters. In the latter case, the leading binary zeros print as question marks (?).

DECIMAL

Specifies an integer field (COMP). If the field identified does not evaluate to an integer value, it is displayed as though REAL had been specified.

REAL

Specifies a floating-point field (COMP-2).

HEXADECIMAL

Specifies a 4-bit alphanumeric field (COMP-2).

OCTAL

Specifies a 3-bit field.

EBCDIC

Specifies an 8-bit alphanumeric field (DISPLAY). If you specified the field offset in digits or in words and digits, you cannot specify a field type of EBCDIC.

ASCII

Specifies a 7-bit alphanumeric field (ASCII). If you specified the field offset in digits or in words and digits, you cannot specify a field type of ASCII.

Example 1

Example 1 describes a field that contains word 3 of the record in real format. Because the field length is not specified, it defaults to 1 word – word in this example because the field offset is specified in words. The field type is not specified so, according to Table 3-1, a one-word field defaults to type real.

[WORD 3]

Example 2

Example 2 describes a field that contains the first five bytes of the record in EBCDIC format. Because the units for field length are not specified, they default to the units specified for the field offset, which in this example is bytes. The field type is not specified so, according to Table 3-1, a field of bytes defaults to the type EBCDIC.

[BYTE 0 FOR 5]

Example 3

Example 3 describes a field that contains bits 47 through 24 of word 4 in hexadecimal format. The field is entered in the defines file under the mnemonic FLD1.

```
[ 'FLD1' := WORD 4 BITS 47:24 IN HEX ]
```

Example 4

Example 4 describes a field that contains digits 72 through 75 of the record in hexadecimal format. The field offset of digit 72 is equivalent to the offset of word 6 because there are 12 digits per word. The field type is not specified, so with the units for the field length (digits), we determine from Table 3-1 that the type is hexadecimal.

```
[ WORD 6 FOR 4 DIGITS ]
```

Example 5

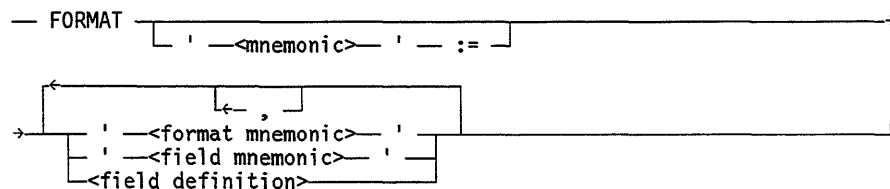
Example 5 describes a field that contains bytes 20 through 29 of the record in EBCDIC. The units for the field length are not specified, so they default to the units for the field offset, which in this example is bytes. The field type is not specified, so according to Table 3-1, a field of bytes defaults to the type EBCDIC.

```
[ BYTE 20 FOR 10 ]
```

Format Definition

A format definition is made up of one or more field definitions. A field definition describes a particular field within a record and the type or format of the data in that field. A format definition can be specified in various print commands to cause DUMPALL to process only the specified fields from each record and to specify how DUMPALL is to print each field. If you specify more than one field in a format, when DUMPALL prints data, DUMPALL separates the display for each field from that of the preceding field by 2 blank characters.

Syntax



Explanation

'<mnemonic>' :=

Defines the subsequent format and gives it the specified name. You can use the mnemonic in subsequent commands to represent its associated format definition. A mnemonic can be any valid identifier from 1 to 63 characters in length.

'<format mnemonic>'

'<field mnemonic>'

Specify a format or field that you have previously defined by using the mnemonic option of the format definition or field definition.

Example 1

Example 1 defines a format by using two field definitions. The first field describes the first 6 bytes of the record in EBCDIC format. (EBCDIC is the default format when the field length is specified in bytes.) The second field describes word 7 of the record in decimal format.

```
FORMAT [BYTE 0 FOR 6] [WORD 7 DECIMAL]
```

Example 2

Example 2 defines a format named FMT1, which is entered into the defines file. FMT1 consists of two fields. The first field is named FLD1 and is entered into the defines file. FLD1 describes bytes 20 through 24 of the record in EBCDIC format. The second field is not named. This field describes byte 50 of the record in EBCDIC format. (EBCDIC is the default format when the field length is specified in bytes.)

```
FORMAT 'FMT1':=['FLD1':=BYTE 20 FOR 5] [BYTE 50]
```

Example 3

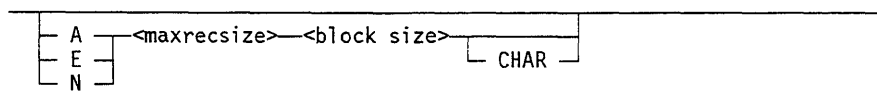
Example 3 defines a format named FMT2, which is entered into the defines file. FMT2 consists of the previously defined field FLD1 and the previously defined format FMT1.

```
FORMAT 'FMT2':='FLD1' 'FMT1'
```

Old Specs

Use old specs with unlabeled or nonstandard labeled tape files to assign values to the attributes INTMODE, MAXRECSIZE, and BLOCKSIZE. B3500 and B5500 standard labeled tape files are examples of nonstandard labeled tape files.

<old specs>



Explanation

The following list explains the available options:

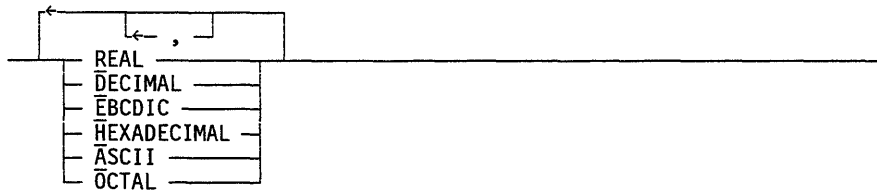
- The A indicates INTMODE value of ASCII format.
- The E indicates INTMODE value of EBCDIC format.
- The N indicates INTMODE value of nonstandard format.
- The maxrecsize is an integer specifying the MAXRECSIZE value of the file.
- The block size is an integer specifying the BLOCKSIZE value of the file.
- The CHAR specifies that maxrecsize and block size are in characters. The default is words.

The default setting of INTMODE is EBCDIC, and the default settings for MAXRECSIZE and BLOCKSIZE are 1500. You can also specify these file attributes directly by using the (<file attribute>) clause in DUMPALL commands. The default value for FRAMESIZE is 48—that is, words. You can change the default value by specifying a different value with the (<file attribute>) clause.

Print Option

Use the print option to indicate the format of the output from DUMPALL.

<print option>



DUMPALL Utility

Explanation

REAL

Prints output in single-precision, floating-point (COMP-4) format.

DECIMAL

Prints output in single-precision integers (COMP, COMP-1) format.

EBCDIC

Prints output in 8-bit EBCDIC (DISPLAY) format.

HEXADECIMAL

Prints output in 4-bit digit (COMP-2) format.

ASCII

Prints output in 7-bit ASCII format.

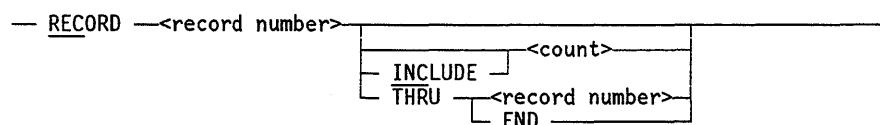
OCTAL

Prints output in 3-bit OCTAL format.

Record Range List

A record range list specifies a group of records to be processed. All record numbers used in the record range list are 1-relative; that is, the first record of a file is record 1.

<record range list>



Explanation

`RECORD <record number>`

Causes only the record given by the record number to be processed. Record number is an integer.

RECORD <record number> <count>
RECORD <record number> INCLUDE <count>

Cause the range to begin with the record specified by the record number and to include the number of records specified by count. Count is an integer.

RECORD <record number> THRU <record number>

Includes all records from the first record number through the second record number, inclusive. The second record number must be greater than the first record number.

RECORD <record number> THRU END

Includes all records beginning with the record number through the end of the file.

Example 1

The following range consists only of record 5:

```
RECORD 5
```

Example 2

The following range consists of records 5, 6, and 7:

```
RECORD 5 3
```

Example 3

The following range consists of records 5, 6, 7, 8, and 9:

```
RECORD 5 THRU 9
```

Example 4

The following range consists of records 5 through the end of the file:

```
RECORD 5 THRU END
```

Considerations for Use

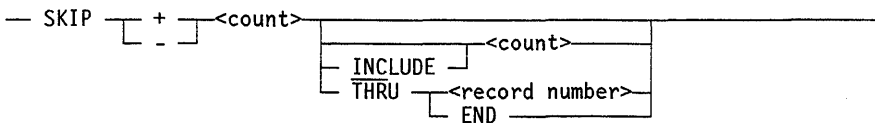
You can use a list that contains a mixture of record range lists and skip specifications. However, after you specify END in either one, you cannot specify any other record range lists or skip specifications.

Skip Specification

The skip specification specifies a group of records to be processed. The skip is relative to the *current record number*:

- The current record number is 1 if this is the first skip specification for this input file in the command and there are no preceding record range list specifications for this input file in the command.
- Otherwise, the current record number is the record number plus one of the last record of the preceding skip specification or record range list.

<skip specification>



Explanation

SKIP + <count>

Causes the file to be repositioned forward the number of records specified for the count variable. The range begins at the record whose record number is equal to the current record number plus the count value and ends at the end of the file.

SKIP - <count>

Causes the file to be repositioned backward the number of records specified for the count variable. The range begins with the record whose record number is equal to the current record number minus the count value and ends at the beginning of the file.

SKIP + <count> <count>

SKIP - <count> <count>

SKIP + <count> INCLUDE <count>

SKIP - <count> INCLUDE <count>

Cause the range to begin at the record whose record number is equal to the current record number plus the count value or to the current record number minus the count value and to include the number of records specified by the second count value.

The keyword INCLUDE is optional.

SKIP + <count> THRU <record number>

SKIP - <count> THRU <record number>

Cause the range to begin at the record whose record number is equal to the current record number plus the count value or equal to the current record number minus the count value and to end at the record specified by the record number.

SKIP + <count> THRU END
SKIP - <count> THRU END

Cause the range to include all the records from the record whose record number is equal to the current record number plus the count value or equal to the current record number minus the count value through the end of the file.

Examples

All the examples that follow assume that the preceding record range list consisted of records 5 through 10, so that the current record number is 11.

Example 1

Example 1 skips three records from the current record number and includes the three records following the skip. The new range includes records 14 through 16, inclusive.

```
SKIP + 3 3
```

Example 2

Example 2 skips two records from the current record number. The new range includes records 13 through 18, inclusive.

```
SKIP + 2 THRU 18
```

Example 3

Example 3 skips backward 3 records from the current record number. The new range begins with record 8 and ends at the end of the file.

```
SKIP - 3
```

Considerations for Use

You can use a list that contains a mixture of record range lists and skip specifications. However, after you specify END in either one, you cannot specify any other record range lists or skip specifications.

Handling Tape Files

This subsection explains concepts of labeled and unlabeled tapes, single-file tapes, and multifile tapes. DUMPALL can copy or print files stored on labeled or unlabeled tapes, and DUMPALL can copy files to labeled or unlabeled tapes. You can instruct DUMPALL to copy more than one file from the same input tape by specifying the MULTIFILE option for input files. You can instruct DUMPALL to create a multifile tape by specifying the MULTIFILE option for output files.

Description of Tape Formats

When you want to print or copy files located on an unlabeled tape or a tape with nonstandard labels, you must use the FR or UL specification for the input file. To use these tapes successfully with DUMPALL, you must understand the location of the tape marks and data records on the input tapes so that you can position the tape correctly.

The following terms are necessary to your understanding the DUMPALL syntax required to create and read tapes:

- Tape mark
- Unlabeled tape
- Labeled tape
- Nonstandard labeled tape

For a description of multivolume tapes, refer to the *A Series I/O Subsystem Programming Guide*.

Tape Marks

A tape mark is a special physical record that a system or program writes on a magnetic tape volume to delimit logical entities, such as files, from one another. By convention, the last valid data block or label record on a tape is followed by two tape marks in sequence.

Unlabeled Tapes

An unlabeled tape is a tape that has no label records. An unlabeled tape does not have a serial number.

A single-file unlabeled tape contains one file and has the following format:

```
data blocks for file
tape mark
tape mark
```

A multifile unlabeled tape is a tape that contains more than one file and no label records. A multifile unlabeled tape has the following format:

```

    data blocks for first file
  tape mark
    data blocks for second file
  tape mark
    .
    .
    .
    data blocks for last file
  tape mark
  tape mark

```

The PER MT system command entered at an ODT for an unlabeled tape produces a display similar to the following:

```

----- MT STATUS -----
      84 P      1600  1 UNLABELED

```

Labeled Tapes

A labeled tape is a tape that has label records. The label records contain information needed to locate a specific file on the tape. Each file on a labeled tape is preceded and followed by a set of label records. A tape mark is used to separate the label records from the records of a file on the tape. Refer to the *I/O Subsystem Programming Guide* for information on standard tape label formats.

Note that library maintenance tapes, which are labeled tapes, contain label information that is unique to library maintenance tapes. A library maintenance tape cannot be read as a labeled tape by DUMPALL, and a labeled tape as described in the preceding paragraph cannot be read by the library maintenance utility.

A single-file labeled tape contains one file and label records and has the following format:

```

    VOL label records for tape
    HDR label records for file
  tape mark
    data blocks for file
  tape mark
    EOF label records for file
  tape mark
  tape mark

```

DUMPALL Utility

A multifile labeled tape contains more than one file and label records. A multifile labeled tape has the following format:

```
VOL label records for tape
HDR label records for first file
tape mark
data blocks for first file
tape mark
EOF label records for first file
tape mark
HDR label records for second file
tape mark
data blocks for second file
tape mark
EOF label records for second file
tape mark
.
.
.
HDR label records for last file
tape mark
data blocks for last file
tape mark
EOF labels for last file
tape mark
tape mark
```

The PER MT system command entered at an ODT for a labeled tape produces a display similar to the following:

```
----- MT STATUS -----
84 P      [SERNUM] 1600 1 1:0 T/FILEONE
```

When it creates a label record for a multifile tape, the operating system uses only the first and last identifiers from the file title for each file written to the tape. For example, the file F/X/Y/Z/A is labeled F/A on the tape. Furthermore, if the first identifier in the title of each file on the tape is the same, a program need only set the internal file title of the program to the first and last identifiers (as it stands on the tape) to access a particular file on the tape. When the system opens a file, the system automatically positions the tape to read the first record of the required file. For example, if a labeled multifile tape contains the files F/A, F/B, F/C, and F/D, a program that opens an internal file with a KIND attribute value of TAPE and a TITLE attribute value of F/C accesses the records contained in file F/C on the tape.

When you create a labeled tape, you can use a single identifier for each file title. In this case, DUMPALL can read files other than the first file on the tape only if you specify the MULTIFILE option and the operator replies with an IL (Ignore Label) system command to the "NO FILE" RSVP message.

Some of the advantages of labeled tapes are the following:

- Labeled tapes have a serial number recorded in the VOL label that helps you and your installation keep track of tape volumes.
- Because the file name is stored in the HDR labels of a file, labeled tapes enable you to reference and locate tape files by name.
- Because the HDR and EOF labels record the values of file attributes such as record size, block size, and creation date, programs such as DUMPALL can automatically determine the proper attributes to be used to access a file stored on labeled tape volumes.
- The system automatically handles switching from one volume to the next whenever a read, write, or copy action reaches the end of a labeled tape volume.

Nonstandard Labeled Tapes

Some tapes created by systems other than A Series might have labels that are not fully compatible with the A Series system. For such tapes, you can use the PER MT system command to show the tapes as labeled or unlabeled. However, DUMPALL might not be able to retrieve files by name from such tapes, or DUMPALL might not be able to determine the correct record size and block size of the files on such tapes. To copy or print such tapes, you might need to use the UL or FR option to indicate that DUMPALL is to ignore the labels. If you use the UL or FR option, you might also need to use the SKIPTM option to position the tape past the tape mark that separates the label records from the data blocks for the files. Also, you must specify the MAXRECSIZE and BLOCKSIZE file attributes for the files on nonstandard labeled tapes. Refer to “Treating Labeled Tapes as Unlabeled Tapes” later in this section.

Output Files

You can control tape selection by specifying a serial number or list of serial numbers in the file attribute list associated with the output file. If the required tape is not available when the output file is opened, DUMPALL is suspended with the following message:

```
<mix number> <file name> REQUIRES MT [<serial number>] #1
```

If a serial number is not specified and tapes are required to run the DUMPALL utility, the operating system performs one of the following operations:

- If the system option SERIALNUMBER is not set, the operating system selects any available scratch tape. If a scratch tape is not available, the operating system suspends DUMPALL with the following message:

```
<mix number> <file name> REQUIRES MT #n
```

When a scratch tape becomes available, the operating system assigns the tape to DUMPALL.

DUMPALL Utility

- If the system option SERIALNUMBER is set, the operating system suspends DUMPALL with the following message:

```
<mix number> <file name> REQUIRES MT #n
```

When a scratch tape becomes available, the operator responds with the following system command. The variable nn is the unit number of the tape drive.

```
<mix number> OU MT nn
```

If DUMPALL fills up the entire tape volume and still has more data to copy, it requests another tape volume from the operator by the same process it used for the first volume. When that tape is assigned, DUMPALL continues the copy process on the new volume.

Input Files from Labeled Tapes

You can control the tape input to DUMPALL by specifying the correct file title of the tape file that you want DUMPALL to print or copy. DUMPALL automatically picks up the record size and block size information for the file from the tape labels.

If you do not specify the correct file title, if the file is not the first file on the tape and the tape does not have a standard tape file name in the form volumeid/fileid, or if the tape is not online when you run DUMPALL, the operating system suspends DUMPALL with the following message:

```
<mix number> NO FILE <file name> (MT) #n
```

The operator can mount the appropriate tape or use the IL (Ignore Label) or FA (File Attribute) system command to force the operating system to select a tape.

Input Files from Unlabeled Tapes

You can use the FR or UL option, explained under “Standard Commands” earlier in this section, to indicate that the input file has no label. Without label information, DUMPALL cannot determine the record size or block size to use, so you must specify the MAXRECSIZE and BLOCKSIZE file attributes. Without label information, the operating system cannot identify the tape to select without operator intervention. Thus, when DUMPALL opens the input tape file, the operating system suspends DUMPALL with one of the following messages:

```
<mix number> NO FILE UL (UNLABELED MT) #1
```

```
<mix number> NO FILE FR (UNLABELED MT) #1
```

By entering the following system command, the operator assigns to DUMPALL an unlabeled file on the indicated unit:

```
<mix number> UL MT nn
```

If you specify UL and do not use either the MULTI or the SKIPTM option, the operating system performs an automatic reel switch when it reads a tape mark from the tape. It then suspends DUMPALL with the following RSVP message:

```
<mix number> NO FILE UL (UNLABELED MT) #2
```

The operator can continue to force the operating system to select a tape by using the UL (Unlabeled) system command. After the operating system reads the last tape, the operator must enter the following response to inform the operating system that the last tape used was the final reel:

```
<mix number> FR
```

If you specified FR, or used UL in conjunction with MULTI, SKIPTM, or both, the system requests no further input file at the end of the first input tape.

Note: *The operator can also select a labeled tape as a response to the “NO FILE” message. If this happens, your DUMPALL command probably executes incorrectly because DUMPALL treats the beginning label records as the data blocks of the first file on the tape. So, DUMPALL copies or prints the label records instead of the data records.*

Treating Labeled Tapes as Unlabeled Tapes

DUMPALL and the operating system permit you to read any tape—labeled, unlabeled, or with nonstandard labels—as though it had no label records. You might want to use this feature if, for some reason, the labels on the tape are defective or not compatible with the A Series operating system, or if you do not know the names of the files on the tape. When you choose to treat a labeled tape as an unlabeled tape, DUMPALL and the operating system do not attempt to interpret any of the labels on the tape. The data contained on the tape is assumed to comprise one or more files. File boundaries are delimited by tape marks.

DUMPALL provides the capability to treat any tape as if it were unlabeled by using the UL or FR option. When you specify UL or FR, DUMPALL opens an input file as an unlabeled tape. This enables you to access any file from the tape. However, you must then make sure you position the tape at the desired file. This action requires a precise knowledge of the location of the file to be read—that is, the number of tape marks that must be skipped to reach the file.

When DUMPALL opens the tape, the operating system suspends the program and requires an RSVP to the following message:

```
<mix number> NO FILE UL (UNLABELED MT) #1
```

The operator must identify the location of the unlabeled tape by entering the following response:

```
<mix number> UL MT <unit number>
```


DUMPALL Utility

For example, in the case of labeled tapes, beginning and ending labels are now treated as files on the tape. Normally, a labeled tape contains n files. When the tape is treated as unlabeled, it appears to contain $3n + 1$ files. DUMPALL treats each set of beginning label records and each set of ending label records as separate files. Thus, to copy the third file on a labeled tape being treated as unlabeled, the following DUMPALL command is required:

```
COPY UL SKIPTM 7 TO FILETHREE (KIND=PACK, PACKNAME=MYPACK)
```

Note that seven files (three beginning labels, two ending labels, and two files) had to be skipped to reach the third file.

If you are unsure of the position of the tape marks on the tape, use the DMPMT command to list the contents of the tape. The resulting display lists the location of the tape marks.

Section 4

FILECOPY Utility

The FILECOPY utility simplifies library maintenance by automating the creation of copy decks. You specify the location and types of files to be copied. FILECOPY uses this information either to start a Work Flow Language (WFL) program to do the copying or to produce a punched card deck that, if run, performs the desired function. The library maintenance files processed by the ZIP command run from the same queue as the FILECOPY run. The class for the FILECOPY run is included as part of the output job deck. If the WFL deck is punched by FILECOPY, a user card must be added to the punched deck to be read into a secured reader. If the WFL deck is too large for WFL to compile, an appropriate error message is displayed. FILECOPY is discontinued after all WFL jobs have been initiated if any of the WFL compilations received a syntax error.

Files can be copied from pack or disk; however, they cannot be copied from tape. Files can be copied to tape, pack, or disk. The device number and serial number of the output device can be specified. If the information copied is expected to be long enough to require continuation reels, a serial number list can be specified.

Files can be copied by using their file names and directory names, and their creation, access, update, or expiration dates.

FILECOPY uses the GETSTATUS procedure of the operating system to initiate all its file and directory requests, thus causing system security to control file access.

FILECOPY does not apply family substitution when searching for and selecting the disk files to be copied; however, all input and output files used or created by FILECOPY are affected by family substitution. For runs of FILECOPY that do not use family substitution, output files are created on the family on which FILECOPY resides, unless requested from some other family.

Running the FILECOPY Utility

The following WFL job initiates FILECOPY:

```
<i>RUN SYSTEM/FILECOPY
<i>EBCDIC CARD
.
.
<i>input to filecopy>
.
.
<i>END JOB.
```

The *<i>* variable specifies an invalid character. When FILECOPY is run from an operator display terminal (ODT) or a remote terminal, the *<i>* variable is the question mark (?). When FILECOPY is run from a card reader, the *<i>* variable includes any

invalid punch. An <i> variable is optional when FILECOPY is run from an ODT or a remote terminal; however, it is required when FILECOPY is run from a card reader.

Input to the FILECOPY Utility

FILECOPY receives its instructions through a file called INPUT. This file is assumed to have card-reader characteristics, but its records must be randomly accessible. Instructions are in a free-field format. FILECOPY correctly handles input from a 15-word-record-size disk file with a FILEKIND value of JOBSYMBOL.

You can enter a maximum of five separate sets of instructions separated by semicolons, to produce a mixture of punch card decks or WFL jobs that are processed by the ZIP command. A set includes a FILECOPY task request and a FILECOPY modifier. The FILECOPY task requests specify the method by which individual files are to be included in the copy output. The FILECOPY modifiers provide data needed for the checking performed by a FILECOPY task request, plus other control information. The task requests and modifiers are explained later in this section.

The input syntax to FILECOPY can include three lists of files. A FILES list specifies files that FILECOPY is to check against the specified criteria (EXPIRED, CREATED, and so forth). An INCLUDE list permits files to be included that do not match the criteria. An EXCLUDE list permits files to be excluded that do meet the criteria. These lists allow several files or directories to be specified from a single family.

Additional information is derived from index files. An index file contains a complete list of all files that meet the testing criteria of the task or are included in the task. The format of the index file is shown under "Index Files" later in this section. An index file is created by FILECOPY when the LOCKINDEX option is specified, and is demanded by FILECOPY in response to the ADDED task request.

After the desired FILECOPY request is entered, any options can be placed in free form in any order on one or more records. Order is important only in cases in which some options override others.

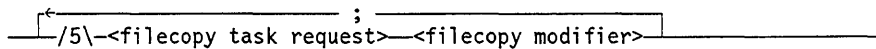
Default specifications exist for each option except the FILECOPY task requests. Default specifications, requested options, and results are listed at the end of a run unless they have been suppressed.

If any input errors are detected, the entire run is terminated, although all input is checked for syntax errors. This action is performed so that an intended multiple run can be completed as such, rather than as two runs taking a longer total time.

If errors are found during the creation of copy lists, attempts are made to recover. If these attempts fail, processing is terminated for those requests that have been flagged with error messages, but other requests proceed unaffected.

The output consists of a list showing each line of the input. If any line contains errors, these errors are flagged with a message immediately below the offending line. Errors that are detected only after the entire input has been processed appear after all input has been printed. For ease of reference, the invalid task request is printed along with the associated error message.

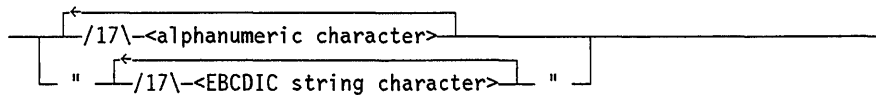
<input to filecopy>



Basic FILECOPY Constructs

The following items commonly appear as syntactic variables in the syntax diagrams featured in this section.

<identifier>



Explanation

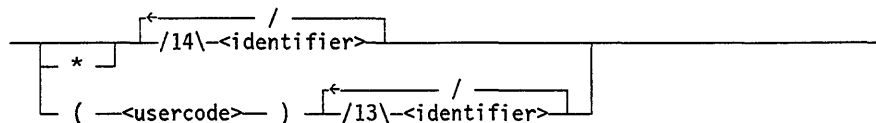
<alphanumeric character>

Any of the characters A through Z or 0 through 9, inclusive.

<EBCDIC string character>

Any one of the 256 EBCDIC characters except a double quotation mark (").

<file name>



Explanation

<usercode>

An identifier that specifies the usercode of the file. The quoted form of the identifier cannot be used for the usercode.

<identifier>

Specifies a file that is not associated with a usercode.

If the file name is part of a file title that includes an *ON* <family name> specification, then the maximum number of identifiers the file-name part can contain is reduced by 1 (that is, from 12 to 11 if no usercode appears, or from 11 to 10 if a usercode appears).

<file title>
 ---<file name>-----
 | ON ---<family name>|

Explanation

ON <family name>

Specifies a disk family. The default family is DISK.

<family name>

Identifies a family. A family is a logical means of grouping mass storage devices together so that they function as one logical unit. Specifying the identifier as the family name indicates a named native-mode disk pack (with any continuation packs) whose name is the specified family name. A file with a KIND attribute value of DISK or PACK refers, by default, to the DISK family. The quoted form of the identifier cannot be used as a family name.

<timestamp>

 |<date>| @ | /4\---<integer>|-----
 | TODAY | |-----
 | - ---<integer>|

<date>

--- <digit> <digit> --- / --- <digit> <digit> --- / --- <digit> <digit> -----

Explanation

<date>

The date is specified in the form *mm/dd/yy*, where m, d, and y are digits representing the month, day, and year, respectively.

<date> @ <integer>

Specifies the day and time of day. If a <date> is not specified, the current date is used. If @ <integer> is not specified, a value of zero (0) is used.

TODAY

Indicates that the timestamp to be used is the date that FILECOPY is executed.

TODAY - <integer>

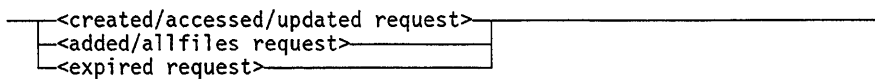
Indicates that the timestamp refers to <integer> days before the TODAY value.

FILECOPY Task Requests

The following text describes the task requests that you can use when running FILECOPY.

Each task request passes over the specified files and directories and produces a single list that contains all files found that meet the specified criteria. A maximum of five task requests are allowed in a single run of FILECOPY. The following diagram lists the possible task requests. Each request is defined under a separate heading in the following text.

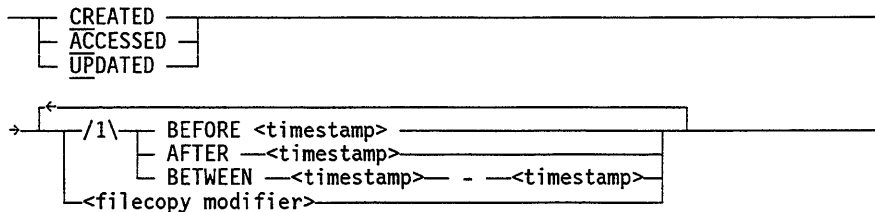
<filecopy task request>



CREATED, ACCESSED, or UPDATED Request

The CREATED/ACCESSED/UPDATED requests copy files that have been created, accessed, or updated during a specified period.

<created/accessed/updated request>



Explanation

CREATED

Copies files whose creation date (the date the file was first locked on disk or a disk pack) is less than the BEFORE <timestamp> value, greater than or equal to the AFTER <timestamp> value, or between (inclusive) the specified timestamps specified by the BETWEEN option. The FILEKIND of the WFL deck created by the LOCKDECK command defaults to JOBSYMBOL when BETWEEN is specified.

ACCESSED

Copies files whose date of last access (the date the file was last opened for input or output) is less than the BEFORE <timestamp> value, greater than or equal to the AFTER <timestamp> value, or between (inclusive) the timestamps specified by the BETWEEN option. The FILEKIND of the WFL deck created by the LOCKDECK command defaults to JOBSYMBOL when BETWEEN is specified.

FILECOPY Utility

UPDATED

Copies files whose timestamp value is within the specified time range or whose ALTERDATE and ALTERMIME values are within the specified time range. The attributes contain the date and time the file was last closed after being written to and is less than the BEFORE <timestamp> value, greater than or equal to the AFTER <timestamp> value, or between (inclusive) the timestamps specified by the BETWEEN option. The FILEKIND of the WFL deck created by the LOCKDEF command defaults to JOBSYMBOL when BETWEEN is specified.

<timestamp>

Indicates a particular date. If no timestamp is specified, the TODAY value is used, and checking is done with the AFTER option.

If no @ <integer> portion of the timestamp is specified, a value of zero (0) is used, except in the second timestamp in the BETWEEN option, in which case a value of 2359 is used.

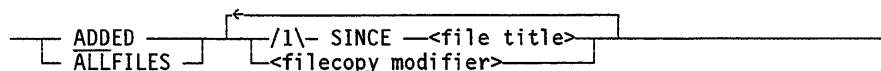
<filecopy modifier>

Provides data needed for the checking performed by the FILECOPY task request plus other control information.

ADDED or ALLFILES Request

The ADDED and ALLFILES requests copy files created since a particular date or simply copy all files encountered.

<added/allfiles request>



Explanation

ADDED SINCE <file title>

Copies files whose creation date is less than the creation date of the index file specified by the file title and that are not listed in that index file.

ALLFILES SINCE <file title>

Copies all files encountered except the files listed in the index file specified by the file title. The file title must refer to an index file. Index files are explained under “Index Files” later in this section.

ALLFILES <filecopy modifier>

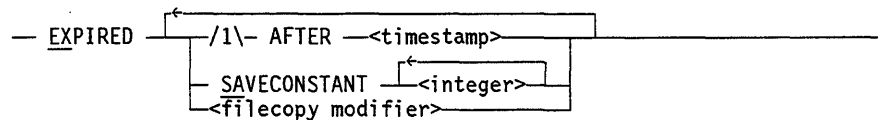
Copies all files encountered (subject to any exclusion lists or criteria specified in the modifiers).

<filecopy modifier>

Provides data needed for the checking performed by the FILECOPY task request, plus other control information.

EXPIRED Request

The EXPIRED request copies files that are older than a particular date.

<expired request>**Explanation****EXPIRED AFTER <timestamp>**

Copies files whose last access date plus the minimum of the SAVEFACTOR and the SAVECONSTANT values of each file is less than or equal to the timestamp. The default value for SAVECONSTANT is 60.

If no AFTER <timestamp> value is specified, the TODAY value is extracted from the system, and checking is done with the AFTER option.

EXPIRED SAVECONSTANT <integer>

Overrides the SAVEFACTOR value of each file if the value of SAVECONSTANT is less than the value of SAVEFACTOR. The algorithm used for EXPIRED is as follows:

```
LASTACCESSDATE + MIN(SAVEFACTOR,SAVECONSTANT)
LEQ AFTER date
```

In this algorithm, LASTACCESSDATE is the date the file was last accessed.

Normally the SAVECONSTANT value is used to determine the EXPIRED condition for all files having a SAVEFACTOR value of zero (0). The SFACORZERO option causes the zero SAVEFACTOR of a file to be used in determining whether the file is expired. Refer to "FILECOPY Options" later in this section for more information.

The default value of SAVECONSTANT is 60. The default can be altered by recompiling FILECOPY after changing the define DEFAULTSFOVERRIDE in SYMBOL/FILECOPY.

FILECOPY Utility

The SAVECONSTANT option enables the system to define when a file has expired instead of permitting each file to define its expiration date.

<filecopy modifier>

Provides data needed for the checking performed by the FILECOPY task request plus other control information.

FILECOPY Modifiers

The following text describes the FILECOPY modifiers and options that can be specified with the FILECOPY task requests.

Unless overriding FILECOPY modifiers are given, the following defaults are used for each task request:

- The date used in checking is the TODAY value extracted from the system. In the task requests CREATED, ACCESSED, and UPDATED, the testing is done with the AFTER option.
- The WFL deck resulting from a successful run is processed by a ZIP command.
- The WFL deck is a simple COPY operation (without COMPARE or BACKUP).
- No index file is created.
- An output summary is printed.
- The usercode under which FILECOPY is running is prefixed to all file names except files that explicitly specify a usercode, to system files, or to a file name that includes the equal sign (=).
- If no family name is specified, the files are assumed to reside on the family DISK.
- Files are copied to a tape whose default name is that of the task request (CREATED, EXPIRED, and so forth) with no DENSITY, UNITNO, or SERIALNO attribute values specified.
- If the task request is EXPIRED, all files are considered to be expired if they were last accessed more than 60 days before the current system date. The default value for SAVECONSTANT attribute is 60.

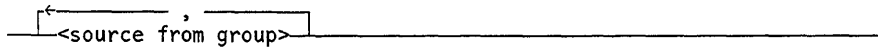
<filecopy modifier>

┌──────────┐ ┌──────────┐ ┌──────────┐
└──────────┘ <file specification> └──────────┘
└──────────┘ └──────────┘

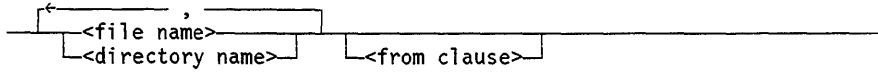
<file specification>

┌── FILES ──┐ (─<source> ─) ┌──────────┐
└── INCLUDE ─┘
└── EXCLUDE ─┘ └──────────┘

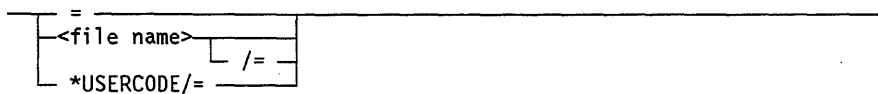
<source>



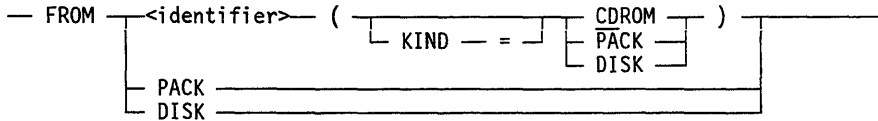
<source from group>



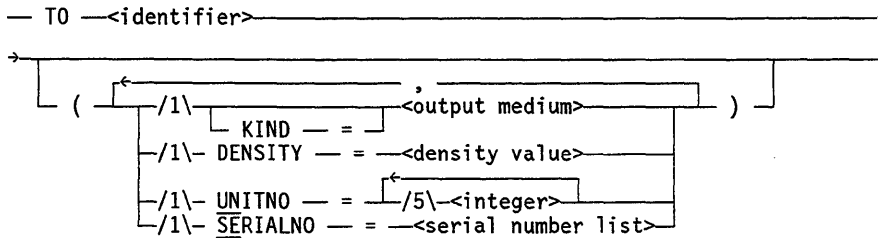
<source file name>



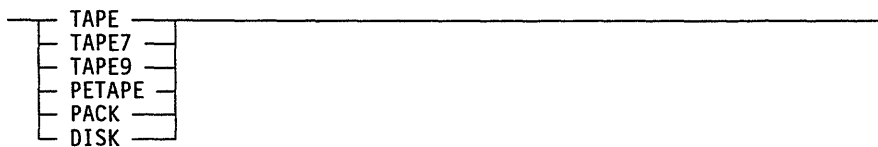
<from clause>



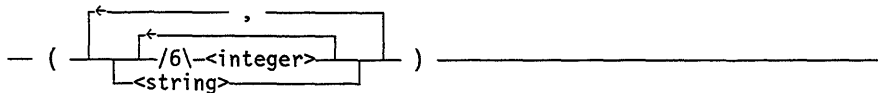
<destination>



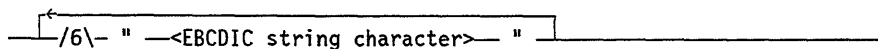
<output medium>



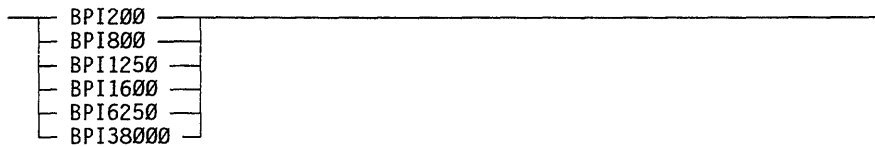
<serial number list>



<string>



<density value>



Explanation

<file specification>

Specifies a list of files to be considered, included, or excluded. Either a FILES list or an EXCLUDE list must be specified.

FILES (<source>)

Causes the files listed in the source variable to be checked against the criteria from the FILECOPY task request. The files that meet the criteria and that are not excluded in some way, are copied. Files specified in this list are obtained by way of GETSTATUS calls and checked against the specified testing criteria.

INCLUDE (<source>)

Causes the files listed in the source variable to be copied whether or not they meet the specified criteria.

INCLUDE overrides EXCLUDE; that is, if a file is on an EXCLUDE list, specified either by name or by FILEKIND value, and is explicitly included by name, the file is included.

EXCLUDE (<source>)

Causes the files listed in the source variable not to be copied, even if they meet the specified criteria. Files can also be excluded based on their FILEKIND attribute values (CODE, DATA, and so forth) as specified under "FILECOPY Options" later in this section. Both the file name and the FILEKIND cannot be used in the same EXCLUDE statement, although each type can appear alone several times in the following form:

```
EXCLUDE (<file names>) EXCLUDE FILEKIND(<kinds>)
```

<source from group>

In a <source from group> phrase, a FROM clause applies to all source file names in that source from group.

If no FROM clause is specified, files are assumed to reside on the family DISK.

<source file name>

Specifies the name of the input files.

The equal sign (=) specifies all files.

The *<file name>/=* format specifies a directory. All files under that directory are processed. If the file name is not followed by a slash and an equal sign (/=), it refers to only one file.

The **USERCODE/=* format specifies all usercoded files.

<from clause>

Specifies the location of the input files. The input files can reside on the family DISK or PACK, but not on TAPE.

The identifier can be used to name a particular family.

A FROM clause can be specified for each input file in the task. If a FROM clause is not specified for an input file, the file is assumed to reside in the location specified by the next-mentioned FROM clause. For example, the statement FILES A, B FROM PACK specifies that files A and B both reside on PACK.

If no FROM clause is specified, or if there is no next-mentioned FROM clause, files are assumed to reside on the family DISK. For example, the statement FILES A, B specifies that files A and B both reside on DISK. The statement FILES A, B FROM PACK, C specifies that files A and B reside on PACK and that file C resides on DISK.

<destination>

Specifies the name and type of the output medium. If no destination is specified, files are copied to a tape whose default name is that of the task request (CREATED, ACCESSED, and so forth) with no UNITNO or SERIALNO attribute value supplied.

TO <identifier>

Labels the output medium.

KIND = <output medium>

Specifies the output medium (PETAPE, PACK, and so forth) to which the files are to be copied.

DENSITY = <density value>

Specifies the density of the medium on which the file is copied.

FILECOPY Utility

UNITNO = <integer>

Specifies the unit number of the medium to which the file are to be copied. The maximum unit number allowed is 32767. The unit number can be used only if the OLDWFL option is specified.

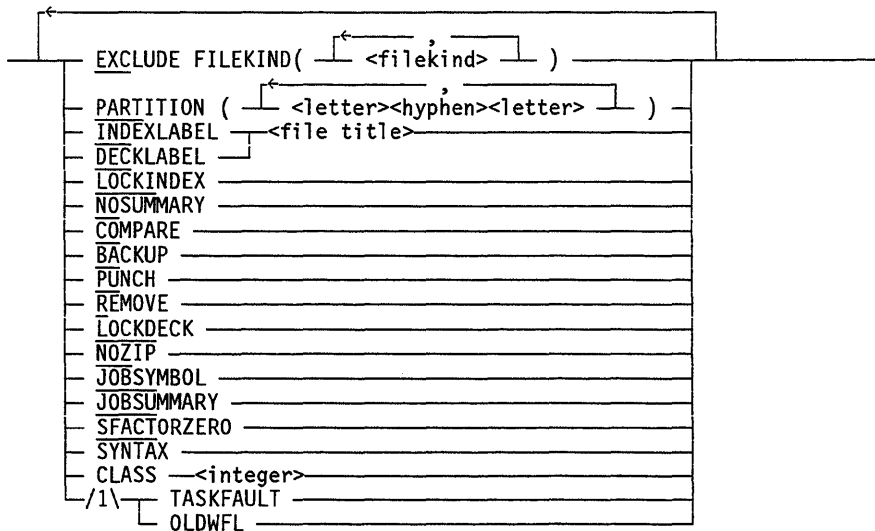
SERIALNO = <serial number list>

Specifies a serial number list to be attached to each destination volume. Serial numbers must be no more than 6 characters long and are not allowed in the same task with partitioning.

FILECOPY Options

Certain options can be specified with the file specifications to indicate how the FILECOPY should be run.

<option>



Explanation

<filekind>

Any FILEKIND attribute defined in the *A Series File Attributes Programming Reference Manual*.

EXCLUDE FILEKIND (<filekind>)

Prevents files from being copied based on their FILEKIND values (CODE, DATA, and so forth). Files can also be excluded by name in an EXCLUDE list in the file specifications. Both kinds of EXCLUDE options cannot be used in the same EXCLUDE statement, although each can appear alone several times in the following form:

EXCLUDE (<file names>) EXCLUDE FILEKIND (<filekinds>)

If a file is in an EXCLUDE list, either by name or by FILEKIND value, and is explicitly included by name, the file is included.

PARTITION <letter> - <letter>

Allows the files copied to be divided by volume based on the first character of the file name. The ranges specified by <letter>-<letter> are used as test points. When the first character of a file name crosses a partition boundary, the WFL program reflects this crossing by sending the copy to the current volume and starting a new copy to a new volume, but with the same KIND attribute value. Also, if an index file is being generated, it is closed and locked, and a new one is opened to handle the next partition.

The partition bounds are appended to the destination tape and index disk file title (either explicitly specified or assumed by default). For example, if the partition is A-H, the LABEL value is AHCREATED and the index filename is INDEX/CREATED/AH. A single-character partition such as P-P is valid. Partitions must not overlap in any way. Any unspecified partitions are interpolated: (given) A-H,Q-T (final) A-H,I-P,Q-T,U-Z. Files that start with special characters are placed in the first partition, while those starting with numbers are placed in the last.

INDEXLABEL <file title>

Names the index file. The default name of the index file is *INDEX/<identifier>*, where the identifier is the name of the medium to which the files are to be copied. The index file is created only if LOCKINDEX is requested. This file is a complete list of all files that meet the testing criteria of the task or have been processed by the INCLUDE command. The format of the index file is shown under "Index Files" later in this section.

DECKLABEL <file title>

Names the output WFL deck. The default name of the file is T<n>OUTPUT, where <n> is the number of the task, for example T1OUTPUT, T3OUTPUT.

LOCKINDEX

Creates and locks an index file.

NOSUMMARY

Suppresses the summary listing of the FILECOPY run.

COMPARE

Causes the library maintenance option COMPARE to be used when FILECOPY is run.

FILECOPY Utility

BACKUP

Causes the library maintenance option **BACKUP** to be used when **FILECOPY** is run. On a cataloging system, this option causes the backup information for the copied files to be stored in the catalog. The destination disk or tape for the copied information must be listed in the volume library.

PUNCH

Sends the generated WFL program to the card punch. This deck is ready for use as card reader input as soon as the **LABEL** cards are taken off.

REMOVE

Removes the copied files as soon as the copy is complete. The operator must **OK** the removal before it takes place.

LOCKDECK

Locks the WFL deck on disk for future use. By default, all decks are processed by the **ZIP** command and removed. If the **LOCKDECK** option is specified, the deck is processed by the command but not removed. This feature permits **FILECOPY** to generate a **COPY** deck that you can modify, use, and reuse.

NOZIP

Allows creation of the output WFL deck without using the **ZIP** command.

JOBSYMBOL

Causes the **FILEKIND** value of the WFL deck created by **LOCKDECK** to be **JOBSYMBOL**. The default **FILEKIND** value for the WFL deck is **DATA**.

JOBSUMMARY

Adds a **MYJOB** (**JOBSUMMARY = UNCONDITIONAL**); statement to the tasks output WFL deck. This option is **RESET** by default and causes no job summary to be printed.

SFACTORZERO

Only valid for the **EXPIRED** task request. When **SFACTORZERO** is specified, **FILECOPY** uses the zero **SAVEFACTOR** value of a file, rather than the modifier **SAVECONSTANT**, for all files having a **SAVEFACTOR** value of zero.

SYNTAX

Causes FILECOPY to do syntax checking of only the input. The use of the SYNTAX option in any task request of a multitask FILECOPY run causes all task requests to stop after the input syntax checking is complete.

CLASS <integer>

Specifies the class of the output WFL deck. If the CLASS option is not specified, the WFL deck inherits the class of the FILECOPY that created it. The <integer> value must be within the range 0 through 1023.

TASKFAULT

Adds an *ON TASKFAULT, GO TO THETOP* statement to the output WFL deck of the task. With the corresponding LABEL "THETOP:" statement, any abnormal termination, including operator use of the DS command, causes the task to restart. The value of this option is RESET by default and causes no changes in the current format of the output WFL deck.

OLDWFL

Causes FILECOPY to create old (pre-2.9) WFL output decks that allow quoted file titles. New WFL (post-2.9 WFL) does not allow any characters in a file title other than letters, numbers, hyphens (-), and underscores (_); the use of quotation marks (") is not allowed. The use of a hyphen or an underscore in a file title causes FILECOPY to quote the title only if the option OLDWFL is specified. For FILECOPY task requests that do not use the OLDWFL option, files that violate the title syntax are reported on the summary report and are not copied.

If the OLDWFL option is specified, the option TASKFAULT cannot be used because it uses WFL features not supported in pre-2.9 WFL.

Note: Eventually, OLDWFL will no longer be supported; its use is not recommended.

Sample FILECOPY Runs

The following text contains several examples of running FILECOPY. Although up to five of the examples shown can be strung together as shown in the <input to filecopy> syntax, this stringing is not done here. Each example illustrates several features of the program.

The following example creates a punch deck that causes COPY & COMPARE of all files on the family named DISK that were created between (inclusive) the dates given. The files are to be dumped to a tape PETAPE whose name is to be WEEK36/FILE000. The serial number of the first tape is WK36X0, and the serial numbers of the first and second continuation reels are WK36X1 and WK36X2, respectively. A summary of the task is suppressed.

FILECOPY Utility

```
CREATED BETWEEN 9/8/90 - 9/14/90 COMPARE PUNCH
FILES (= FROM DISK) TO WEEK36 (KIND=PETAPE,
SERIALNO=("WK36X0", "WK36X1", "WK36X2")) NOSUMMARY
```

The following example copies all usercoded files on the diskpack PK that are EXPIRED and then, following the operator's OK, removes them. Files chosen for copying are those whose LASTACCESSDATE modifier value, plus the minimum of 60 or the file SAVEFACTOR value, is less than or equal to September 31, 1990. However, no files under the usercode of SITE are copied or removed, even though they can meet the expired criteria. A WFL file is generated and the ZIP command is used to do the copying. The library maintenance statement generated is COPY&COMPARE&BACKUP. Files are copied to a tape named EXPIRED by default.

```
EXPIRED AFTER 2/13/90 FILES (USERCODE/= FROM PK (PACK))
EXCLUDE ((SITE)= FROM PK (KIND=PACK))
COMPARE BACKUP REMOVE
```

The following example copies all files created or altered on the pack SAFEPACK prior to 12:30 on September 10. However, BD files and the file whose name is SYMBOL/MCP are not copied. An index file of all copied files is created with the name SAFETY/DUMP. The files are copied to a tape named UPDATED by default.

```
UPDATED BEFORE 9/10/90 @ 1230 FILES (= FROM SAFEPACK
(KIND=PACK)) EXCLUDE (*BD/=, *SYMBOL/MCP
FROM SAFEPACK (PACK)) LOCKINDEX
INDEXLABEL SAFETY/DUMP
```

The following example copies all files on DISK that are not listed in the index file SAFETY/DUMP and whose creation date is less than the creation date of SAFETY/DUMP. The files are copied to a tape mounted on unit 116 and labeled ADDER.

```
ADDED SINCE SAFETY/DUMP FILES (=) TO ADDER (UNITNO=116)
```

The following example copies all usercoded files on the default family DISK whose LASTACCESSDATE modifier value, plus the minimum of 14 or the file SAVEFACTOR value, is less than or equal to September 15, 1990. Also, all files whose first node is SYSTEM and the file named ADM are copied even if they are not usercoded or do not meet the EXPIRED criteria. Several volumes are created. These volumes are on tape and are labeled AHOLDUSERS, IROLDUSERS, and SZOLDUSERS. Each volume contains only files whose first title character falls within the bounds indicated by the leading two characters of the tape label.

```
EXPIRED AFTER 9/15/90 SAVECONSTANT 14
PARTITION (A-H,S-Z) FILES (USERCODE/=)
INCLUDE (SYSTEM/=, ADM) TO OLDUSERS
```

The following example copies all usercoded files created after September 21, 1990, that are not MCPCODE files or ESPOLSYMBOLIC files. However, the INCLUDE option overrides the CREATED criteria and the EXCLUDE list; therefore, all files under the usercode of SITE are copied even if they were created before September 21, 1990 or are

MCPCODE files or ESPOLSYMBOLIC files. The WFL deck is locked and processed by the ZIP command on the family X and is called SAVEWFL.

```

CREATED AFTER 9/21/90 FILES (USERCODE/=)
EXCLUDE FILEKIND (MCPCODE, ESPOLSYMBOLIC)
INCLUDE ((SITE)=) LOCKDECK DECKLABEL SAVEWFL ON X

```

The following example illustrates a syntax error. Due to the multiple volumes that are created (AHX, IPX, QZX), compliance with the SERIALNO request is impossible. Hence, this task (and any others requested in the same run) is not run. Instead, the syntax error message "PARTITIONING, SERIAL S NOT OK" appears.

```

CREATED AFTER 9/1/90 FILES (USERCODE/=)
PARTITION (I-P) TO X (SERIALNO=123)

```

The following example attempts to create a copy list of all files that were created today on the specified families. The files are copied to a tape named CREATED. However, the family IMNOT is not available, which causes FILECOPY to wait for this family to become available.

```

CREATED FILES (= FROM IMHERE (KIND=PACK), = FROM
IMNOT (KIND=PACK), ABC FROM BUTIAM (KIND=PACK))

```

In the following example, consider what happens if a hardware or MCP error causes FILECOPY to detect an error in the file information in the family HEADCRASH. If elements of the family have already been included in the copy list, then the task is terminated since it is impossible to recall the (possibly) bad information that has already been copied. However, if the problem is detected before any part of HEADCRASH has been used, then that family is ignored (although the copy deck reflects the lack of information from the family), and the program continues with the family COMFORTABLE.

```

ALLFILES FILES (= FROM BRANDNEW (KIND=PACK), = FROM
HEADCRASH (KIND=PACK), = FROM COMFORTABLE(KIND=PACK))

```

The following example copies all usercoded files on disk (not MYPACK) that are updated the day FILECOPY is run. The index file and output WFL deck are locked on a pack named MYPACK under the usercode PRIV with titles SAVEINDEX and SAVEDECK.

Family substitution does not apply to files accessed by FILECOPY; however, input and output files used or created by FILECOPY are affected by family substitution. For runs of FILECOPY that do not use family substitution, output files are created on the family on which FILECOPY resides, unless requested from some other family.

```

?BEGIN JOB RUNFILECOPY; USER=PRIV/ILEGED;
FAMILY DISK=MYPACK OTHERWISE DISK;
RUN FILECOPY ON DISK; EBCDIC CARD
UPDATED
FILES(USERCODE/= FROM DISK)
LOCKINDEX LABELINDEX SAVEINDEX
LOCKDECK DECKLABEL SAVEDECK
?END JOB

```

FILECOPY Utility

The following example copies all usercoded files on disk (not MYPACK) that were updated the day FILECOPY is run. The index file is locked on a pack named MYPACK (in the system node, not the usercode node) with the title SAVEINDEX. The output WFL deck is locked on a pack called CONTROL in the system node with the title SAVEDECK.

```
?BEGIN JOB RUNFILECOPY; USER=PRIV/ILEGED;
FAMILY DISK=MYPACK OTHERWISE DISK;
RUN FILECOPY ON DISK; EBCDIC CARD
UPDATED
    FILES(USERCODE/= FROM DISK)
    LOCKINDEX LABELINDEX *SAVEINDEX
    LOCKDECK DECKLABEL *SAVEDECK ON CONTROL
?END JOB
```

In the following example, the usercode PRIV has no family associated with it. All usercoded files on TEMP that were updated the day FILECOPY is run are copied. The index file is locked on the pack from which FILECOPY is being run (UTILPACK) and is titled (PRIV)SAVEINDEX.

```
?BEGIN JOB RUNFILECOPY; USER=PRIV/ILEGED;
?RUN FILECOPY ON UTILPACK; EBCDIC CARD
UPDATED
    FILES(USERCODE/= FROM TEMP(KIND=PACK))
    LOCKINDEX LABELINDEX SAVEINDEX
?END JOB
```

Index Files

An index file contains a complete list of all files that meet the testing criteria of the task or are included in the task. These files are created by the program only if the option LOCKINDEX is specified. However, they are used as input by FILECOPY when the ADDED request is used.

An index file is a disk or disk pack file with a MAXRECSIZE value of 30 words. The first record is reserved for future use. The body of an index file consists of logical records describing files. Each record contains the file name and other information. A logical record can extend over more than one physical record. Each logical record starts at the boundary of a physical record.

The following describes the file layout:

Logical Record 0
Length: 30 words
Contents: Reserved

Logical Record 1 - end
Length: Variable in 30-word segments

Contents:

Word 0 : Reserved
Word 1 : Word displacement from
start of record to
start of standardform
<file name>.

Word 2 - (Word 1) : Reserved

Words (Word 1) - End of logical record:
Standardform <file name>.

Section 5

FILEDATA Utility

The SYSTEM/FILEDATA utility produces selected reports regarding permanent disk files and library maintenance tapes. The following are some of the report types you can request, together with the name of the task request to use for each report type:

- A hierarchical list of files with certain file attributes (FILENAME request).
- A hierarchical list of files with archive information about each file (ARCHIVEINFO request).
- A hierarchical list of files with catalog information about each file (CATALOGINFO request).
- The storage layout of files (STRUCTUREMAP request).
- Disk checkerboarding (CHECKERBOARD request).
- A list of files selected and/or sorted by the value of their AREASIZE attribute (AREASUMMARY request).
- Specified attributes of a file or a group of files (ATTRIBUTES request).
- A list of file names contained in the directory of a library maintenance tape (TAPEDIR request).
- A list of library maintenance tapes that contain backup copies of cataloged files (BACKUP request).
- Specified attributes of a code file or group of code files (CODEFILEINFO request).
- A list of code files compatible or incompatible with a given set of host systems (COMPATIBILITY and INCOMPATIBILITY requests).
- A file suitable for use in a library maintenance COPY statement (COPYDECK request).
- A raw (HEX) dump of disk file headers and, optionally, catalog information and archive records (HEADERCONTENTS request).

Each of these task request types is discussed in more detail later in this section.

Basic FILEDATA Constructs

The following items commonly appear as syntactic variables in the syntax diagrams featured in this section. The variables are presented in alphabetical order (ignoring nonalphabetic characters).

<alphanumeric character>

Any of the characters A through Z or 0 through 9, inclusive.

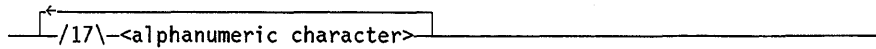
<digit>

Any one of the decimal digits 0 through 9, inclusive.

<EBCDIC string character>

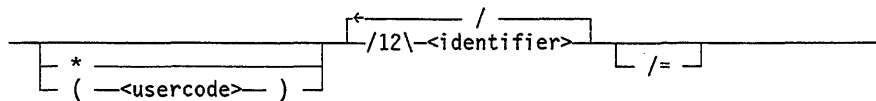
Any one of the displayed EBCDIC characters except the double quotation mark (").

<family name>

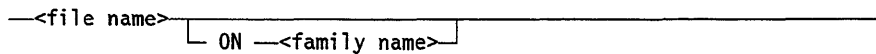


Identifies a disk family. The actual disk family reported on can be affected by family substitution, as discussed under "Effects of Family Substitution" later in this section.

<file name>



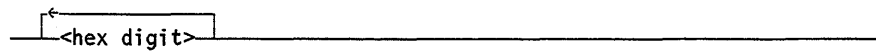
<file title>



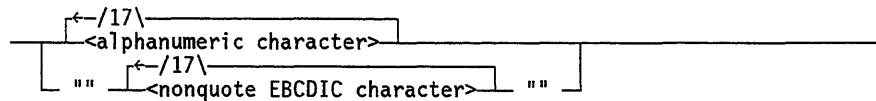
<hex digit>

Any one of the hexadecimal digits 0 through 9 or A through F. Hexadecimal digits can be used to create strings.

<hex string>

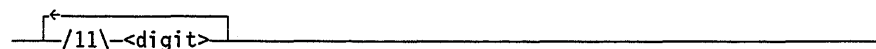


<identifier>



When non-alphanumeric characters are included in an identifier, the identifier must be enclosed in two sets of quotation marks. The quotation marks must be doubled because they are embedded within a longer string (the parameter list).

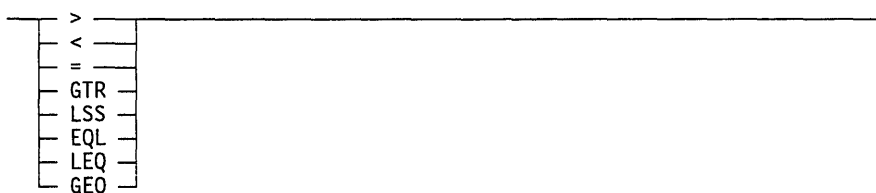
<integer>



<nonquote EBCDIC character>

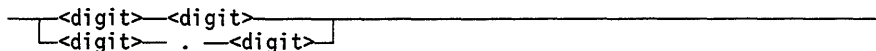
Any EBCDIC character for which the hexadecimal code is greater than or equal to hexadecimal 40 and which is not the quotation mark (").

<relation>

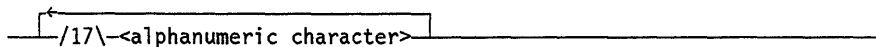


The GTR and > operators both mean *greater than*. The LSS and < operators both mean *less than*. The EQL and = operators both mean *equal to*. The GEQ operator means *greater than or equal to*, and the LEQ operator means *less than or equal to*.

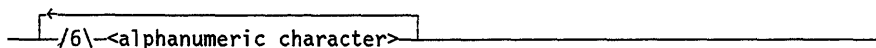
<release level>



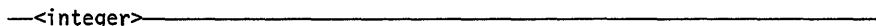
<tape name>



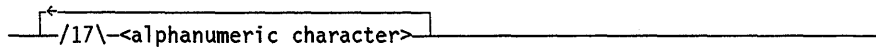
<tape serial number>



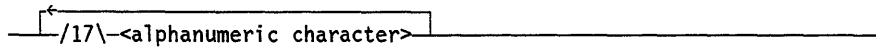
<unit number>



<usercode>



<volume name>



The name of a tape or CD-ROM optical disk.

Output Options

You can use output options to specify the device that FILEDATA writes a report to. These output options can be specified in each FILEDATA task request. The following are the output options provided by FILEDATA:

<output option>

```
—/1\  PRINTER _____|
      PUNCH _____|
      TTY   _____|
      SCREEN _____|
      SPO   _____|
      FILENAME — = —<file title>|
```

- **PRINTER**
Causes output to be written to a line printer file. The default form size, in terms of characters per line and lines per page, is determined by the convention under which FILEDATA runs. To determine your current convention and to view the convention definition, enter GO CONVENTION in the Action field of the MARC home menu. For further information about conventions, refer to the *A Series MultiLingual System (MLS) Administration, Operations, and Programming Guide*.
- **PUNCH**
Causes output to be punched on a card punch, on a default card size 80 columns wide.
- **SCREEN**
Causes output to be displayed on the originating remote terminal or ODT, with a default screen size of 80 characters by 24 lines. You should use SCREEN for ODT devices that have been configured with invisible end-of-text (ETX) characters.
- **SPO**
Causes output to be displayed on the originating ODT, on a default screen size of 80 characters by 24 lines. You should use SPO for ODT devices that have been configured with visible end-of-text (ETX) characters. Paging is controlled interactively by you.
- **TTY**
Causes output to be displayed on the originating remote terminal, on a default form size of 80 characters by 60 lines. There is no interactive paging control.

- FILENAME = <file title>

Causes output to be written to a disk file, by default a data file with a record size of 80 characters. The title of the output file is the supplied file title.

***Note:** The FILENAME = <file title> option is used to write output to a disk file that has a default record size of 80 characters. However, some of the FILEDATA task requests require a record size of 132 characters. If you use the FILENAME option with one of these task requests, you must first use the DEFINEOUTPUT request to specify a LINEWIDTH of 132 characters. Otherwise, an error occurs and your job is terminated.*

Refer to the descriptions of the FILEDATA task requests for information on output and all default values.

Running the FILEDATA Utility

The FILEDATA utility program can be executed in the following three ways:

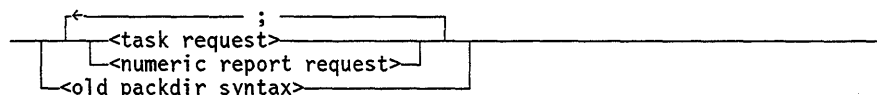
- By using a RUN command in a CANDE or MARC session, in a WFL job, or at an ODT. The RUN command must be of the form
RUN *SYSTEM/FILEDATA("<parameter list>").
- By entering the DIR (Directory) or TDIR (Tape Directory) system commands. Refer to "Using System Commands to Initiate FILEDATA" later in this section.
- By entering the CANDE LFILES command. The LFILES command can be used to display information for one or more files in the same directory. For an explanation of the CANDE LFILES command, refer to the *A Series CANDE Operations Reference Manual*.
- By entering the DATA selection on the UTIL screen in a MARC session.

Examples of statements that run FILEDATA are given under "Sample FILEDATA Runs" later in this section.

FILEDATA Parameter List

The input to FILEDATA determines the files that are to be reported on, the information that is to be included in the report, and the format of the output.

<parameter list>



Explanation

<task request>

For information about task requests, refer to "Task Requests" later in this section.

<numeric report request>

Numeric report requests can be specified in the parameter list or in an assignment to the TASKVALUE task attribute. For information about numeric report requests, refer to “Numeric Report Requests” later in this section.

<old packdir syntax>

Uses an older, nonpreferred syntax to initiate various FILEDATA reports. For information about this syntax, refer to “Old PACKDIR Syntax” later in this section.

Selecting the Files to Be Reported On

If you run FILEDATA under a privileged usercode or from an ODT, then by default FILEDATA reports on all the files on the disk family. However, if you use the TITLE, DIRECTORY, or DATABASE modifier, then FILEDATA limits its report to files in the specified directory or in the specified database.

If you run FILEDATA under a nonprivileged usercode, then by default FILEDATA reports on all public, nonusercoded files. You can use the TITLE, DIRECTORY, or DATABASE modifier to limit the report. However, if you use the TITLE or DIRECTORY modifier to specify files stored under a different usercode, FILEDATA returns the message “REQUESTED FILE OR DIRECTORY NOT FOUND”.

Effects of Family Substitution

By default, FILEDATA reports on the family named DISK. The FAMILYNAME modifier can be used to specify that a different disk family should be reported on.

If FILEDATA is run with the FAMILY task attribute set, this attribute can direct FILEDATA to report on different families than it otherwise would. For example, if the FAMILY task attribute value is “DISK = XPACK OTHERWISE DISK”, then XPACK becomes the default family for FILEDATA reports. If the FAMILY value is “XPACK = DBFAM ONLY”, and a FILEDATA task request uses the modifier FAMILYNAME = XPACK, then FILEDATA reports on family DBFAM instead. For a detailed explanation of the FAMILY task attribute, refer to the *A Series Task Attributes Programming Reference Manual*.

FILEDATA overrides the effects of the FAMILY task attribute for task requests that include the GUARDFILE modifier. Refer to “GUARDFILE” later in this section.

Error Reporting

Your input to FILEDATA can include more than one request, separated by semicolons. FILEDATA handles one request at a time. First it scans the input text for a request and then, if there are no severe errors in the syntax, it produces the requested report. Then FILEDATA prints out the identifier of the requested task, your input syntax for that task, and error or warning messages for any discrepancies FILEDATA found in your input.

Database Generation and Reuse

In order to handle any specific request and generate a report, FILEDATA uses information in the disk or tape directory to build a disk file called the *database*, or else selects an existing database for reuse. FILEDATA then reads information from the database and generates the report you requested.

By default, FILEDATA creates a new database for the first request in a run, and this database is reused for each subsequent request in the same run. At the end of the FILEDATA run, the database is removed. The following rules modify this default behavior:

1. FILEDATA always creates a new database for a request that includes any of the following modifiers: DIRECTORY, FAMILYNAME, LEVEL, PACKNAME, TAPE, or TITLE.
2. If a request includes the NEWDATABASE = <file title> modifier, FILEDATA creates a new database for the request and saves the database as a permanent file with the specified title. Subsequent requests, in the same or later FILEDATA runs, can use the DATABASE = <file title> modifier to reuse this database.
3. You can use the NOREPORTS request to create a database without issuing any report immediately. The NEWDATABASE modifier must be used with this request.
4. If a request includes none of the modifiers discussed in rules 1 through 3, then the request reuses the database that was used in the immediately preceding request.

Note: *A database contains information only about files specified by the request that creates the database. For example, if the request that creates the database includes the DIRECTORY modifier, then subsequent requests that reuse that database can report only on the files in that directory.*

Certain requests make use of information that is not always included in a database when it is created. The following are modifiers that should be used in the database-creating request if the database is to be reused for particular reports:

- If a database is intended for reuse by an ARCHIVEINFO request, then the ARCHIVE modifier should be used in the database-creating request.
- If a database is intended for reuse by an CATALOGINFO request, then the CATALOGUE modifier should be used in the database-creating request.
- If a database is intended for reuse by a HEADERCONTENTS request, then the RAWHEADERS modifier should be used in the database-creating request.
- If a database is intended for reuse by an ATTRIBUTES or CODEFILEINFO request that includes the WARNINGS modifier, the WARNINGS modifier should be used in the database-creating request.

FILEDATA Utility

Database Examples

The following examples illustrate when FILEDATA creates a database and when it reuses an old database.

```
RUN *SYSTEM/FILEDATA("ATTRIBUTES:ALL;FILENAMES")
```

In the preceding statement, the "ATTRIBUTES:ALL" request causes a new database to be created. This database is reused for the FILENAMES request.

```
RUN *SYSTEM/FILEDATA("STRUCTUREMAP:DATABASE=MYDB;FILENAMES:DIRECTORY=A;  
HEADERCONTENTS:DIRECTORY=B")
```

In the preceding statement, the STRUCTUREMAP request reuses the existing database titled MYDB ON DISK. The FILENAMES and HEADERCONTENTS requests both create new databases, because each includes a DIRECTORY modifier. Only the database created by HEADERCONTENTS includes raw headers.

```
RUN *SYSTEM/FILEDATA("FILENAMES:RAWHEADERS;  
CHECKERBOARD:DIRECTORY=A RAWHEADERS;  
ATTRIBUTES:MAXRECSIZE MINRECSIZE BLOCKSIZE")
```

In the preceding statement, the FILENAMES request creates a database because FILENAMES is the first request, and the CHECKERBOARD request creates a database because of the DIRECTORY modifier. Raw disk file headers are included in both these databases because of the RAWHEADERS modifier. The ATTRIBUTES request reuses the database from the CHECKERBOARD request.

Sample FILEDATA Runs

The following WFL job prints a hierarchical listing of the file names under the directory SYMBOL/= on the family DISK. The report includes the file kind, creation date, size in disk segments, security, and status of each file under the directory. Refer to "FILENAMES Request" later in this section.

```
BEGIN JOB;  
  RUN *SYSTEM/FILEDATA ("FILENAMES: TITLE =SYMBOL");  
END JOB
```

The following WFL job prints a report of those disk segments in use by permanent files on the family XPACK and those disk segments not in use by permanent files—that is, those segments that are available or in use by temporary files. Refer to "CHECKERBOARD Request" later in this section.

```
BEGIN JOB;  
  RUN *SYSTEM/FILEDATA ("CHECKERBOARD: FAMILY=XPACK");  
END JOB
```

The following WFL job prints a FILENAMES report, a STRUCTUREMAP report, and a CHECKERBOARD report for the family DISK. Refer to "Numeric Report Requests" later in this section.

```
BEGIN JOB;
  RUN *SYSTEM/FILEDATA (" "); VALUE = 0;
END JOB
```

The following CANDE command initiates a FILEDATA report with three task requests. The DEFINEOUTPUT request defines the output for the following requests. The output is sent to a terminal screen and has 79 characters per line. The ATTRIBUTES request reports the LASTRECORD and security information on all files in the XYZ directory. The COPYDECK request produces a copydeck of all files with two-level file names in the XYZ directory. In this request the output is sent to the card punch. The HEADERCONTENTS request produces a report from the family MYPACK. In this request, the output is sent to the printer.

```
RUN *SYSTEM/FILEDATA ("DEFINEOUTPUT: MEDIATYPE = SCREEN LINewidth = 79;
  ATTRIBUTES: LASTRECORD SECURITY DIRECTORY = XYZ;
  COPYDECK: LEVEL = 2 PUNCH DIRECTORY = XYZ;
  HEADERCONTENTS: PRINTER FAMILYNAME = MYPACK");
```

The following CANDE command initiates a FILEDATA run with three task requests. The first request, NOREPORTS, does not produce a report but creates a database MYDB. MYDB contains information for all the files on the family MYPACK. MYDB is used in the FILENAMES and ATTRIBUTES requests. FILEDATA retrieves the information from the database rather than having to gather the information independently for each task request.

```
RUN *SYSTEM/FILEDATA ("NOREPORTS: NEWDATABASE = MYDB
  FAMILYNAME = MYPACK;
  FILENAMES: NAMESONLY DATABASE = MYDB;
  ATTRIBUTES: ALL DATABASE = MYDB");
```

The following CANDE command generates three FILEDATA reports. The DEFINEOUTPUT request defines the output for the requests that follow it. The output is sent to a remote terminal assumed to be a hard-copy print device. The length of each line is 80 characters, and the page size is 24 lines. The STRUCTUREMAP request produces a map showing file storage layout for all files under the usercode JPLANGE. The CHECKERBOARD request reports on all files from the pack OTHERPACK. The ATTRIBUTES request reports on the file MYFILE. The file name and the timestamp are included in the report.

```
RUN *SYSTEM/FILEDATA ("DEFINEOUTPUT: MEDIATYPE = TTY
  LINewidth = 80 PAGESIZE = 24;
  STRUCTUREMAP: TITLE=(JPLANGE);
  CHECKERBOARD: FAMILYNAME = OTHERPACK;
  ATTRIBUTES: TITLE = MYFILE TIMESTAMP");
```

Task Requests

The following are the available task requests:

<task request>

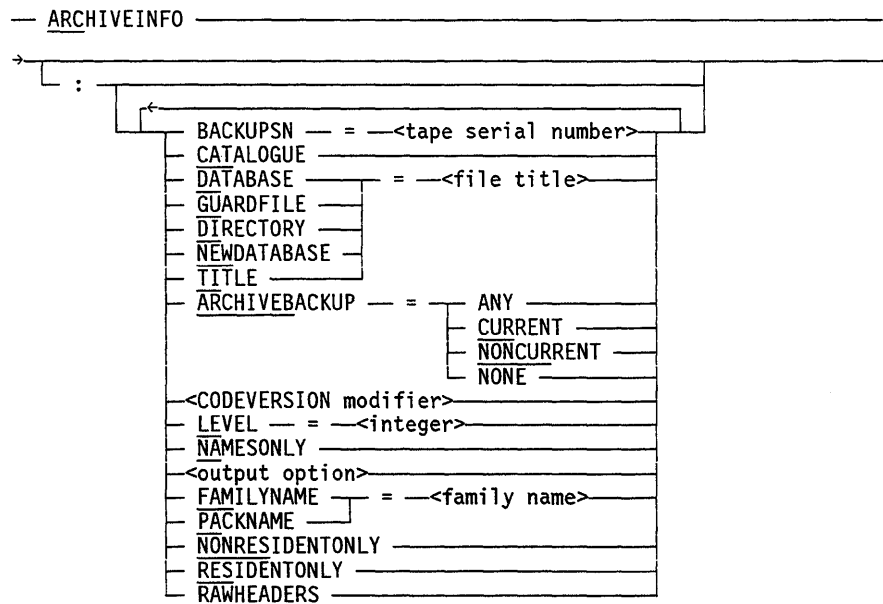
<archiveinfo request>	
<areasummary request>	
<attributes request>	
<backup request>	
<cataloginfo request>	
<checkerboard request>	
<codefileinfo request>	
<compatibility request>	
<copydeck request>	
<defineoutput request>	
<filenames request>	
<headercontents request>	
<incompatibility request>	
<noreports request>	
<structuremap request>	
<tapedir request>	

The following subsections provide the syntax and explanations of the task requests. Various modifiers that can be used in task requests are explained under “FILEDATA Modifiers” later in this section.

ARCHIVEINFO Request

The ARCHIVEINFO request produces a hierarchical list of files, including the access and the creation dates, the size in segments, the security class, the status and the file kind. This request produces the same report that is produced by the FILENAMES request that includes the ARCHIVE modifier. Refer to the description of the FILENAMES request in this section for more information. The output is sent to the printer by default.

<archiveinfo request>



Explanation

The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be reported on, and specifying the types of information to be reported for each file. These modifiers are explained under "FILEDATA Modifiers" later in this section.

- ARCHIVEBACKUP
- BACKUPSN
- CATALOGUE
- CODEVERSION
- DATABASE
- DIRECTORY
- FAMILYNAME
- GUARDFILE
- LEVEL

- NAMESONLY
- NEWDATABASE
- NONRESIDENTONLY
- RESIDENTONLY
- RAWHEADERS
- TITLE

ARCHIVEINFO Examples

The following example produces a ARCHIVEINFO report on all the files specified in the database UTILITYDB. UTILITYDB is a database created by a previous FILEDATA run.

```
RUN *SYSTEM/FILEDATA ("ARCHIVEINFO: DATABASE = UTILITYDB");
```

The following statement produces a report of all files under the directory SYMBOL/X ON PACK that do not have any archive backups:

```
RUN *SYSTEM/FILEDATA ("ARCHIVEINFO: ARCHIVEBACKUP=NONE  
                      DIRECTORY = SYMBOL/X FAMILY = PACK");
```

The following statement, when run under a privileged usercode or started from an ODT, produces a report of all the files on the family WORKPACK that have backup copies on the tape with the serial number X12.

```
RUN *SYSTEM/FILEDATA ("ARCHIVEINFO: BACKUPSN = X12 FAMILY = WORKPACK");
```

Note that the report includes both resident and nonresident files. If tape X12 was originally produced by a WFL *ARCHIVE BACKUP* or *ARCHIVE ROLLOUT* statement for the family WORKPACK, then you could use this FILEDATA report to determine whether tape X12 is still needed.

AREASUMMARY Request

The AREASUMMARY request produces a list of files showing the AREASIZE value, the number of uncrunched and crunched rows, the total segments, and the file name. The default output option is SPO.

<areasummary request>

```

— AREASUMMARY — : ————|—————|—————|
                        |<output option>|
                        |FAMILYNAME  = —<family name>|
                        |PACKNAME    |
                        |SIZE      = —<integer>|
                        |SORT      |
  
```

Explanation

SIZE = <integer>

Specifies the minimum number of segments per area of files to be included in the report. For crunched files with just one area allocated, if the number of segments in that area is less than the specified size, the file is reported. If the SIZE clause is not specified, AREASUMMARY reports on all files.

SORT

Causes the output to be sorted in descending order by the size of the full areas. Crunched files with just one area allocated are sorted by the last area. The potential size of full areas in such files is reported in parentheses.

FILEDATA Modifiers

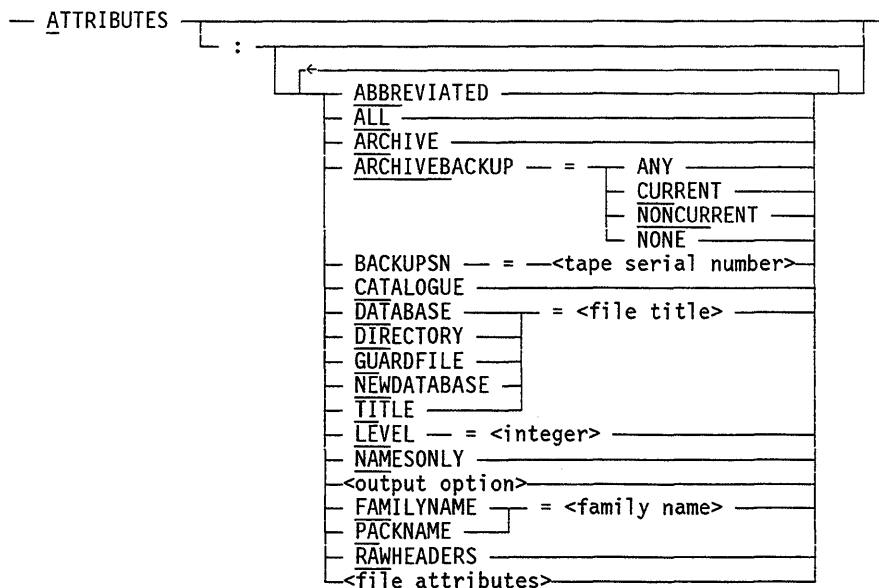
The following modifiers are used to specify the disk family to be reported on, as discussed under “FILEDATA Modifiers” later in this section.

- FAMILYNAME
- PACKNAME

ATTRIBUTES Request

The ATTRIBUTES request produces a report about various requested attributes of a file or group of files. The output is sent to the printer by default. The ATTRIBUTES request performs many of the same functions as the CODEFILEINFO request.

<attributes request>



Explanation

FILEDATA Modifiers

The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be reported on, and specifying the types of information to be reported for each file. These modifiers are explained under "FILEDATA Modifiers" later in this section.

- ABBREVIATED
- ALL
- ARCHIVE
- CATALOGUE
- DATABASE
- DIRECTORY
- FAMILYNAME
- LEVEL
- NAMESONLY
- NEWDATABASE

- RAWHEADERS
- TITLE

<file attributes>

FILEDATA reports information only for the file attributes you request. The following table lists the available attributes and their abbreviated names. The file attributes are explained in “FILEDATA Modifiers” later in this section.

Attribute Name	Minimum Abbreviation
ALTERDATE	ALT
AREALENGTH	AREAL
AREAS	AREAS
AREASECTORS	AREASE
AREASIZE	AREASI
BLOCKSIZE	BL
BLOCKSTRUCTURE	BLOCKST
CCSVERSION	CCSV
CODEVERSION	CODEV
CREATIONDATE	CRE
CRUNCHED	CRU
CYCLE	CY
DOCUMENTTYPE	DOC
EXTMODE	EXE
FILEKIND	FILEK
FILELENGTH	FILEL
FILEORGANIZATION	FILEO
FILESTRUCTURE	FILEST
FILETYPE	FILET
FRAMESIZE	FRA
IDENTITY	ID
INTMODE	IN
LASTACCESSDATE	LASTA
LASTRECORD	LASTR
LICENSEKEY	LIC
LOCKEDFILE	LOCK
MAXRECSIZE	MA
MINRECSIZE	MI
NOTE	NOTE
PERMITTEDACTIONS	PER

continued

FILEDATA Utility

continued

Attribute Name	Minimum Abbreviation
RELEASEID	RE
SAVEFACTOR	SA
SECTORSIZE	SECT
SECURITY	SE
TIMESTAMP	TIM
TOTALSECTORS	TOTA
UNITS	UN
USERINFO	US
VERSION	V
WARNINGS	WARN

ATTRIBUTES Example

The following command initiates an ATTRIBUTES report on all files under the directory ACCOUNTS. The report includes the timestamp, creation date, number of the last record, and the version number of each file. The output is sent to the printer by default.

```
RUN *SYSTEM/FILEDATA ("ATTRIBUTES: DIR = ACCOUNTS  
TIMESTAMP CREATIONDATE LASTRECORD VERSION")
```

The following command produces a report of all the files under the directory SYMBOL/X ON PACK that do not have any archive backups. The report includes the ALTERDATE, CREATIONDATE, SAVEFACTOR, and TIMESTAMP of those files.

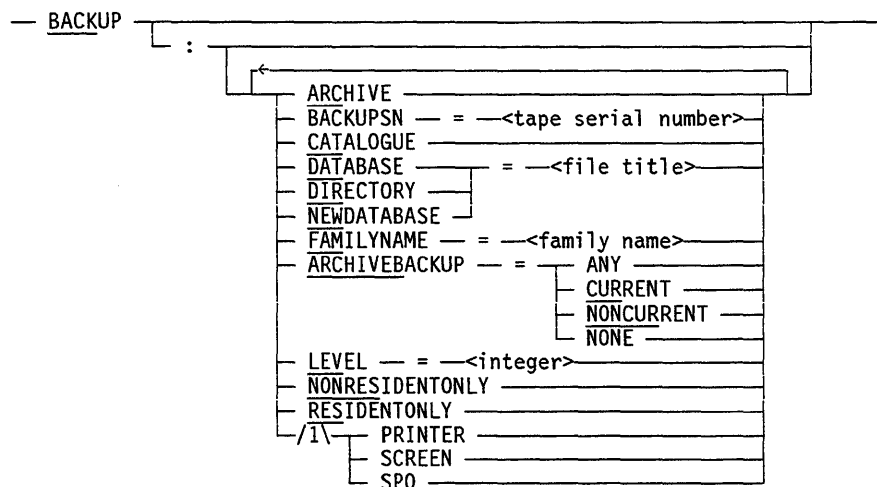
```
RUN *SYSTEM/FILEDATA ("ATTRIBUTES: ARCHIVEBACKUP=NONE,  
DIRECTORY=SYMBOL/X, FAMILYNAME=PACK,  
ALTERDATE, CREATIONDATE,SAVE, TIMESTAMP");
```

BACKUP Request

The BACKUP request reports the usage information of backup library maintenance tapes for cataloged and archived disk files. You can use this report to determine whether library maintenance backup tapes contain active backups for a specified disk family or directory.

The BACKUP request produces a printed report of backup tapes for the specified disk family or for the files in a specified directory of the disk family. The report lists the tapes that contain active backup copies of the disk files according to the SERIALNO value and the number of active backup files for each serial number. For archive backup tapes, the report also includes the name of each tape, the date and time the tape was written, and any abnormal status for the tape.

<backup request>



Explanation

BACKUP

You can specify that the report is to be for the backup tapes of a specific file or directory of files by specifying *DIRECTORY* = <filename> or *DIRECTORY* = <directory name>.

The default value for this request reports both catalog and archive tape information. If you specify CATALOGUE and not ARCHIVE or ARCHIVEBACKUP, then only catalog backup tape information is reported. If you specify ARCHIVE or ARCHIVEBACKUP and not CATALOGUE, then only archive information is reported.

Both resident and nonresident files are reported unless the RESIDENTONLY or NONRESIDENTONLY modifier is specified. For ARCHIVE tapes, the report includes tapes for files that are resident or nonresident files that have backup copies whose generation and timestamp values match those values for the resident files.

FILEDATA Utility

If you specify ARCHIVEBACKUP = CURRENT, then FILEDATA reports on all tapes that include archive backup copies of nonresident files or of resident files whose generation and timestamps match those of their backup copy.

If you specify ARCHIVEBACKUP = NONCURRENT, then FILEDATA reports on all tapes that include archive backup copies for resident files with generations and timestamps that do not match those of their backup copies.

If you specify a tape serial number using the BACKUPSN modifier, then only the files on that tape are reported.

The PRINTER, SCREEN, and SPO options have the effects described under "Output Options" earlier in this section.

FILEDATA Modifiers

The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be reported on, and specifying the types of information to be reported for each file. Further information about these modifiers is provided under "FILEDATA Modifiers" later in this section.

- ARCHIVE
- ARCHIVEBACKUP
- BACKUPSN
- CATALOGUE
- DATABASE
- DIRECTORY
- FAMILYNAME
- LEVEL
- NEWDATABASE
- NONRESIDENTONLY
- RESIDENTONLY

BACKUP Examples

The following statement produces a report of all the archive backup tapes for the user UC on the family DISK. The tapes reported for this statement might also contain backup copies of files that do not belong to the user UC. However, the FILE COUNT reported for the tapes does not include those files.

```
RUN *SYSTEM/FILEDATA ("BACKUP: ARCHIVE DIRECTORY=USERCODE/UC");
```

The following statement, if run by a privileged user or submitted from an ODT, produces a report of all the archive and catalog backup tapes for all the files on the disk family named XPACK:

```
RUN *SYSTEM/FILEDATA ("BACKUP: FAMILYNAME=XPACK");
```

The following statement, if run by a privileged user or submitted from an ODT, produces a report of all the archive backup tapes for all the files on the family DISK for which there is not a resident file on the family. Some of the tapes reported for this example might also contain backup copies of resident files, but the FILE COUNT reported for the tapes does not include those files.

```
RUN *SYSTEM/FILEDATA ("BACKUP: NONRESIDENT");
```

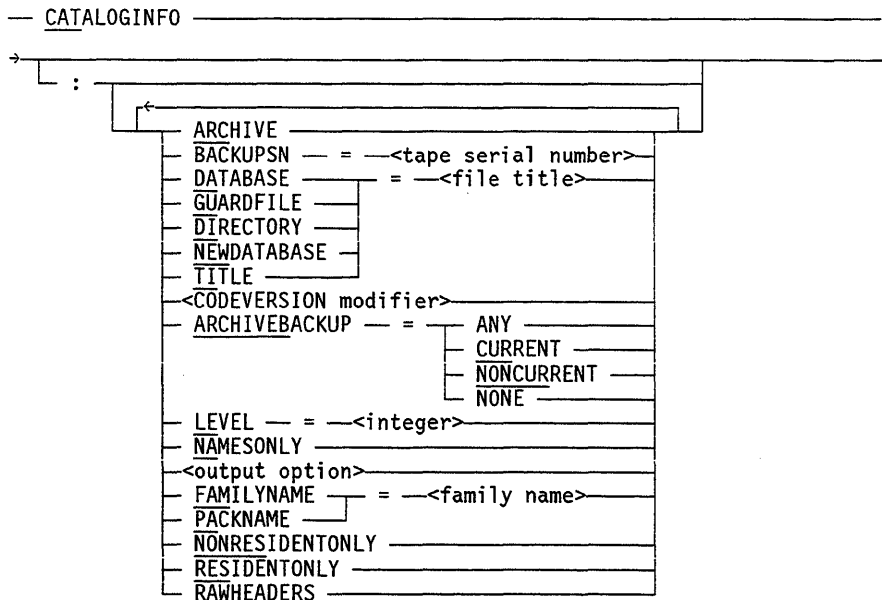
You can use the following statement to determine if any files under the directory SYMBOL on the family PACK were backed up on a tape with the serial number X12:

```
RUN *SYSTEM/FILEDATA ("BACKUP: DIRECTORY=SYMBOL FAMILYNAME = PACK  
BACKUPSN = X12");
```


CATALOGINFO Request

The CATALOGINFO request produces a hierarchical list of files, including the access and creation dates, the size in segments, the security class, the status, and the file kind. The CATALOGINFO request produces the same output as the FILENAMES request with the CATALOGUE modifier. Refer to "FILENAMES Request" in this section for more information. The output is sent to the printer by default.

<cataloginfo request>



Explanation

The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be reported on, and specifying the types of information to be reported for each file. These modifiers are explained under "FILEDATA Modifiers" later in this section.

- ARCHIVE
- ARCHIVEBACKUP
- BACKUPSN
- CODEVERSION
- DATABASE
- DIRECTORY
- FAMILYNAME
- GUARDFILE
- LEVEL
- NAMESONLY

- NEWDATABASE
- NONRESIDENTONLY
- PACKNAME
- RAWHEADERS
- RESIDENTONLY
- TITLE

CATALOGINFO Example

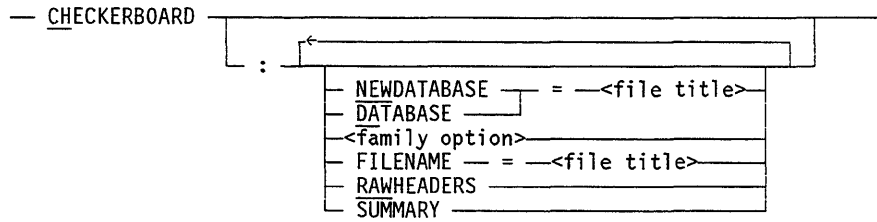
The following example produces a CATALOGINFO report on the file ACCOUNTS and all files in the directory ACCOUNTS/=.

```
RUN *SYSTEM/FILEDATA ("CATALOGINFO: TITLE = ACCOUNTS");
```

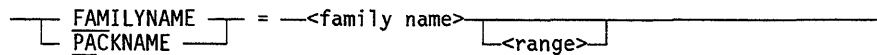
CHECKERBOARD Request

The CHECKERBOARD request produces a disk checkerboard displaying permanent files and the space around them. The output includes the family index, the base address, the end address, the length, the area, the file name, and the space between rows.

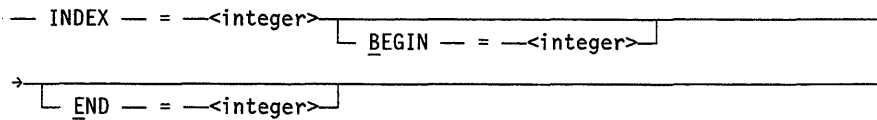
<checkerboard request>



<family option>



<range>



Explanation

FILENAME = <file title>

Causes the output to be saved as a disk file instead of being printed.

Note: If you use the FILENAME output option with CHECKERBOARD, then you should precede the CHECKERBOARD request with a DEFINEOUTPUT request that sets LINEWIDTH = 132. Otherwise, the record size defaults to 80 bytes, which is too short for a CHECKERBOARD report.

<range>

Specifies an address range to list only the files in that range. A family name and a family index must also be supplied.

SUMMARY

Suppresses the listing of the location of each file and available area. Only file conflicts, the total in-use and available sectors counts, and the summary graph of available and in-use areas are reported.

FILEDATA Modifiers

The following FILEDATA modifiers perform functions such as selecting the disk family to be reported on, and specifying the types of information to be reported for each file. These modifiers are explained under "FILEDATA Modifiers" later in this section.

- DATABASE
- FAMILYNAME
- NEWDATABASE
- RAWHEADERS

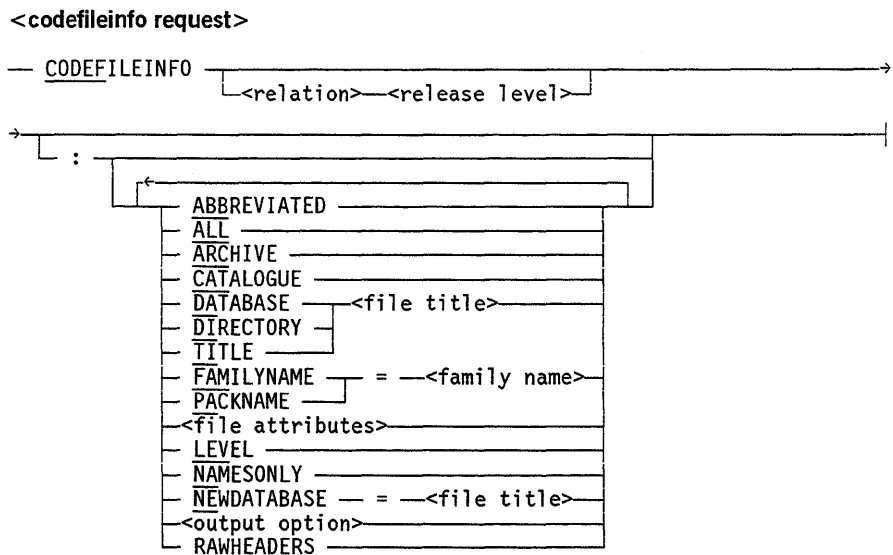
CHECKERBOARD Example

The following command produces disk checkerboard information on all the files under the family MYPACK:

```
RUN *SYSTEM/FILEDATA("CHECKERBOARD: FAMILYNAME = MYPACK")
```

CODEFILEINFO Request

The CODEFILEINFO request produces a report about various requested attributes of a code file or group of code files. The output is sent to the printer by default. The CODEFILEINFO request performs many of the same functions as the ATTRIBUTES request.



Explanation

<relation> <release level>

Reports on each code file whose version number bears the indicated relationship to the given release level.

FILEDATA Modifiers

The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be reported on, and specifying the types of information to be reported for each file. These modifiers are explained under "FILEDATA Modifiers" later in this section.

- ABBREVIATED
- ALL
- ARCHIVE
- CATALOGUE
- DATABASE
- DIRECTORY
- LEVEL

- NAMESONLY
- NEWDATABASE
- PACKNAME
- RAWHEADERS
- TITLE

<file attributes>

FILEDATA reports information only for the file attributes you request. The following table lists the available attributes and their abbreviated names. The file attributes are explained in "FILEDATA Modifiers" later in this section.

Attribute Name	Minimum Abbreviation
ALTERDATE	ALT
AREALENGTH	AREAL
AREAS	AREAS
AREASECTORS	AREASE
AREASIZE	AREASI
BLOCKSIZE	BL
BLOCKSTRUCTURE	BLOCKST
CCSVERSION	CCSV
CODEVERSION	CODEV
CREATIONDATE	CRE
CRUNCHED	CRU
CYCLE	CY
DOCUMENTTYPE	DOC
EXTMODE	EXE
FILEKIND	FILEK
FILELENGTH	FILEL
FILEORGANIZATION	FILEO
FILESTRUCTURE	FILEST
FILETYPE	FILET
FRAMESIZE	FRA
IDENTITY	ID
INTMODE	IN
LASTACCESSDATE	LASTA
LASTRECORD	LASTR
LICENSEKEY	LIC
LOCKEDFILE	LOCK
MAXRECSIZE	MA

continued

FILEDATA Utility

continued

Attribute Name	Minimum Abbreviation
MINRECSIZE	MI
NOTE	NOTE
PERMITTEDACTIONS	PER
RELEASEID	RE
SAVEFACTOR	SA
SECTORSIZE	SECT
SECURITY	SE
TIMESTAMP	TIM
TOTALSECTORS	TOTA
UNITS	UN
USERINFO	US
VERSION	V
WARNINGS	WARN

CODEFILEINFO Examples

The following command initiates a CODEFILEINFO report on all code files under the directory MYFILES whose version numbers are Mark 3.8 or earlier. The report includes the timestamp, number of the last record, and the version number of each file. The output is sent to the printer by default.

```
RUN *SYSTEM/FILEDATA ("CODEFILEINFO LEQ 3.8 : DIR = MYFILES  
TIMESTAMP LASTRECORD VERSION");
```

The following command initiates a CODEFILEINFO report on all code files under the directory USER whose version number is Mark 3.6 or earlier. The report includes the code version (with code file privileges and status such as MC, PP, nonexecutable or UNSAFE) and security (such as SECURITYTYPE, SECURITYUSE, or guard file title if applicable) of each file. Refer to the description of CODEVERSION and SECURITY modifiers under "FILEDATA Modifiers" in this section for more details.

```
RUN *SYSTEM/FILEDATA("CODEFILEINFO LEQ 3.6 : DIR = (USER)  
CODEVERSION SECURITY")
```

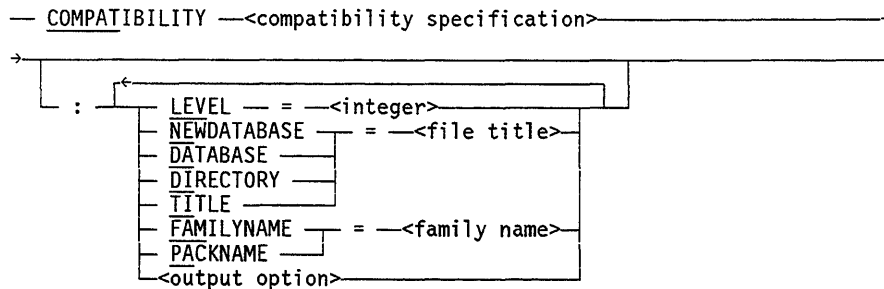
The following output is sent to the printer by default:

```
(USER)
. TEST
. . OBJECT
. . . AL
. . . . HELLO : CODEVERSION=36 EXECUTABLE
                SECURITY=GUARDED BY *TEST/GUARD ON DISK.
. . . TADS
. . . . TEST : CODEVERSION=36 TADS-CAPABLE EXECUTABLE
                SECURITY=PRIVATE (I/O)
. . . TEST
. . . . UNSAFE : CODEVERSION=36 NON-EXECUTABLE:UNSAFE
                SECURITY=PRIVATE (I/O)
. SYSTEM
. . OBJECT
. . . POINTER : CODEVERSION=35 MC-ED EXECUTABLE
                SECURITY=CONTROLLED BY (USER)TEST/GUARD2 ON MYPACK.
```

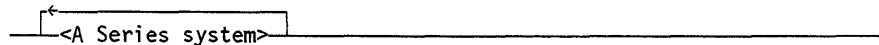

COMPATIBILITY Request

The COMPATIBILITY request produces a report indicating compatibility with a specified host or set of hosts for all code files in a given directory, with a given title, or on a given pack. The report includes a list of systems and MCP levels compatible with each request.

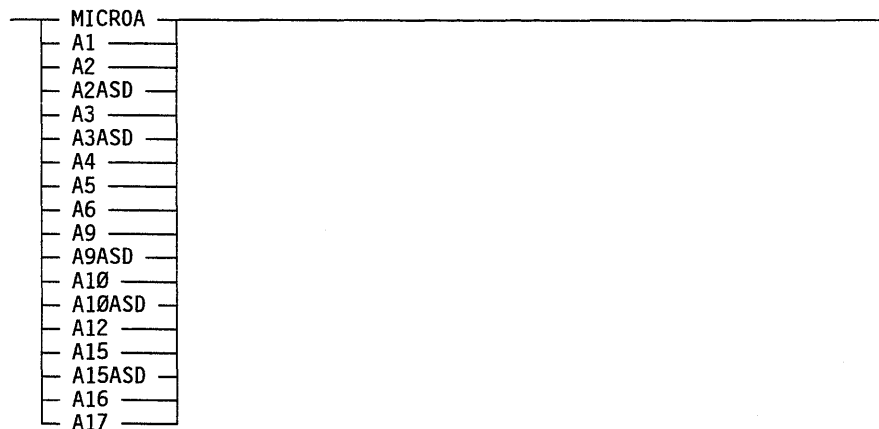
<compatibility request>



<compatibility specification>



<A Series system>



Explanation

COMPATIBILITY <compatibility specification>
 COMPATIBILITY <compatibility specification>:

Produce a report of code files that are compatible with the systems included in the compatibility specification. The compatibility report contains the following information on each code file:

- FILEKIND
- Mark level

- Cycle number
- Compatible systems

FILEDATA Modifiers

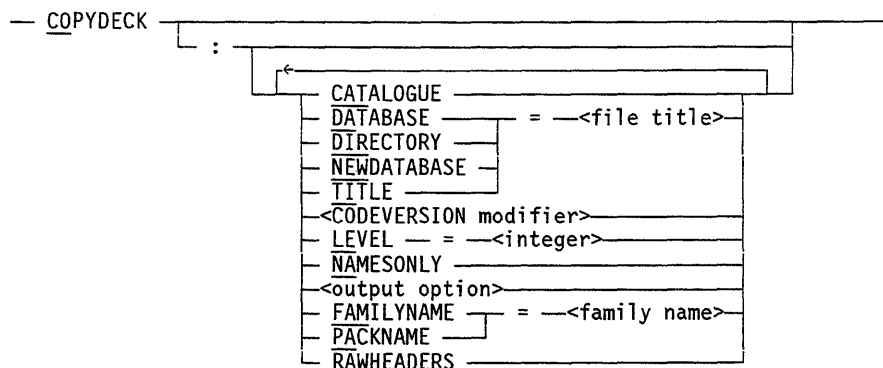
The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be reported on, and specifying the types of information to be reported for each file. These modifiers are explained under "FILEDATA Modifiers" later in this section.

- DATABASE
- DIRECTORY
- LEVEL
- NEWDATABASE
- PACKNAME
- TITLE

COPYDECK Request

The COPYDECK request causes the printing or punching of a COPY&COMPARE card deck (with no FROM or TO specifications), which can then be modified and used as a library maintenance statement. The output is sent to the card punch by default.

<copydeck request>



Explanation

The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be reported on, and specifying the types of information to be reported for each file. These modifiers are explained under "FILEDATA Modifiers" later in this section.

- CATALOGUE
- CODEVERSION
- DATABASE
- DIRECTORY
- FAMILYNAME
- LEVEL
- NAMESONLY
- NEWDATABASE
- RAWHEADERS
- TAPE
- TITLE

COPYDECK Example

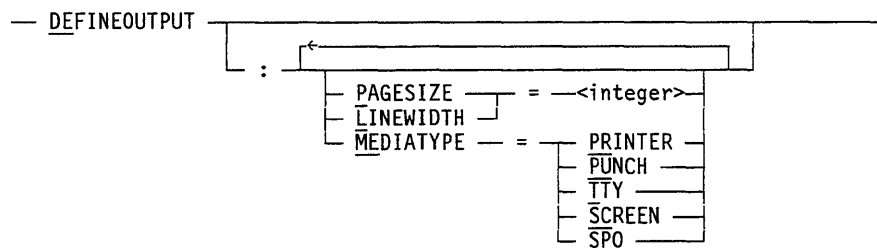
The following example punches a copydeck that includes all the files on the family MYPACK:

```
RUN *SYSTEM/FILEDATA ("COPYDECK: FAMILYNAME = MYPACK");
```

DEFINEOUTPUT Request

The DEFINEOUTPUT request formats a report's output. This request permits you to explicitly control line width, page size, and output medium. The DEFINEOUTPUT request applies to all task requests that appear after the DEFINEOUTPUT request in the FILEDATA statement. If one of the subsequent task requests specifies its own output parameters (as the ATTRIBUTES request can with the options PRINTER or SCREEN), then those output parameters apply only to that task request.

<defineoutput request>



Explanation

MEDIATYPE

Specifies the output device. For an explanation of the device options, refer to "Output Options" earlier in this section.

FILEDATA Modifiers

The following modifiers are used to modify the output. Refer to "FILEDATA Modifiers" later in this section.

- PAGESIZE
- LINEWIDTH

CHECKERBOARD Example

This example generates a CHECKERBOARD report and saves it in a disk file titled CB/REPORT ON PACK. The DEFINEOUTPUT request is used to specify a 132-byte record size, which is the minimum needed for a CHECKERBOARD report.

```

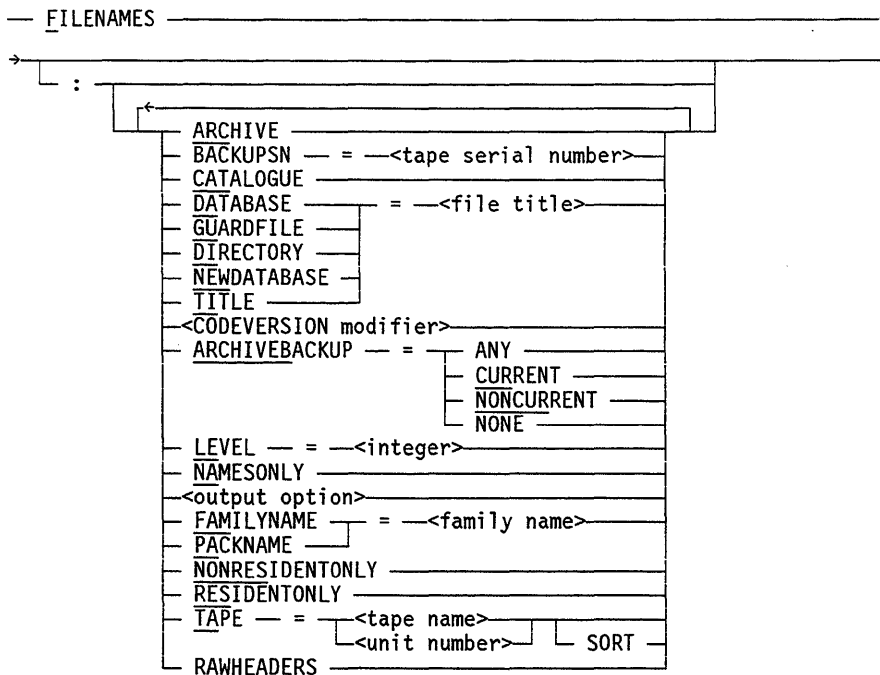
RUN *SYSTEM/FILEDATA ("DEFINEOUTPUT: LINEWIDTH=132;
CHECK: FAMILY=DBFAM, FILENAME=CB/REPORT ON PACK")
  
```

FILENAMES Request

The FILENAMES request produces a hierarchical list of files, including access and creation dates, size in segments, security class, status, and file kind. The output is sent to the printer by default.

If the output is written to a disk file, the LINEWIDTH value specified in the DEFINEOUTPUT request must be 132.

<file names request>



Explanation

TAPE = <tape name>
 TAPE = <unit number>

The TAPE = <tape name> and TAPE = <unit number> modifiers cause FILEDATA to report on the files on the specified library maintenance tape. Multiple reel tapes function best when the unit number is specified. The TAPE modifier overrides the DATABASE, DIRECTORY, and FAMILYNAME modifiers.

SORT

Causes FILEDATA to sort the file names from the specified tape in the same order that files on disk are reported.

FILEDATA Modifiers

The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be reported on, and specifying the types of information to be reported for each file. These modifiers are explained under "FILEDATA Modifiers" later in this section.

- ARCHIVE
- ARCHIVEBACKUP
- BACKUPSN
- CATALOGUE
- CODEVERSION
- DATABASE
- DIRECTORY
- FAMILYNAME
- GUARDFILE
- LEVEL
- NAMESONLY
- NEWDATABASE
- NONRESIDENTONLY
- RESIDENTONLY
- RAWHEADERS
- TITLE

FILENAMES Examples

The following example produces a FILENAMES report on all the files specified in the file UTILITYDB. UTILITYDB is a file created by the NEWDATABASE modifier in a previous FILEDATA run.

```
RUN *SYSTEM/FILEDATA ("FILENAMES: DATABASE = UTILITYDB");
```

The following example produces a report of all the nonresident files for the family PACK under the usercode UCX that have archive backup copies, catalog backup copies, or both on tape:

```
RUN *SYSTEM/FILEDATA ("FILES: DIR=USERCODE/UCX  
FAMILY=PACK NONRESIDENTONLY")
```

FILEDATA Utility

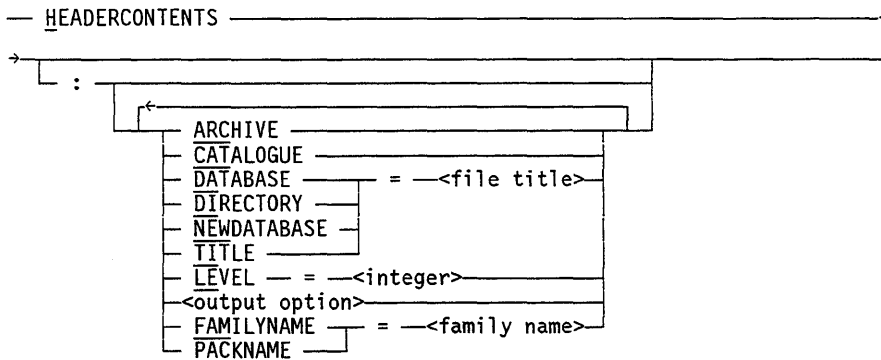
The following example produces a FILENAMES report on the file MYFILES. The report is written to the disk file FDREPORT and is formatted with a line width of 132 characters.

```
RUN *SYSTEM/FILEDATA ("DEFINEOUTPUT:LINEWIDTH=132;  
FILENAMES:TITLE=MYFILES FILENAME=FDREPORT");
```

HEADERCONTENTS Request

The HEADERCONTENTS request produces a hexadecimal dump of file headers, row address words, GUARDED or CONTROLLED attribute security information and, optionally, catalog and archive record information. The output is sent to the printer by default.

<headercontents request>



Explanation

The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be reported on, and specifying the types of information to be reported for each file. These modifiers are explained under "FILEDATA Modifiers" later in this section.

- ARCHIVE
- CATALOGUE
- DATABASE
- DIRECTORY
- FAMILYNAME
- LEVEL
- NEWDATABASE
- PACKNAME
- TITLE

HEADERCONTENTS Example

The following command produces a hexadecimal display of the disk file header for the file (CAD)R/LIST and for any files under that directory on the family XPACK:

```

RUN *SYSTEM/FILEDATA ("HEADERCONTENTS: FAMILYNAME=XPACK
                      DIRECTORY=(CAD)R/LIST")
  
```


HEADERCONTENTS Considerations for Use

The HEADERCONTENTS request can be performed successfully only on a database containing raw headers. The database used for the request contains raw headers if either of the following conditions is true:

- The database was created for a previous HEADERCONTENTS request, or is newly created for this HEADERCONTENTS request.
- The RAWHEADERS modifier was included in the request that created the database currently in use.

The circumstances that control whether FILEDATA reuses a database for a request or creates a new database for the request are discussed under “Database Generation and Reuse” earlier in this section.

When a HEADERCONTENTS request that does not create a new database encounters a database that does not contain raw disk file headers, one of following two error messages is printed:

- If the database used by the program was created by an earlier run of FILEDATA, then this message is printed:

THE DATABASE SPECIFIED DOES NOT CONTAIN RAW HEADERS

- If the database was created during the same run by an earlier request to FILEDATA, then this message is printed:

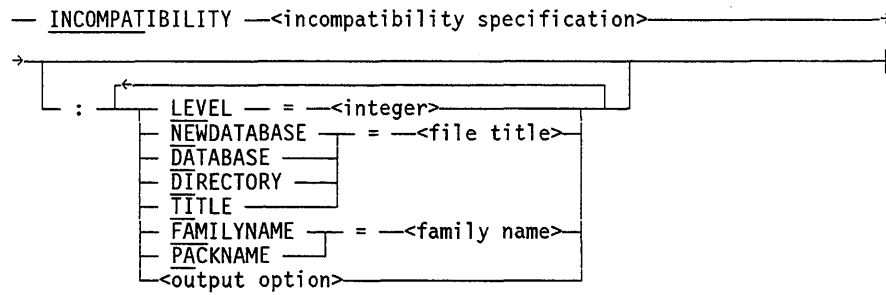
THE DATABASE CREATED BY AN EARLIER REQUEST DOES NOT CONTAIN RAW HEADERS

To correct this situation, rerun FILEDATA with the RAWHEADERS modifier in the last database-creating request prior to the HEADERCONTENTS request, or with just the HEADERCONTENTS request on the same files.

INCOMPATIBILITY Request

The INCOMPATIBILITY request gives information on all code files in a given directory with a given TITLE, or on a given FAMILYNAME.

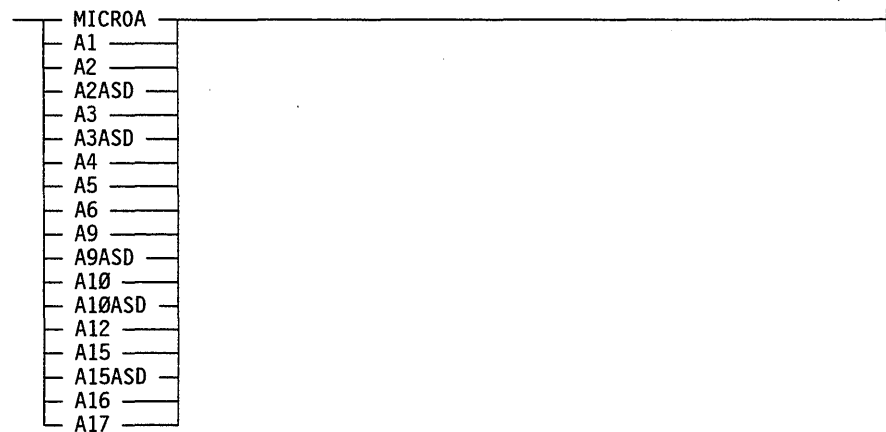
<incompatibility request>



<compatibility specification>



<A Series system>



Explanation

INCOMPATIBILITY <incompatibility specification>
 INCOMPATIBILITY <incompatibility specification>:

Produce a report of code files that are not compatible with the systems included in the incompatibility specification. The incompatibility report contains the following information on each code file:

- FILEKIND
- mark level

FILEDATA Utility

- cycle number
- incompatible systems

FILEDATA Modifiers

The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be reported on, and specifying the types of information to be reported for each file. These modifiers are explained under "FILEDATA Modifiers" later in this section.

- DATABASE
- DIRECTORY
- LEVEL
- NEWDATABASE
- PACKNAME
- TITLE

The following is an example of the output for the above command.

```
*SYSTEMCOMS/PROCESSING/ITEMS : DCALGOLCODE  37.114
*SYSTEMCOMS/STATISTICS/FORMATS : ALGOLCODE  37.112
*SYSTEMMARCUS/FORMATS/BACKUP : DCALGOLCODE  37.263
*SYSTEMSIMPLEINSTALL/SCREENS : DCALGOLCODE  37.141
*SYSTEMSIMPLEINSTALL/SCREENS/ENGLISH : ALGOLCODE  37.212
```

NOREPORTS Request

The NOREPORTS request generates a new database without generating any reports. The database produced by FILEDATA contains file information gathered from the disk directory for the family and files you specify. This new database can then be used as the database in future runs of FILEDATA, thus making it unnecessary for FILEDATA to gather file information from scratch. To use a database produced by a NOREPORTS request in a previous FILEDATA run, specify the DATABASE option in the task request.

<noreports request>

```

-- NOREPORTS -- : -- NEWDATABASE -- = --<file title>----->
|
|----->
| ARCHIVE -----> = --<file title>----->
| CATALOGUE ----->
| DATABASE ----->
| DIRECTORY ----->
| TITLE ----->
|----->
| FAMILYNAME -----> = --<family name>----->
| PACKNAME ----->
| TAPE -----> = --<tape name>----->
|                   |<unit number>----->
|----->
| RAWHEADERS ----->
| WARNINGS ----->

```

Explanation

TAPE = <tape name>

TAPE = <unit number>

Causes FILEDATA to report on the tape with the specified tape name or unit number.

FILEDATA Modifiers

The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be stored in the new database, and specifying the types of information to be stored for each file. These modifiers are explained under "FILEDATA Modifiers" later in this section.

- ARCHIVE
- CATALOGUE
- NEWDATABASE
- DATABASE
- DIRECTORY
- TITLE
- FAMILYNAME
- PACKNAME

- TAPE
- RAWHEADERS

NOREPORTS Example

In the following examples, the first statement creates a database file containing information about files on the family XPACK. This information is then used in the subsequent runs to display information about code files that were on XPACK.

```
RUN *SYSTEM/FILEDATA("NOREPORTS: NEWDATABASE = FD/XPACK  
FAMILY = XPACK");
```

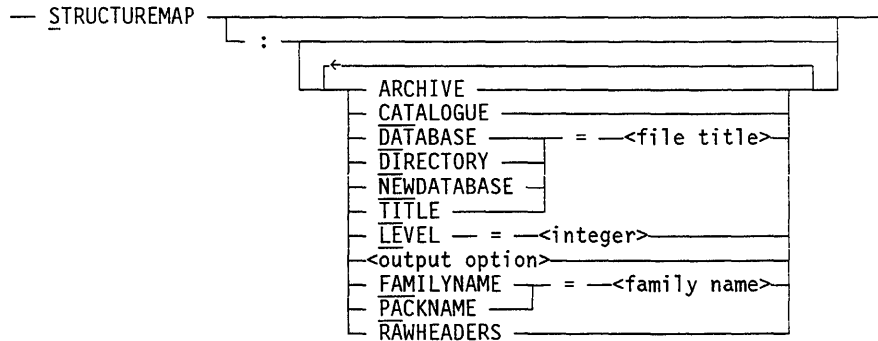
```
RUN *SYSTEM/FILEDATA("CODEFILEINFO: DATABASE=FD/XPACK CODEVERSION");
```

```
RUN *SYSTEM/FILEDATA("COMPATIBILITY A15: DATABASE=FD/XPACK");
```

STRUCTUREMAP Request

The STRUCTUREMAP request produces a map showing file storage layout by family index and address. The report contains the file name, the areas, the area class, the family index, the segment address, the size in segments, and the number of segments on the family. Output is sent to the printer by default.

<structuremap request>



Explanation

The following FILEDATA modifiers perform functions such as selecting the disk family or file names to be reported on, and specifying the types of information to be reported for each file. These modifiers are explained under “FILEDATA Modifiers” later in this section.

- ARCHIVE
- CATALOGUE
- DATABASE
- DIRECTORY
- FAMILYNAME
- LEVEL
- NEWDATABASE
- PACKNAME
- RAWHEADERS
- TITLE

STRUCTUREMAP Example

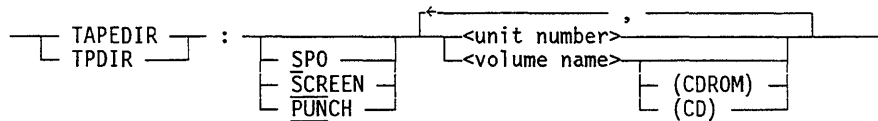
The following command generates a STRUCTUREMAP request on all files and directories on MYPACK with one or two level file names. The output is sent to a remote terminal.

```
RUN *SYSTEM/FILEDATA("STRUCTUREMAP:  
LEVEL = 2 SCREEN FAMILYNAME = MYPACK")
```

TAPEDIR Request

The TAPEDIR request reads library maintenance tape or CD-ROM directories and prints the volume name, the unit number, the serial number, the creation date, the tape type, and a list of the disk file names that were copied to that volume. The TAPEDIR request can be requested only by a privileged user because security usercode information is accessible. TAPEDIR and TPDIR are synonyms.

<tapedir request>



TAPEDIR Examples

The following command produces a FILEDATA report on the library maintenance tape MYTAPE:

```
RUN *SYSTEM/FILEDATA ("TAPEDIR : MYTAPE")
```

The following command produces a FILEDATA report on the tape mounted on unit number 15:

```
RUN *SYSTEM/FILEDATA ("TAPEDIR : 15")
```


FILEDATA Modifiers

Modifiers specify options for task requests. Each task request permits a different set of modifiers. The modifiers apply to all reports until they are overridden by the specification of another modifier.

The FILEDATA modifiers are explained in the following text.

ABBREVIATED

Causes the titles of the requested attributes AREALENGTH is output as ALEN when ABBREVIATED is specified.

ALL

Displays all the attributes of the specified files, except FILETYPE, UNITS, and USERINFO. Even though FILETYPE and UNITS are not displayed, equivalent information is available as part of the display of the BLOCKSTRUCTURE and FRAMESIZE attribute values.

ALTERDATE

Displays the date and time that the specified file was last written.

ARCHIVE

Causes FILEDATA to select resident and nonresident disk files and to display archive information for those files.

When the ARCHIVE modifier is used with the ATTRIBUTES request, FILEDATA displays the backup tape names, serial numbers, and backup times for each resident or nonresident file selected that has archive backup information.

When the ARCHIVE modifier is used with the CODEFILEINFO request, FILEDATA displays the backup tape names, serial numbers, and backup times for each resident code file selected that has archive backup information.

When the ARCHIVE modifier is used with the HEADERCONTENTS request, FILEDATA displays the hexadecimal value of the archive record for each resident or nonresident file selected that has archive backup information.

When the ARCHIVE modifier is used with the NOREPORTS request, FILEDATA collects the archive backup information for files and puts the information into the database so that later requests can report the information.

ARCHIVEBACKUP

If you specify ARCHIVEBACKUP = ANY, then FILEDATA reports only those files that have archive backups. FILEDATA reports on files even if their backups do not match the resident file or even if there is not a resident file.

If you specify ARCHIVEBACKUP = NONE, then FILEDATA reports information only for those files that do not have archive backup copies listed in the directory.

If you specify ARCHIVEBACKUP = CURRENT, then FILEDATA reports information for nonresident files that have archive backup copies and resident files with generation and timestamp values that match the values of their archive backup copy.

If you specify ARCHIVEBACKUP = NONCURRENT, then FILEDATA reports only on resident files whose archive backup copies have a generation or timestamp value that does not match the values of the resident file.

The reports that FILEDATA produces for ARCHIVEBACKUP = NONCURRENT and ARCHIVEBACKUP = NONE together include all the files that would be copied by the WFL *ARCHIVE INCREMENTAL* statement. These two FILEDATA reports would also include a few special files, such as BADDISK files, that would not be copied by *ARCHIVE INCREMENTAL* statements.

AREALENGTH

Displays the size of an area in the specified file in FRAMESIZE units and also the number of full sized records that can be completely contained within that area. A full sized record is a record whose size is equal to the value of the MAXRECSIZE modifier.

A file with variable length records can contain records that are smaller than the MAXRECSIZE value and can, therefore, contain a larger number of records in an area. The displayed values do not include any unused portions of the area. The specified file can be either uncrunched or crunched.

AREAS

Displays the number of words in the file header allocated for row addresses.

AREASIZE

Displays some of the same information as the AREALENGTH modifier. The value of the AREASIZE file attribute is the second number in the AREALENGTH attribute display. The value displayed is the number of records in the area.

BACKUPSN

The BACKUPSN = <tape serial number> modifier limits the report to files that have an archive or catalog backup copy on the tape with a specified serial number.

The BACKUPSN modifier implies that FILEDATA should retrieve archive or catalog information, or both. If you do not explicitly specify either the ARCHIVE modifier or the CATALOGUE modifier, and if the task request is neither ARCHIVEINFO nor CATALOGINFO, then FILEDATA searches for the tape serial number in both the archive and catalog directories (if available).

BLOCKSIZE

Displays the block size—that is, the length of a block of the specified file in FRAMESIZE units.

BLOCKSTRUCTURE

Displays the format of the records for the specified file. Refer to the *A Series File Attributes Programming Reference Manual* for information about the various BLOCKSTRUCTURE values.

CATALOGUE

Causes FILEDATA to select resident and nonresident files and to display catalogue information for the files.

When the CATALOGUE modifier is used with an ATTRIBUTES request, FILEDATA displays the catalog information for each resident and nonresident file selected.

When the CATALOGUE modifier is used with a CODEFILEINFO request, FILEDATA displays the catalog information for each resident code file selected.

When the CATALOGUE modifier is used with a FILENAMES request or an ARCHIVEINFO request, FILEDATA displays the catalog information for each resident or nonresident file selected. Note that this is the same output as that generated by the CATALOGINFO request.

When the CATALOGUE modifier is used with a HEADERCONTENTS request, FILEDATA displays the catalog information, in hexadecimal format, for each resident and nonresident file selected.

When the CATALOGUE modifier is used with a NOREPORTS request, catalog information for files is collected and put into the new database so that later requests can report the information.

CCSVERSION

Displays the current character set. For more information about the CCSVERSION attribute, refer to the *A Series File Attributes Programming Reference Manual*.

CODEVERSION

Depending on the context in which you use it, the `CODEVERSION` keyword is interpreted by FILEDATA as either the `CODEVERSION` modifier or the `CODEVERSION` file attribute. In all the syntax diagrams in this section, the metatoken `<CODEVERSION modifier>` is used to refer to `CODEVERSION` in its sense as a modifier. Where `CODEVERSION` is used in its sense as a file attribute, the metatoken `<file attributes>` appears in the syntax diagram, and `CODEVERSION` is listed as one of the permitted file attributes in the explanation of the task request. The following paragraphs discuss the `CODEVERSION` modifier and the `CODEVERSION` file attribute separately.

CODEVERSION Modifier

```
— CODEVERSION [ <relation> <release level> ]
```

The simple `CODEVERSION` form of the modifier causes FILEDATA to report only on the code files that will not be executable with the next release.

The `CODEVERSION <relation> <release level>` form of this modifier specifies that FILEDATA is to report on only the code files whose version number bears the indicated relationship to the given release level.

CODEVERSION File Attribute

Displays the version numbers of the code files among the specified files, the version of the compiler that compiled the code file, and the relevant code file privilege or status information, as shown in Table 5-1.

Table 5-1. Code File Status Information

Status	Explanation
MC-ED	A compiler code file, enabled by the MC system command
CPed	A control program, enabled by the CP system command
PPed	A privileged program, enabled by the PP system command
PP:TRANSPARENT	A privileged-transparent program
TADS-CAPABLE	A TADS-capable program
EXECUTABLE	The default for a code file
NON-EXECUTABLE:	A nonexecutable program, contains UNSAFE code
UNSAFE	The program uses constructs, that if not properly used, could damage the system.

The terms `EXECUTABLE` and `NON-EXECUTABLE` refer to the privilege category that allows you to execute a given code file. Normally, a user with the appropriate privilege can run any `EXECUTABLE` code file. A nonprivileged user cannot run a

NON-EXECUTABLE code file. For example, only an operator can apply the CM (Change MCP) system command to an MCP code file, while only a privileged user, a system user, or an operator can apply the SL (Support Library) system command to an unsafe library code file.

CREATIONDATE

Displays the date and time that the specified file was created.

CRUNCHED

Indicates whether the specified file has been closed with CRUNCH; that is, whether the specified file has returned the unused portion of its last area to the system.

CYCLE

Identifies generations of a permanent file. The most current file is indicated by the highest cycle and the highest version of that cycle.

DATABASE

The DATABASE = <file title> modifier causes FILEDATA to obtain information from a database file created in a previous run of FILEDATA instead of gathering the information from a disk or tape directory. A database file can be created by using the NEWDATABASE modifier.

The copy of FILEDATA that creates a database and the copy of FILEDATA that uses the database must be of the same Mark level or compatible Mark levels. If FILEDATA detects that the database file was created by an incompatible version of FILEDATA, then the following error message is printed:

```
DATABASE AND FILEDATA LEVELS ARE NOT COMPATIBLE
```

For the HEADERCONTENTS request to be used on an existing database, the database must contain raw disk file headers; otherwise, the following error message is printed:

```
DATABASE SPECIFIED DOES NOT CONTAIN RAW HEADERS
```

DIRECTORY

The DIRECTORY = <file title> modifier causes FILEDATA to report on the specified file or on all the files in the specified directory. If the file title represents a file and a directory, both are reported on. TITLE and DIRECTORY are synonyms.

DOCUMENTTYPE

Displays the DOCUMENTTYPE file attribute, which enables File Access, Transfer, and Management (FTAM) to detect the type of file being transferred. Refer to the *A Series File Attributes Programming Reference Manual* for a description of the DOCUMENTTYPE file attribute.

EXTMODE

Displays the EXTMODE file attribute of the specified file. Refer to the *A Series File Attributes Programming Reference Manual* for information on the EXTMODE file attribute.

FAMILYNAME

The FAMILYNAME = <family name> modifier changes the source of information from the family named DISK to the named disk pack. The entire pack is used in the report. This modifier overrides the DATABASE, DIRECTORY, TITLE, and TAPE modifiers. FAMILYNAME is the preferred mnemonic for PACKNAME.

FILEKIND

Describes the internal structure and purpose of a record of a disk file. Refer to the *A Series File Attributes Programming Reference Manual* for the various FILEKIND attributes.

FILELENGTH

Displays the size of the specified file in FRAMESIZE units.

FILEORGANIZATION

Describes the organization under which the specified file was opened. FILEORGANIZATION is shown only if the FILEKIND value is greater than or equal to the VALUE(DATA) value.

FILESTRUCTURE

Displays the FILESTRUCTURE file attribute of the specified file. Refer to the *A Series File Attributes Programming Reference Manual* for information on the FILESTRUCTURE file attribute.

FILETYPE

Displays the format of the records and the structure of the specified file. Refer to the *A Series File Attributes Programming Reference Manual* for a description of the various FILETYPE values.

FRAMESIZE

Displays the number of bits transferred as a unit during an I/O procedure to the specified file. Refer to the *A Series File Attributes Programming Reference Manual* for a description of the FRAMESIZE file attribute.

GUARDFILE

The GUARDFILE = <file title> modifier causes the FILENAMES request of FILEDATA to report only on the files guarded or controlled by the specified guard file. Because family substitution is not applied for either the input guard file title or the guard file titles associated with the files, you should provide the exact guard file title, including the usercode or the family name, to ensure proper matching with guarded files.

FILEDATA also checks for existence of the specified guard file. Warning messages such as "GUARDFILE DOES NOT EXIST" and "GUARDFILE IS NOT VISIBLE" are displayed to caution you about potential errors with the requested guard file. Despite such warnings, any files that are guarded or controlled by the specified guard file are still reported.

If no files are found to be guarded by the specified guard file, the output is the following:

```
TOTAL SEGMENTS SHOWN = 0
```

IDENTITY

Displays the IDENTITY attribute of the specified file. The attribute is used in DISPLAY and ACCEPT messages. Use the MP (Mark Program) system command to set the IDENTITY attribute.

INTMODE

Causes the EXTMODE value to be displayed; however, Unisys recommends that you not use this modifier. The INTMODE file attribute is not an attribute for a permanent file; therefore, this attribute cannot be displayed by FILEDATA. Refer to the *A Series File Attributes Programming Reference Manual* for a description of the INTMODE file attribute.

LASTACCESSDATE

Displays the date and time the specified file was last accessed.

LASTRECORD

Displays the record number of the last record. The records are zero-relative; that is, the first record is record 0. The message "LASTRECORD IS UNKNOWN" is displayed for this attribute for files that were not defined with a BLOCKSTRUCTURE attribute value of FIXED.

LEVEL

The LEVEL = <integer> modifier specifies the number of file name levels that should be listed beyond the name given in the DIRECTORY or TITLE parameter. For example, if the files A/B, A/C/D, and A/E/F/G exist, and a FILEDATA report is initiated on DIR = A LEVEL = 2, FILEDATA would report on A/B and A/C/D. It would also show that the directory A/E/F exists, but would not report on the file A/E/F/G. If no DIRECTORY or TITLE value is given, LEVEL specifies the total number of levels to be listed. Usercodes are always counted as a level.

When LEVEL is specified, only information about the files that are shown is listed. Information, such as size in segments, is not listed for files that are not shown and is not included in any totals of such information.

LEVEL does not work if DATABASE or TAPE is specified.

LICENSEKEY

Displays the key used to control any copy operations performed on the file by Library Maintenance. Refer to the *A Series File Attributes Programming Reference Manual* for more information on the LICENSEKEY attribute.

LINEWIDTH

The LINEWIDTH = <integer> modifier defines the length of an output line.

LOCKEDFILE

Indicates whether the LOCKEDFILE attribute has been set to TRUE. When the LOCKEDFILE attribute has been set, the file cannot be removed and the file title cannot be changed unless the attribute value is reset to FALSE. To reset the value, use the WFL ALTER statement or reset the value programmatically. You must request that this value be displayed if the attribute value is not set to TRUE.

MAXRECSIZE

Displays the maximum size of logical records in the specified file.

MINRECSIZE

Displays the minimum size of logical records in the specified file.

NAMESONLY

Causes only the file names to be displayed. Header information is to be neither extracted nor processed in any report.

NEWDATABASE

The NEWDATABASE = <file title> modifier causes FILEDATA to create a permanent disk file that contains the directory information gathered for the request. In subsequent runs, if DATABASE is specified with the name of the file created by NEWDATABASE, FILEDATA retrieves the information from that file rather than gathering the information again from the disk or tape directory.

NONRESIDENTONLY

Specifies that FILEDATA is not to report on any resident files, even if they have archive or catalog backup copies. FILEDATA is to report only on files that are not currently resident on disk but do have at least one archive or catalog backup copy on tape. You can use this report to locate files that you no longer need (the files you might want to issue *WFL ARCHIVE PURGE* or *CATALOG PURGE* statements for).

A request that includes the NONRESIDENTONLY and ARCHIVE modifiers, but not the CATALOGUE modifier, reports on the same files that a *WFL ARCHIVE RESTOREADD* statement would copy from tape to disk.

NOTE

Displays the value, if any, of the NOTE file attribute. For a printer backup file, this is the message to be printed on the banner page that precedes the file if the BANNER attribute has a value of TRUE. Refer to the *A Series File Attributes Programming Reference Manual* for more information on the NOTE file attribute.

PACKNAME

Use of the FAMILYNAME modifier is preferred.

PAGESIZE

The PAGESIZE = <integer> modifier specifies the number of output lines per page.

PERMITTEDACTIONS

Reports the value of the PERMITTEDACTIONS file attribute, which specifies the actions that can be performed on a file through File Transfer, Access, and Management (FTAM). Refer to the *A Series File Attributes Programming Reference Manual* for more information on the PERMITTEDACTIONS file attribute.

RAWHEADERS

When this modifier is used along with a database creating FILEDATA request, FILEDATA includes raw disk file headers in the database it creates. All FILEDATA requests, except HEADERCONTENTS, that create a new database should contain the RAWHEADERS modifier if the database they create is ever to be used as input to a HEADERCONTENTS request. Refer to "HEADERCONTENTS Request" earlier in this section.

RELEASEID

Displays the software release level of a file, if any. Refer to the *A Series File Attributes Programming Reference Manual* for a description of the RELEASEID file attribute.

RESIDENTONLY

Specifies that FILEDATA is not to report on any nonresident files. The report includes resident disk files, regardless of whether the files have archive or catalog backup copies on tape.

SAVEFACTOR

Displays the expiration date of a file in terms of the number of days past the creation date.

SECURITY

Displays the SECURITYTYPE and SECURITYUSE attributes of a file. If the SECURITYTYPE value is GUARDED or CONTROLLED, the SECURITYGUARD file attribute value is also shown. The guard file title displayed for SECURITYGUARD is the exact title of the file that is checked for access restrictions. No family substitution is done when the system searches for the guard file at access check time. Refer to the *A Series File Attributes Programming Reference Manual* for more information on security.

TIMESTAMP

Displays the date and time that the last alteration was made to the file.

TITLE

The **TITLE = <file title>** modifier causes FILEDATA to report on the specified file or on all the files in the specified directory. If the file title represents a file and a directory, both are reported on. **TITLE** and **DIRECTORY** are synonyms.

TOTALSECTORS

Causes FILEDATA to report the **TOTALSECTORS** file attribute value of each file reported. For a description of the **TOTALSECTORS** file attribute, refer to the *A Series File Attributes Programming Reference Manual*.

UNITS

Displays the value of the **UNITS** attribute of the file.

USERINFO

Displays the file attribute **USERINFO** in hexadecimal form. The entire word is displayed. This modifier must be specified explicitly to obtain the **USERINFO** display.

VERSION

Identifies generations of a permanent file. The most current file is indicated by the highest cycle and the highest version of that cycle. Refer to the *A Series File Attributes Programming Reference Manual* for more information.

WARNINGS

Reports the warnings that have been accumulated by a file. The report includes the text of the warnings.

NOREPORTS requests should include the **WARNINGS** modifier if the resulting database is to be used as input to an **ATTRIBUTES** or **CODEFILEINFO** request with the **WARNINGS** modifier specified.

Numeric Report Requests

A numeric report request permits a report to be requested by number. All numeric report requests are implemented as executable statements within SYMBOL/FILEDATA. These requests reduce the amount of input that must be supplied for standard functions such as LISTDIRECTORY. Any number that has been defined in SYMBOL/FILEDATA to represent a particular report can be specified in a numeric report request. Numeric report requests cannot contain modifiers.

A numeric report request can be entered by way of a *VALUE=<numeric report request>* statement or in the regular parameter list. A numeric report request entered by way of a *VALUE=<numeric report request>* statement is performed before the parameter list, if any, is processed. The numeric report requests in the parameter list are treated like any other FILEDATA task request.

Reports for 0, 1, 3, and 5 are currently defined. A number 0 includes the FILEDATA task requests FILENAMES, STRUCTUREMAP, and CHECKERBOARD. A number 1 is equivalent to the FILENAMES task request. A number 3 is equivalent to the CODEFILEINFO task request. A number 5 is equivalent to the COMPILEDECK task request.

<numeric report request>

—*<integer>*—

Example

The following command initiates three FILEDATA reports. The first report is a numeric report request number 0. The second report includes the task request ATTRIBUTES with the modifiers DIRECTORY and ALL. The third request specifies the numeric report request number 1.

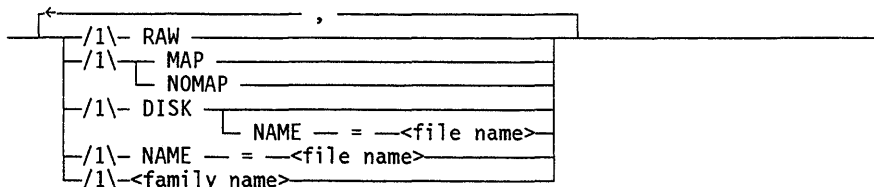
```
RUN SYSTEM/FILEDATA ("0;ATTRIBUTES:DIR=MYSELF ALL; 1 ")
```

Old PACKDIR Syntax

The old PACKDIR syntax is composed of keywords that initiate various FILEDATA reports. The keywords are equivalent to certain FILEDATA task requests.

If the <old packdir syntax> variable is used, all nonalphanumerics except the slash are translated to blanks and hence are treated as delimiters. Therefore, names preceded by an asterisk (*), parenthesized usercodes, and names containing special characters, whether the names are quoted or not, are cannot be used. The file name can represent a directory, a file, or a directory and a file with the same name. In the last case, both the directory name and the file name are reported. A family name can be specified separately in the old packdir syntax. An alpha token that is not one of the keywords indicated in the syntax diagram is treated as a file name; also, the first identifier in a file name cannot be any of those keywords, or a syntax error occurs.

<old packdir syntax>



Explanation

RAW

Equivalent to the HEADERCONTENTS task request.

MAP

Equivalent to the CHECKERBOARD task request.

NOMAP

Suppresses the disk checkerboard information. NOMAP is the default.

DISK

Generates a FILENAMES and STRUCTUREMAP report on all files on the family DISK.

DISK NAME = <file name>

Generates a FILENAMES and STRUCTUREMAP report on all files under the directory specified by the file name value residing on the family DISK.

NAME = <file name>

Generates a FILENAMES and STRUCTUREMAP report for all files under the directory specified by the file name value. For disk packs, the first level of the name must be the family name. No blanks, quotes, or parentheses can occur anywhere in the family name.

<family name>

Indicates on which family the files reside.

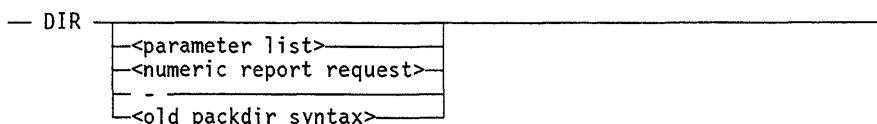
Using System Commands to Initiate FILEDATA

DIR (Directory) and TDIR (Tape Directory) are system commands that initiate SYSTEM/FILEDATA.

DIR (Directory) Command

The DIR system command is used to obtain disk directory information.

<dir command>



Explanation

DIR

Causes FILEDATA to generate a FILENAMES report and a STRUCTUREMAP report for all files on the DISK family.

<parameter list>

Initiates a FILEDATA report by specifying various FILEDATA task requests. The variable parameter list is defined later in this section.

<numeric report request>

This variable is explained later in this section.

DIR-

Invokes numeric report request number 1.

<old packdir syntax>

Uses key words to initiate various FILEDATA reports. This variable is explained later in this section.

Examples

The following command initiates a FILEDATA report on the directory A/B on the family DISK:

```
DIR DISK NAME=A/B
```

The following command requests a report on the UZERPK family:

```
DIR UZERPK RAW
```

Either of the following commands invokes the numeric report request number 1:

```
DIR 1
DIR-
```

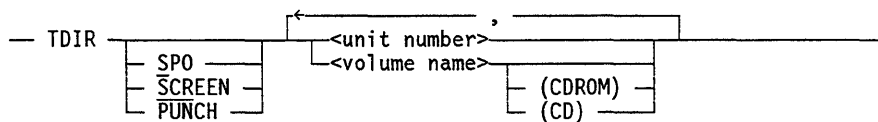
The following command produces three FILEDATA reports. The first report invokes the COPYDECK task request with the CATALOGUE modifier. The second report invokes the CHECKERBOARD task request. The third report invokes the numeric request number 0.

```
DIR COPYDECK: CATALOGUE; CHECKERBOARD; 0
```

TDIR (Tape Directory) Command

The TDIR system command lists the directory of the specified library maintenance tape or tapes, or a library maintenance format CD-ROM.

<tdir command>



Explanation

TDIR <unit number>

TDIR <volume name>

Lists the directory of the tape volume that is mounted on the specified unit or that has the specified tape name.

As many intermixed tape names or unit numbers as desired can be specified. Each must be separated from the next by a comma (,).

The unit number must be used in place of specifying tape name if the *n*th reel of a multireel library dump is required.

If none of the output options PUNCH, SCREEN, or SPO is specified, the output is sent to the printer.

TDIR <unit number>

TDIR <volume name> (CDROM)

Displays the directory of the CD that is loaded on the specified unit or the directory of the CD volume that has the specified name.

SPO

Displays the directory of the specified tape on the originating ODT. This option should be used if the ODT is configured with visible end-of-text (ETX) characters.

SCREEN

Displays the directory of the specified tape on the originating ODT. This option should be used if the ODT is configured with invisible end-of-text (ETX) characters.

PUNCH

Causes the directory of the specified tape to be punched by the card punch.

Section 6

INTERACTIVEXREF Utility

INTERACTIVEXREF Operation

The SYSTEM/INTERACTIVEXREF utility permits interactive access to detailed information about the identifiers declared in a program.

A cross-reference of a program contains an entry for each identifier declared in the program. This entry is referred to as the *header line information*. The header line contains the following information about the identifier:

- The alphabetic name
- The declared type of the identifier
- The program environment
- The stack location
- The sequence number of the declaration

Files Used by the INTERACTIVEXREF Utility

The INTERACTIVEXREF utility obtains information from files generated by the SYSTEM/XREFANALYZER utility.

These files are called *XREFFILES* and have the titles *XREFFILES/<code file name>/DECS* and *XREFFILES/<code file name>/REFS*. The code file name is the name of the code file that was being generated by the compiler when the XREFFILES were created. When compiled through the Command and Edit (CANDE) message control system (MCS), the code file name is normally prefixed by OBJECT followed by a slash (/).

You can produce the XREFFILES by using the compiler control option XREFFILES. When XREFFILES is TRUE, the compiler runs SYSTEM/XREFANALYZER to produce the cross-reference files.

Assigning another compiler control option, XREF, the value SET causes XREFANALYZER to produce only a printed output of the cross-reference information.

When both XREFFILES and XREF are TRUE, both the XREFFILES and a printed output are produced. These options are available in the following compilers:

- ALGOL
- COBOL74

INTERACTIVEXREF Utility

- ESPOL
- FORTRAN
- FORTRAN77
- NEWP
- PASCAL
- RPG

If any syntax errors are encountered, XREFANALYZER is not run.

The XREFFILES can also be produced by running XREFANALYZER with a negative task value. The XREFANALYZER input file *TITLE=XREF/<code file name>* must be specified during initialization.

Version information is included in the XREFFILES. INTERACTIVEXREF checks the version compatibility of the XREFFILES and displays an appropriate error message if the XREFFILES were created with an incompatible XREFANALYZER. These files must be present for INTERACTIVEXREF to run.

Information from the original symbol file used in the compilation is needed by several commands.

Running the INTERACTIVEXREF Utility

To run INTERACTIVEXREF from a terminal, enter the following CANDE command:

```
RUN $SYSTEM/INTERACTIVEXREF
```

This command initializes INTERACTIVEXREF. You must then load the XREFFILES using the LOAD command. The original symbol file used in the compilation is needed only for certain commands and can be loaded by means of the SYMBOL command. The LOAD and SYMBOL commands are explained later in this section.

To load the XREFFILES and the symbol file during initialization enter the following CANDE command:

```
RUN $SYSTEM/INTERACTIVEXREF; FILE LOAD=<load file name>;  
FILE SYMBOL=<symbol file name>
```

Running INTERACTIVEXREF with the SW1 task attribute set to TRUE causes the program to create a LOCS file, if one does not already exist. The LOCS file can be used by the LOADXREF command in DUMPANALYZER, as described in the *A Series System Software Support Reference Manual*.

After INTERACTIVEXREF has been initialized and the XREFFILES have been loaded, you can enter commands to obtain specific information about the identifiers. Commands can be entered one at a time, or multiple commands can be placed on the

same line, separated by semicolons (;). You can continue a command line over two or more lines by ending each line with a percent sign (%) and continuing on the next line.

You can discontinue a command by entering *BREAK*. At that point, any remaining commands on the current input line are ignored.

Caution:

Make sure that the correct symbol file is associated with the correct XREFFILES file. INTERACTIVEXREF cannot recognize an error, and unexpected responses could result.

Input to the INTERACTIVEXREF Utility

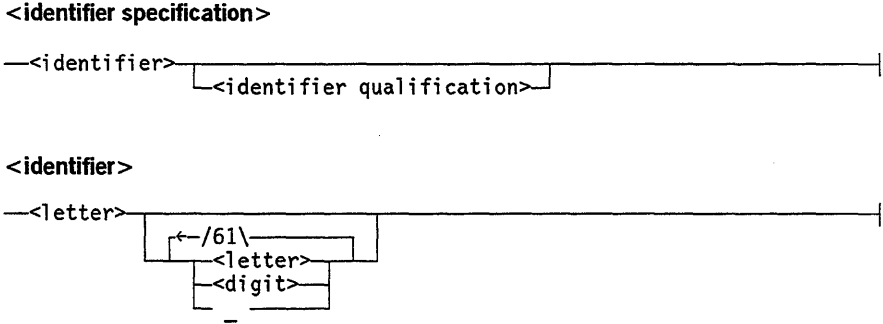
The following text explains the basic constructs and the commands used when running INTERACTIVEXREF.

Basic INTERACTIVEXREF Constructs

The following items commonly appear as syntactic variables in the syntax diagrams in this section.

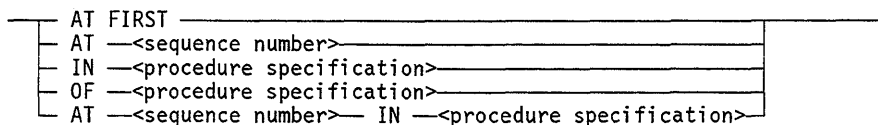
Identifier Specification

An identifier specification defines a particular identifier within a program. If the identifier is redeclared in many different procedures or blocks, you can also indicate the particular procedure or block by specifying an identifier qualification along with the identifier value.



INTERACTIVEXREF Utility

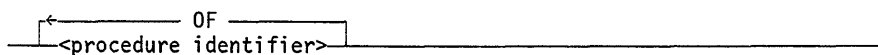
<identifier qualification>



<sequence number>



<procedure specification>



Explanation

<identifier>

An alphanumeric identifier beginning with a letter that is composed entirely of any combination of letters (A through Z), digits (0 through 9), and underscores (_). In an ALGOL program, an identifier of the form B.0002 is acceptable.

<identifier qualification>

Qualifies the occurrence of the identifier that is intended in case the same identifier is redeclared in many different procedures or blocks.

AT FIRST

Selects the first occurrence of the identifier that the compiler encountered.

AT <sequence number>

Selects the occurrence of the identifier declared, or used closest to, the specified sequence number. If an exact match is not found, a warning message is given. The identifier qualification gives undefined results if the symbol file that was loaded is not sequenced properly.

IN <procedure specification>

Causes INTERACTIVEXREF to look for a use of the identifier in the specified procedure. INTERACTIVEXREF looks first for an identifier declared by the specified procedure. If the identifier is not declared by the specified procedure, an identifier declared in a procedure nested within the specified procedure is sought. If this search fails, a global identifier referenced by the procedure is sought. If all these searches fail, an error occurs.

OF <procedure specification>

Selects the occurrence of the identifier declared by the specified procedure. An occurrence of the identifier declared by a procedure contained within the specified procedure is not acceptable.

AT <sequence number> IN <procedure specification>

Selects the occurrence of the identifier declared or referenced closest to the specified sequence number within the specified procedure. This option is useful as an alternative to *AT <sequence number>* when the specified procedure, but not the entire source, is properly sequenced. Refer to “Use with Improperly Sequenced Source” in this section.

<procedure specification>

Specifies a particular procedure. The procedure specification need be only long enough so that its outermost environment—a module or a procedure—is the best candidate of all the possible environments specified by that identifier. The best candidate is defined as follows:

- If only one environment exists and uses the procedure identifier name, it is used.
- If more than one environment exists with this name, the most global environment is used.
- If equally global environments exist with this name, the first environment is used.

If more than one environment exists for a specified name, a warning of possible ambiguity and the name of the chosen environment are displayed.

Examples

The following examples use the example INTERACTIVEXREF program contained at the end of this section.

The following REFERENCE command requests information on the identifier B at line 5600:

```
REF B AT 5600

CLOSEST MATCH: B @ 00005400
B :: REAL @ (2,2) :: DECLARED @ 00001200
00002600 *00005400 *00005800
%
```

INTERACTIVEXREF Utility

The following REFERENCE command requests information on the identifier B at line 5601:

```
REF B AT 5601
```

```
CLOSEST MATCH: B @ 00005800
B :: REAL @ (2,2) :: DECLARED @ 00001200
    00002600 *00005400 *00005800
%
```

The following REFERENCE command requests information on the identifier B at line 2600. Because this command does not specify a <procedure specification> value, the global REAL B is used by default. Therefore, the INTEGER B declared at 3300 is not located.

```
REF B AT 2600
```

```
CLOSEST MATCH: B @ 00002600
B :: REAL @ (2,2) :: DECLARED @ 00001200
    00002600 *00005400 *00005800
%
```

The following REFERENCE command requests information on the INTEGER type B in procedure TWO:

```
REF B IN TWO
```

```
B OF TWO :: INTEGER @ (3,2) :: DECLARED @ 00003300
    *00004600
%
```

The following REFERENCE command requests information on the identifier C in procedure TWO:

```
REF C IN TWO
```

```
C OF THREE OF TWO :: INTEGER @ (4,2) :: DECLARED @ 00003700
    *00003800 00003900
%
```

The following command causes an error because C is not declared by procedure TWO. *OF <procedure specification>* does not check nested procedures for an occurrence of the identifier.

```
REF C OF TWO
```

```
ERROR: REFERENCES - IDENTIFIER NOT DECLARED BY SPECIFIED ENVIRONMENT.
    SCANNING C
%
```

Either one of the following two REFERENCE commands would produce information on the identifier C in procedure THREE of TWO:

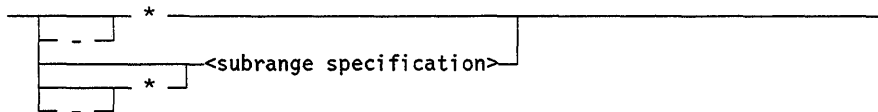
```
REF C OF THREE
REF C OF THREE OF TWO
```

```
C OF THREE OF TWO :: INTEGER @ (4,2) :: DECLARED @ 00003700
*00003800 00003900
%
```

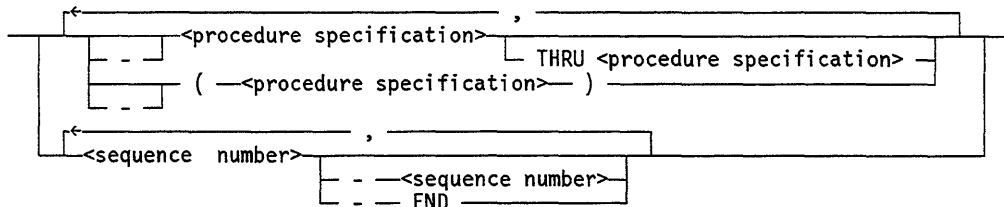
Range Specification

A range specification is used to restrict a request to a certain subset of the source file. Ranges can be specified in terms of either sequence numbers or environments (procedure names). Sequence numbers and environments cannot be intermixed in the same range specification.

<range specification>



<subrange specification>



Explanation

*

Specifies the current default reference range.

_*

Specifies the opposite of the current default reference range.

<subrange specification>

Specifies a procedure range or sequence range to be included.

INTERACTIVEXREF Utility

<procedure specification>

Includes the specified procedure and all procedures nested within it.

-<procedure specification>

Includes all procedures except the one specified, and all procedures nested within this procedure.

<procedure specification> THRU <procedure specification>

Includes all procedures in order of declaration from the first specified procedure through the second specified procedure.

-<procedure specification> THRU <procedure specification>

Includes all procedures except the procedures in order of declaration from the first specified procedure through the second.

(<procedure specification>)

Includes the specified procedure, but not the procedures nested within it.

- (<procedure specification>)

Includes all procedures except the one specified. The procedures nested within that procedure are included.

Examples

The following examples use the example INTERACTIVEXREF program contained at the end of this section.

The following range specification includes all sequence numbers from 1850 through 2300 and 5000 through the end of the file:

1850-2300, 5000-END

The following range specification includes global procedure ONE and all procedures declared within it, and global procedure TWO, but not the procedure declared within it – that is, procedure THREE:

ONE, (TWO)

The following range specification includes the entire source file except the global procedure ONE and all the procedures declared within it:

-ONE

The following range specification specifies the opposite of the current default reference range:

-*

The following range specification includes the current default reference range as well as global procedure ONE and all the procedures declared within that procedure:

*, ONE

The following range specification includes the entire source file except for the global procedure TWO. The procedures nested within TWO are included.

-(TWO)

INTERACTIVEXREF Commands

The following list names all the commands that you can enter while running INTERACTIVEXREF:

- DECLARATIONS Command
- EXPAND Command
- HELP Command
- LIST Command
- LOAD Command
- LOCATE Command
- MERGE and COINCIDENCE Commands
- QUALIFY Command
- RANGE Command
- REFERENCE Command
- SET and RESET Commands
- STOP Command
- SUMMARY Command
- SYMBOL Command
- TERMINAL Command

- WHAT Command
- WHATFILES Command

DECLARATIONS Command

The DECLARATIONS command causes a specified set of declarations to be isolated and listed along with optional information. The options available fall into three categories:

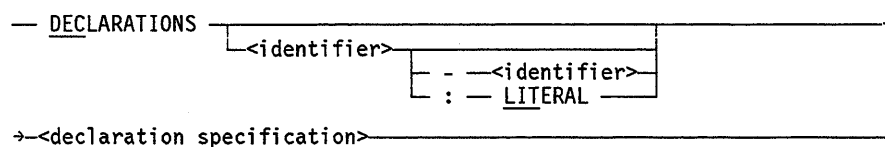
- Those that select the set of declarations
- Those that specify the output to be produced for each selected declaration
- Those that specify the order and destination of the output

The identifier specified in the most recently entered LOCATE, REFERENCE, EXPAND, or SUMMARY command is called the *work identifier*. INTERACTIVEXREF remembers the work identifier from command to command and can use it as a default identifier in certain commands. To avoid confusion, the DECLARATIONS command nullifies the work identifier.

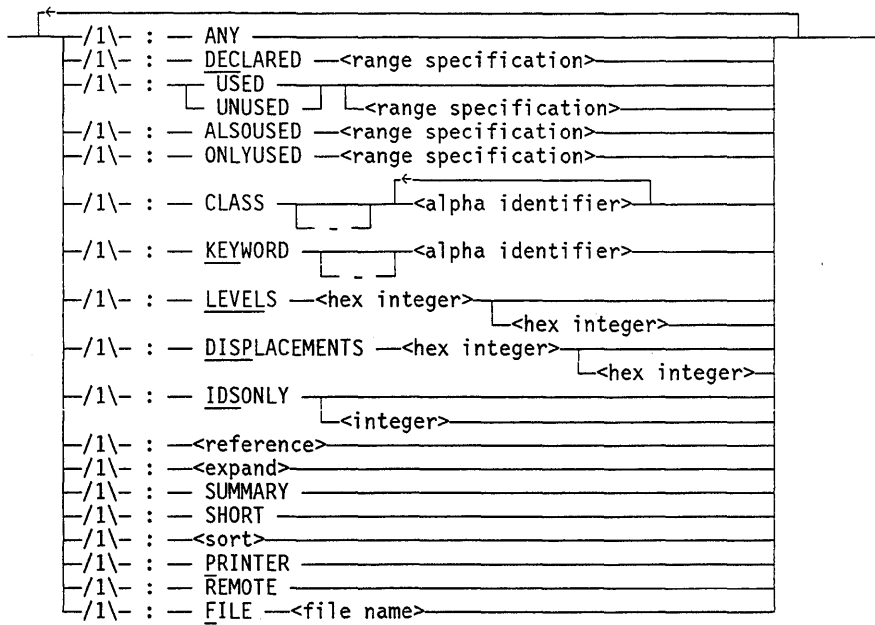
If no other options are specified as part of the DECLARATIONS command, the defaults are as follows:

- All declarations are included.
- The header line is given. The information contained in the header line includes the name, the compiler class, the environment where declared, the stack location, the sequence number where declared, and the aliases.
- The output is ordered alphabetically by identifier name and is sent to the terminal.

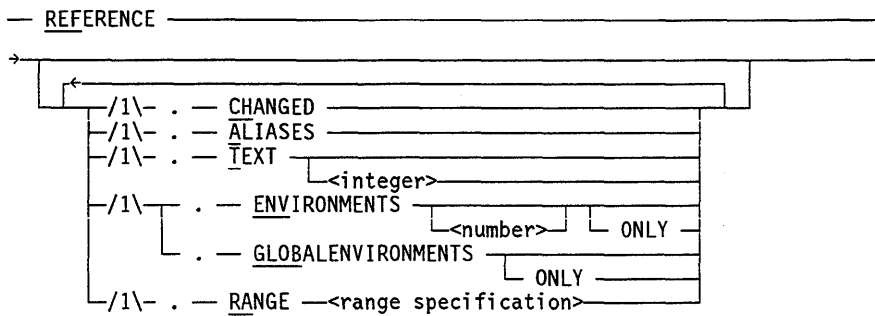
<declarations command>



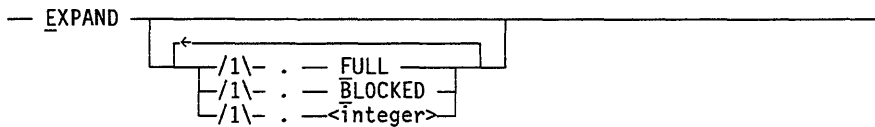
<declaration specification>



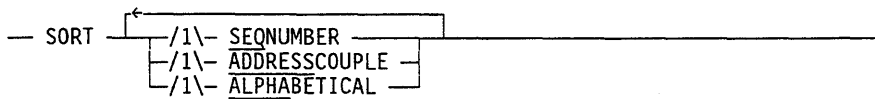
<reference>



<expand>



<sort>



Explanation

Options that control the selection of the set of declarations are defined as follows:

DECLARATIONS

Gives declaration information on all identifiers.

<identifier>

Gives declaration information for the specified identifier.

<identifier> - <identifier>

Gives declaration information for all the identifiers that are alphabetically between the specified pair of identifiers. The identifier pair must be ordered alphabetically.

<identifier> : LITERAL

Gives declaration information for all identifiers that contain the specified identifier as a substring.

ANY

Gives declaration information only for the first declaration meeting all other qualifications.

DECLARED <range specification>

Gives declaration information for all identifiers declared within the specified range.

USED

Gives declaration information for all identifiers that are referenced somewhere in the program. If a range is specified, this option restricts the set to identifiers referenced within that range.

UNUSED

Gives declaration information for identifiers that are declared but never referenced in the program. If a range is specified, the set is restricted to identifiers that are not referenced within the specified range.

ALSOUSED <range specifications >

Restricts declaration information to identifiers that also have references in this specified range.

ONLYUSED <range specification>

Gives declaration information for identifiers referenced within the specified range and not referenced elsewhere.

CLASS <alpha identifier>

Gives declaration information for all identifiers with the specified compiler class or group of compiler classes. The compiler class must appear exactly as it appears in a header line – for example, BOOLEAN ARRAY, INTEGER, FORMAL NAME REAL. Only one compiler class can be specified for each CLASS option; however, the compiler class can contain more than one alpha identifier – for example, REAL PROCEDURE). CLASS can be specified as often as desired; thus, a group of classes can be specified.

CLASS – <alpha identifier>

Gives declaration information for all identifiers in all the compiler classes except those specified in the alpha identifier list.

KEYWORD <alpha identifier>

Gives declaration information for the identifiers in the group of compiler classes that contain the specified alpha identifier. For example, KEY BOOLEAN causes classes such as BOOLEAN, BOOLEAN ARRAY, and BOOLEAN PROCEDURE to be included. KEYWORD and CLASS can be specified as often as desired to generate the desired group of classes.

KEYWORD – <alpha identifier>

Gives declaration information for the identifiers in the group of compiler classes that do not contain the specified alpha identifier. For example, KEY – BOOLEAN includes exactly the opposite of the classes included by KEY BOOLEAN.

LEVELS <hex integer>

Gives declaration information for the identifiers that have stack cells with the specified lexicographical levels.

DISPLACEMENTS <hex integer>

Gives declaration information for the identifiers that have stack cells with the specified displacements.

Options that specify the output to be produced for each selected declaration are described as follows:

IDONLY

Displays only identifier names and omits the other header information. The output is displayed in ascending alphanumeric order with a default field width of 20. The field width can be altered by specifying <integer> as the new field width.

REFERENCE

Displays references to the selected declaration. This option can itself be modified by any of the options listed under the REFERENCE command except PRINTER or REMOTE, with the same effects. For an explanation of these options, refer to "REFERENCE Command" later in this section.

EXPAND

Writes out the text of the selected declaration if the identifier is an item such as a DEFINE, ARRAY, or FILE statement that has text associated with its declaration.

If this option is modified by FULL, the text is completely expanded before it is displayed. If a full expansion of a DEFINE statement is requested, it is performed in the context of its first use.

If this option is modified by BLOCKED, the output is indented at the BEGIN statement, and statements are placed on separate lines. Only the first or the final expansion can be obtained.

The integer value specifies an approximate limit on the lines of text printed for each declaration. The default limit is 10.

SUMMARY

Lists a summary of the number and kinds of references to the selected declaration.

SHORT

Suppresses listing of the aliases of the selected declaration. Currently, only ESPOL and NEWP keep track of aliases. SHORT has no effect for other languages.

Options that specify the order and destination of output are described as follows:

SORT

Controls the order in which the selected declarations are printed:

- When SORT is followed by SEQNUMBER, the output is sorted by sequence number where each declaration was declared.
- When SORT is followed by ADDRESSCOUPLE, the output is sorted first on the address couple—the lexicographical level and the displacement of stack cell—of each declaration, and then on the sequence number where each was declared.

- When SORT is followed by ALPHABETICAL, the output is sorted first alphabetically, and then in order of occurrence. This output is the same output that would be produced if SORT were not specified.

When SORT is followed by more than one item, multiple sets of output are produced.

PRINTER

Sends output to the line printer by way of a file internally named LINE.

REMOTE

Sends output to the terminal. This option can be used when PRINTER has been specified so that output is sent to the terminal as well as to the printer.

FILE <file name>

Causes all referenced text lines from the symbol file to be output to a disk file with the specified file name. This file cannot already exist. The file name is created with the same FILETYPE value as the file loaded by the SYMBOL command. Refer to the "SYMBOL Command" in this section. If no symbol file is loaded, an error message is given at the terminal. The FILE option is valid only when used with the REFERENCE option.

Examples

The following examples use the example INTERACTIVEXREF program contained at the end of this section.

The following command requests information about all the declarations of the identifier B. The output is listed following the command.

```
DECLARATIONS B

B  :: REAL @ (2,2)  :: DECLARED @ 00001200
B OF TWO  :: INTEGER @ (3,2)  :: DECLARED @ 00003300
```


INTERACTIVEXREF Utility

The following command requests information about all of the declarations for identifiers that are alphabetically between B and MEAN inclusive:

```
DECLARATIONS B - MEAN

B :: REAL @ (2,2) :: DECLARED @ 00001200
B OF TWO :: INTEGER @ (3,2) :: DECLARED @ 00003300
B.0000 :: PROCEDURE @ (1,2)::DECLARED @ 00001100 ENDS @ 00006000
C :: REAL @ (2,3) :: DECLARED @ 00001200
C OF THREE OF TWO :: INTEGER @ (4,2) :: DECLARED @ 00003700
EQUATION :: DEFINE :: DECLARED @ 00002000
FOURR :: DEFINE :: DECLARED @ 00001900
I :: INTEGER @ (2,6) :: DECLARED @ 00001300
MEAN :: REAL @ (2,4) :: DECLARED @ 00001200
```

The following command lists information about declarations of identifier B under the compiler class of INTEGER:

```
DECLARATIONS B: KEYWORD INTEGER

B OF TWO :: INTEGER @ (3,2) :: DECLARED @ 00003300
```

The following command lists information for references to the identifier MEAN where its value can change:

```
DECLARATIONS MEAN: REF. CHANGED. TEXT

MEAN :: REAL @ (2,4) :: DECLARED @ 00001200
*00002400      MEAN:=2;
*00002600      MEAN:= MEAN * B;
*00003900      MEAN:= MEAN + C;
*00004200      MEAN:= MEAN / 3;
```

The following command lists information about all identifiers that are global to procedure TWO but are used within procedure TWO:

```
DECLARATIONS: DECLARED -TWO: USED TWO

C :: REAL @ (2,3) :: DECLARED @ 00001200
I :: INTEGER @ (2,6) :: DECLARED @ 00001300
MEAN :: REAL @ (2,4) :: DECLARED @ 00001200
ONEE :: DEFINE :: DECLARED @ 00001500
R :: REAL ARRAY @ (2,7) :: DECLARED @ 00001400
TWOO :: DEFINE :: DECLARED @ 00001600
```

The following command lists information about all identifiers that have stack cells within lexicographical level 2:

DECLARATIONS: LEVELS 2

```

B :: REAL @ (2,2) :: DECLARED @ 00001200
C :: REAL @ (2,3) :: DECLARED @ 00001200
I :: INTEGER @ (2,6) :: DECLARED @ 00001300
MEAN :: REAL @ (2,4) :: DECLARED @ 00001200
ONE :: PROCEDURE @ (2,8) :: DECLARED @ 00002200   ENDS @ 00002900
R :: REAL ARRAY @ (2,7) :: DECLARED @ 00001400
STRG :: REAL @ (2,5) :: DECLARED @ 00001200
TWO :: PROCEDURE @ (2,9) :: DECLARED @ 00003100   ENDS @ 00005000
%
```

EXPAND Command

The EXPAND command causes the text of a specified declaration to be written out (expanded) if the identifier is an item such as a DEFINE, ARRAY, or FILE statement that has text associated with its declaration.

The identifier specified in the EXPAND command becomes the new work identifier. If no identifier is specified, the current work identifier is used.

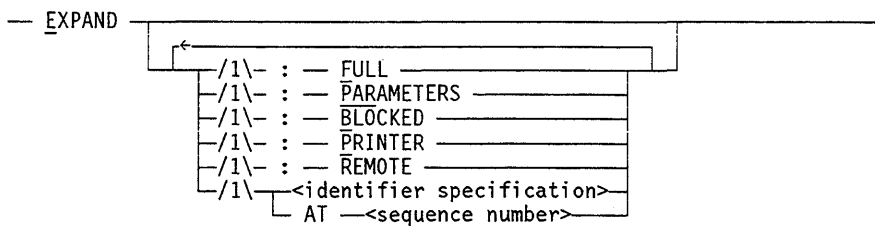
Unless the EXPAND command is modified by the available options, the action taken is as follows:

1. The first time the work identifier is expanded, the text of the declaration is given.
2. Subsequent requests to expand the work identifier cause this text to be scanned for occurrences of other DEFINE statements. If such nested DEFINE statements are found, they are replaced by their text. This process can be repeated, one step at a time, until the text is completely expanded (that is, no more nested DEFINE statements are found), a new work identifier is specified, or the work identifier is nullified.
3. Once the expansion is complete, a message to that effect is displayed. Subsequent requests to expand the work identifier cause the final version to be displayed.

The expansion of a DEFINE statement is context-sensitive. A DEFINE statement can be used within a procedure or block more local than that in which it was declared. In this case, the inner procedure or block might have redeclared some of the identifiers used in the text of the DEFINE statement. Unless the identifier was specified using a qualification such as *AT<sequence number>*, which gives some indication of the context to be used, a context must be chosen. If the DEFINE statement is referenced, it is expanded in the context of its first reference. If the DEFINE statement is never referenced, it is expanded in the context of its declaration. In either case, a warning message is displayed.

You must load the symbol file to use this command.

<expand command>



Explanation

EXPAND

Expands the work identifier. The work identifier is the identifier used in the most recently entered LOCATE, REFERENCE, EXPAND, or SUMMARY command. If no identifier is specified and the work identifier is empty, an error occurs.

FULL

Causes the text to be completely expanded before it is displayed.

PARAMETERS

If the identifier being expanded is a DEFINE with parameters, and if it is being expanded in the context of a reference, then actual parameters are extracted from the text of that reference and substituted for the formal parameters in the expansion. Even if the expansion was complete before the actual parameters were inserted, the expansion reverts to the incomplete state because the actual parameters can contain identifiers that are DEFINE statements.

Note: If both the FULL option and the PARAMETERS option are specified, the actual parameters are inserted before the full expansion is performed. PARAMETERS works only if the DEFINE statement was referenced directly and not by another DEFINE statement.

BLOCKED

Causes the first expansion to be blocked, that is, to indent at the BEGIN statement and place statements on separate lines. By default, the first expansion is printed out exactly as it appears in the symbol file (including comments). Subsequent levels of expansion are blocked.

PRINTER

Sends output to the line printer by way of a file internally named LINE.

REMOTE

Sends output to the terminal. This option can be used when PRINTER has been specified so that output is sent to the terminal as well as the printer.

<identifier specification>

Specifies the identifier to be expanded.

AT <sequence number>

Expands the identifier in the context of the reference nearest the specified sequence number.

Examples

The following examples use the example INTERACTIVEXREF program contained at the end of this section. These examples feature each step of the expansion process in order to show the effect of entering several EXPAND commands in a row.

The following command establishes EQUATION as the work identifier and gives the text of the declaration:

```
EXPAND EQUATION
      EQUATION :: DEFINE :: DECLARED @ 00002000
      EQUATION=R[ONEE] + R[TWOO] * R[THREE] - R[MORE] / R[ONEE];
      %
```

The first time the work identifier is expanded, the output is the expansion of all first-level defines, as follows:

```
EXPAND
      *** WARNING: EXPANDING IN CONTEXT OF FIRST USE ***
      EQUATION=R[1]+R[2]*R[3]-R[FOURR]/R[1];
      %
```

The next EXPAND command causes the expansion of all second-level defines:

```
EXPAND
      EQUATION=R[1]+R[2]*R[3]-R[4]/R[1];
      %
```

The next time the work identifier is expanded, the following message indicates that the work identifier is fully expanded:

```
EXPAND
```

```
EXPANSION COMPLETE
```

Every time following the completed expansion of the work identifier, the final expansion is given as follows:

```
EXPAND
```

```
EQUATION=R[1]+R[2]*R[3]-R[4]/R[1];
```

Considerations for Use

The room available for storing expansion text is limited. The first level of expansion is always completely printed out. Higher levels might be truncated if internal storage is insufficient.

For any given expansion, expansion of nested DEFINE statements is carried out in only one context. Also, text is displayed in complete syntax form, and declarations cannot be distinguished from other statements. As a consequence, higher-level expansions of DEFINE statements that contain declarations might be incorrect.

HELP Command

The HELP command displays a list of all commands, with a short description of each.

```
<help command>
```

```
— HELP —————|
   | : — PRINTER |
```

Explanation

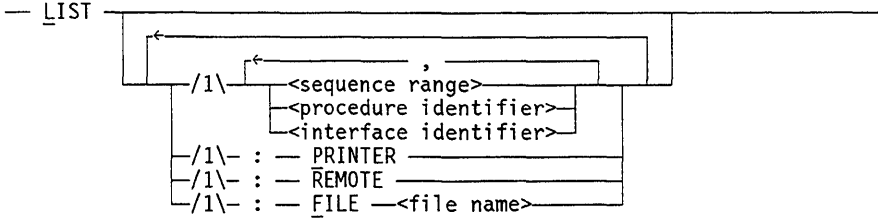
HELP: PRINTER

Causes the listing to go to the printer.

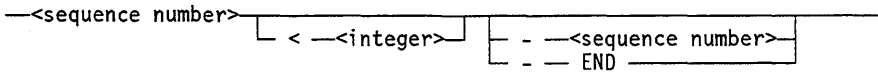
LIST Command

The LIST command lists text from the symbol file. The symbol file must be loaded before the LIST command can be used.

<list command>



<sequence range>



Explanation

LIST

Lists the entire file.

<sequence number>

Lists only the line specified by the sequence number. The file must be properly sequenced if any of the sequence number options are to be used.

<sequence number> < <integer>

Causes the command to begin listing the specified number of lines back from the sequence number and to list through the end of the file.

<sequence number> - <sequence number>

Lists the file beginning with the first sequence number through the second sequence number.

<sequence number> - END

Lists the file beginning with the specified sequence number through the end of the file.

<procedure identifier>

<interface identifier>

List only the specified procedure or interface.

PRINTER

Sends the output to the line printer by way of a file internally named LINE.

REMOTE

Sends the output to the terminal. This option can be used with the PRINTER option to send the output to the terminal as well as to the PRINTER.

FILE <file name>

Causes all lines in the sequence range, or all lines of the requested procedures or interfaces, to be written to a disk file with the specified file name. This file cannot already exist. The file name is created with the same FILETYPE value as the file loaded by the SYMBOL command. Refer to "SYMBOL Command" in this section.

Examples

The following command lists procedure ONE:

```
LIST ONE

00002200 PROCEDURE ONE;
00002300 BEGIN
00002400 MEAN:=2;
00002500 FOR I:= 0 STEP 1 UNTIL 5 DO
00002600     MEAN:= MEAN * B;
00002700     C:= C * MEAN;
00002800     STRG:= EQUATION;
00002900 END ONE;
%
```

LOAD Command

The LOAD command loads the INTERACTIVEXREF information files – that is, the XREFFILES. To prevent confusion, this command nullifies any previous SYMBOL command. Refer to "SYMBOL Command" in this section.

<load command>

— LOAD —<file name>—————|

Explanation

LOAD <file name>

Loads the XREFFILES associated with the specified file name. The file name should specify the object file generated by the compiler when the XREF information files were

generated. The titles of the XREF information files are constructed from this file name and then loaded.

Note: *When compiling a CANDE work file, use the format CANDE/CODE <number> for the name of the file.*

Example

The following command causes the files XREFFILES/CANDE/CODE390/REFS and XREFFILES/CANDE/CODE390/DECS to be loaded:

```
LOAD CANDE/CODE390
```

The following command causes the files XREFFILES/OBJECT/MAV/INTX/REFS and XREFFILES/OBJECT/MAV/INTX/DECS to be loaded:

```
LOAD OBJECT/MAV/INTX
```

LOCATE Command

The LOCATE command provides the following information about the specified identifier:

- The alphabetic name
- The declared type
- The program environment
- The stack location
- The sequence number of the declaration
- Any of the aliases

Other commands, such as REFERENCE, also print out this header line information.

<locate command>

```
— LOCATE —<identifier specification>—————|
```

Examples

The following examples use the example INTERACTIVEXREF program contained at the end of this section.

INTERACTIVEXREF Utility

The following command locates the identifier B declared or used closest to sequence number 5500:

```
LOCATE B AT 5500
```

```
CLOSEST MATCH: B @ 00005400
B :: REAL @ (2,2) :: DECLARED @ 00001200
%
```

The following command locates the identifier C declared by procedure THREE of global procedure TWO:

```
LOCATE C OF THREE OF TWO
```

```
C OF THREE OF TWO :: INTEGER @ (4,2) :: DECLARED @ 00003700
%
```

MERGE and COINCIDENCE Commands

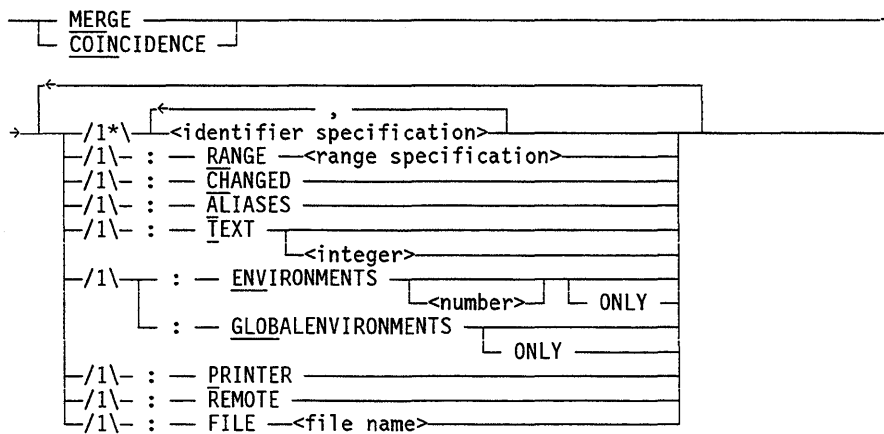
The MERGE command produces a merged list of the references to the specified identifiers.

The COINCIDENCE command produces a list of the places where all the references to the specified identifiers appear on the same line.

Note: Although the identifiers can be used in the same statement or expression, the statement or expression can be split across a line boundary, in which case the identifiers would not be flagged by the COINCIDENCE command.

<merge command>

<coincidence command>



Explanation

<identifier specification>

Specifies the identifiers to be used in the command.

RANGE <range specification>

Produces a **MERGE** or a **COINCIDENCE** list of references to the identifiers in the specified range. The default reference range, as specified by the **RANGE** command, is used if a range specification is not specified. The default value of the **RANGE** command is the entire program.

CHANGED

When specified with **MERGE**, causes only references where the value of the identifier might be changed by the statement to be listed.

When this option is specified with **COINCIDENCE**, it produces a list where all the specified identifiers appear on one line and where one or more might be changed by the statement within which it appears.

ALIASES

When specified with **MERGE**, produces a list of all references to the identifiers and all their aliases (if any exist).

When the **ALIASES** option is specified with **COINCIDENCE**, it produces a list of the locations where all specified identifiers and all their aliases appear.

For either command, sequence numbers where an alias is referenced are marked with a plus sign (+). Currently, only the **ESPOL** and **NEWP** languages keep track of aliases.

TEXT

Causes the text from the symbol file to be printed with each reference. If an integer is specified with this option, that many lines of text, centered at the line containing the reference, is displayed with each reference. You must load the symbol file to use this option. Refer to "SYMBOL Command" in this section.

ENVIRONMENTS

When specified with **MERGE**, produces a list of the names of the environments – procedures and blocks – where the references to any of the specified identifiers occur, appropriately interleaved with the references.

When **ENVIRONMENTS** is specified with **COINCIDENCE**, it produces a list of the names of the environments in which all specified identifiers appear on the same line, appropriately interleaved with the references.

When either command is modified by the <number> value, then only <number> levels of environments are listed. When either command is modified by ONLY then only the environments and not the references are listed. ENVIRONMENTS ONLY and TEXT are mutually exclusive options.

GLOBALENVIRONMENTS

When specified with MERGE, produces a list of the names of the global environments (global procedures and blocks) where the references to the specified identifiers occur, appropriately interleaved with references.

When GLOBALENVIRONMENTS is specified with COINCIDENCE, it produces a list of the names of the global environments in which all specified identifiers appear on the same line, appropriately interleaved with the references.

When either command is modified by the number value, then only that many levels of environments are listed. When either command is modified by ONLY, then only the global environments and not the references are displayed. GLOBALENVIRONMENTS ONLY and TEXT are mutually exclusive options.

PRINTER

Sends the output to the line printer by way of a file internally named LINE.

REMOTE

Sends the output to the terminal. This option can be used when PRINTER has been specified so that output is sent to the terminal as well as to the printer.

FILE <file name>

Causes all referenced text lines from the symbol file to be output to a disk file with the specified file name. This file cannot already exist. The file name is created with the same FILETYPE as the file loaded by the SYMBOL command. You must load the appropriate symbol file to use this option.

Examples

The following examples use the example INTERACTIVEXREF program contained at the end of this section.

The following command produces a merged list of references to the identifiers B, C, and MEAN. The references are grouped by the procedures within which they occur, and the name of the procedure precedes each group. The main procedure references are listed first. Text for the references is also printed.

MERGE B, C, MEAN: ENVIRONMENTS: TEXT

```

B :: REAL @ (2,3) :: DECLARED @ 00001200
C :: REAL @ (2,3) :: DECLARED @ 00001200
MEAN :: REAL @ (2,4) :: DECLARED @ 00001200
*00005400      B:= 3;
*00005500      C:= 25;
*00005800      B:=5;
ONE:
*00002400      MEAN:=2;
*00002600      MEAN:= MEAN * B;
*00002700      C:= C * MEAN;
TWO:
*00004200      MEAN:= MEAN / 3;
*00004800      C:= 43;
THREE OF TWO:
*00003900      MEAN:= MEAN + C;

```

The following command produces a list of statements where the value of B or C might change:

MERGE B, C: CHANGED: TEXT

```

B :: REAL @ (2,2) :: DECLARED @ 00001200
C :: REAL @ (2,3) :: DECLARED @ 00001200
*00002700      C:= C * MEAN;
*00004800      C:= 43;
*00005400      B:= 3;
*00005500      C:= 25;
*00005800      B:=5;

```

The following command produces a list of statements where both B and MEAN appear:

COINCIDENCE B, MEAN:TEXT

```

B :: REAL @ (2,2) :: DECLARED @ 00001200
MEAN :: REAL @ (2,4) :: DECLARED @ 00001200
*00002600      MEAN:= MEAN * B;
%
```

QUALIFY Command

The QUALIFY command establishes a new default identifier qualification. The default value is AT FIRST. When an identifier specification is encountered that does not have an explicit identifier qualification, the default identifier qualification is used to locate the identifier.

When an identifier specification has an explicit identifier qualification, the default qualification is overridden rather than supplemented.

<qualify command>
— QUALIFY _____
 └─<identifier qualification>┘

Explanation

QUALIFY

Displays the current default identifier qualification.

QUALIFY <identifier qualification>

Establishes the identifier qualification as the default to be used when an identifier specification is encountered that does not have an explicit identifier qualification.

Example

The following example establishes the procedure TWO as the default identifier qualification. OF TWO is used to qualify any identifier specification that does not already contain an explicit identifier qualification.

```
QUAL OF TWO
```

RANGE Command

The RANGE command establishes a default range specification. The default range specification is used when processing a REFERENCE, MERGE, COINCIDENCE, or DECLARATIONS command that does not include its own range specification.

If no RANGE command has been entered, the default reference range is the entire program.

<range command>
— RANGE _____
 └─<range specification>┘

Explanation

RANGE

Displays the current default reference range.

RANGE <range specification>

Establishes the range specification as the default range. If the range specification is a procedure and is enclosed in parentheses, it does not list lines of nested procedures. If the range specification is not enclosed in parentheses, it does not list lines in nested procedures.

Example

The following command establishes the procedure TWO and all procedures nested within it as the default range specifications:

```
RANGE TWO

NEW DEFAULT RANGE ESTABLISHED: TWO
```

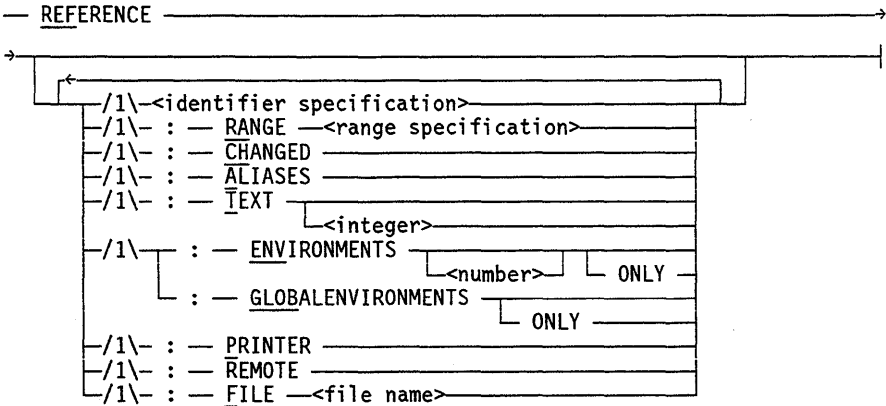
REFERENCE Command

The REFERENCE command lists the references to a particular identifier.

Unless modified by some of the options, the references are given in the following form:

- The 8-digit sequence number of each line where the identifier is referenced is given.
- The sequence number is preceded by an asterisk (*) if the value might be changed by the statement. The sequence number is followed by a number sign (#) if the reference occurred as part of an expanded DEFINE statement.

<reference command>



Explanation

REFERENCE

Lists the references to the work identifier. The work identifier is the identifier used in the most recently entered LOCATE, REFERENCE, EXPAND, or SUMMARY command. If the work identifier is empty, an error occurs.

<identifier specification>

Specifies a particular identifier. All references to that identifier are listed.

RANGE <range specification>

Restricts the range over which references are to be listed. The default reference range, as specified by the RANGE command, is used if the RANGE option is not specified. If no RANGE command has been specified, the entire program is used.

CHANGED

Lists only references where the value of the identifier might be changed by the statement.

ALIASES

Produces a merged list of references to the identifier and all its aliases (if any exist). Sequence numbers where an alias is referenced are marked with a plus sign (+). Currently, only the ESPOL and NEWP languages keep track of aliases.

TEXT

Lists the text from the symbol file with each reference. You must load the symbol file to use the TEXT option.

TEXT <integer>

Causes the specified number of lines, centered at the line containing the reference, to be printed with each reference.

ENVIRONMENTS

Lists the names of the environments – procedures and blocks – where the references occur, appropriately interleaved with the references. If modified by a specified <number> value, then only <number> levels of environments are listed. If modified by ONLY, then only the environments and not the references are listed. ENVIRONMENTS ONLY and TEXT are mutually exclusive options.

GLOBALENVIRONMENTS

Similar to ENVIRONMENTS, except that references are broken down only by global environment – global procedure. When this command is modified by ONLY, then only the global environments and not the references are listed. GLOBALENVIRONMENTS ONLY and TEXT are mutually exclusive options.

PRINTER

Sends the output to the line printer by way of a file internally named LINE.

REMOTE

Sends the output to the terminal. This option can be used when PRINTER has been specified so that output goes to the terminal as well as to the printer.

FILE <file name>

Causes all referenced text lines from the symbol file to be output to a disk file with the specified file name. This file cannot already exist. The file name is created with the same FILETYPE value as the file loaded by the SYMBOL command. You must load the symbol file to use this option.

Examples

The following examples refer to the program under “Example INTERACTIVEXREF Program” in this section:

The following command locates the identifier MEAN used in procedure TWO and all procedures nested within it, and lists the associated line numbers:

```
REFERENCE MEAN: RANGE TWO

      MEAN  :: REAL @ (2,4)  :: DECLARED @ 00001200
          *00003900  *00004200
          %
```

The following command locates the identifier MEAN used in procedure TWO and lists all associated line numbers for procedure TWO, but not the procedures nested within it:

```
REFERENCE MEAN: RANGE (TWO)

      MEAN  :: REAL @ (2,4)  :: DECLARED @ 00001200
          *00004200
          %
```


INTERACTIVEXREF Utility

The following command locates the identifier C used in procedure THREE of procedure TWO. A list of references where the value of C might be changed and the line of text corresponding to each reference is displayed.

REFERENCE C OF THREE OF TWO:CHANGED:TEXT

```
C OF THREE OF TWO :: INTEGER @ (4,2) :: DECLARED @ 00003700
*00003800      C:= 5;
%
```

The following command locates the identifier MEAN and lists all references with the associated text. The references are grouped by the procedures within which they occur. The name of the procedure precedes each group.

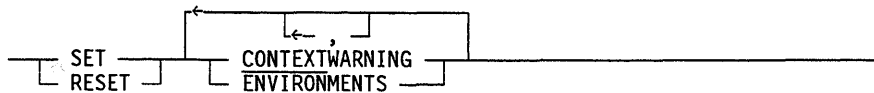
REFERENCE MEAN: TEXT: ENVIRONMENTS

```
MEAN :: REAL @ (2,4) :: DECLARED 00001200
ONE:
*00002400      MEAN:=2;
*00002600      MEAN:= MEAN * B;
 00002700      C:= C * MEAN;
TWO:
*00004200      MEAN:= MEAN / 3;
THREE OF TWO:
*00003900      MEAN:= MEAN + C;
%
```

SET and RESET Commands

The SET and RESET commands enable or disable run-time options, respectively.

<set command>
<reset command>



Explanation

CONTEXTWARNING

When used with the SET command, causes a warning to be given when the EXPAND command is forced to expand a DEFINE statement in the context of its first use. When this option is used with the RESET command, the warning is suppressed. The default is the SET command.

ENVIRONMENTS

When used with the SET command, the environment information is included in the header line of a nonglobal identifier. When this option is used with the RESET command, environment information is not included. The default is the SET command.

STOP Command

The STOP command terminates the INTERACTIVEXREF session.

<stop command>

— STOP _____|

SUMMARY Command

The SUMMARY command lists a summary of the number and kinds of references to a given identifier.

<summary command>

— SUMMARY _____|
 └<identifier specification>┘

Explanation

SUMMARY

Lists the SUMMARY information for the work identifier. The work identifier is the identifier used in the most recently entered LOCATE, REFERENCE, EXPAND, or SUMMARY command. If the work identifier is empty, an error occurs.

SUMMARY <identifier specification>

Lists the SUMMARY information for the specified identifier.

Example

The following example refers to the program under “Example INTERACTIVEXREF Program” in this section.

The following command prints a summary of references to the global identifiers MEAN, B, and C:

```
SUMMARY MEAN; SUMMARY B; SUMMARY C;

MEAN  :: REAL @ (2,4)  :: DECLARED @ 00001200
TOTAL REFERENCES =           5
NUMBER POSSIBLY CHANGED =    4
REFERENCES OCCUR IN RANGE 00002400-00004200
B     :: REAL @ (2,2)  :: DECLARED @ 00001200
TOTAL REFERENCES =           3
NUMBER POSSIBLY CHANGED =    2
REFERENCES OCCUR IN RANGE 00002600-00005800
C     :: REAL @ (2,3)  :: DECLARED @ 00001200
TOTAL REFERENCES =           3
NUMBER POSSIBLY CHANGED =    3
REFERENCES OCCUR IN RANGE 00002700-00005500
```

SYMBOL Command

The SYMBOL command loads the symbol file from which text for DEFINE statement expansion, text corresponding to a given reference, and text for the LIST command are taken. If none of these items is desired, a symbol file need not be loaded.

<symbol command>

— SYMBOL —<file name>—————|

Explanation

SYMBOL <file name>

Loads the specified file. The file name should refer to the source file that was compiled when the INTERACTIVEXREF information files were generated. It is desirable, but not necessary, that the symbol file and the XREF information files correspond exactly. If discrepancies are found when a command that requires information from both sources is processed, a warning or error is issued.

Example

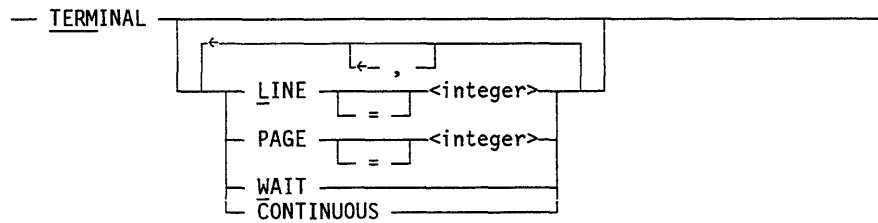
The following command loads the symbol file MAV/INTX:

```
SYMBOL MAV/INTX
```

TERMINAL Command

The **TERMINAL** command specifies attributes that control the format of output to the terminal. If no attributes are specified, the current terminal specifications are displayed. The initial values of **PAGE** and **LINE** options are taken from the file attributes of the remote file when it is opened.

<terminal command>



Explanation

LINE <integer>
LINE = <integer>

Specify the maximum width of an output line, expressed in characters. The integer value must be between 72 and 132.

PAGE <integer>
PAGE = <integer>

Specify the number of lines on each page. This option is relevant only if **WAIT** is specified.

WAIT

Groups the output into pages in the form described by the LINE and PAGE options. After each page except the last page, the program stops and waits for input from the terminal. If the input is blank, the next page is printed; otherwise, the command is aborted, as if a break on output had occurred.

CONTINUOUS

Disables the WAIT option.

Examples

The following command sets the maximum width of an output line on the terminal to 80 characters:

```
TERM LINE 80
```

The following command causes each page of output to contain 23 lines. The program waits for a blank response at the end of each page. The count starts at the beginning of each command.

```
TERM PAGE 23 WAIT
```

WHAT Command

The WHAT command lists a description of the current work identifier, which is used as the default identifier in subsequent REFERENCE, EXPAND, or SUMMARY commands.

<what command>

```
— WHAT —————|
```

WHATFILES Command

The WHATFILES command produces a list that indicates which INTERACTIVEXREF information files and which symbol file are loaded.

<whatfiles command>

```
— WHATFILES —————|
```

Using the INTERACTIVEXREF Utility

The information that follows explains several different methodologies for using the INTERACTIVEXREF utility.

Use with Improperly Sequenced Source

The environment information in each header line clearly identifies an identifier even if the sequence number is meaningless. The ENVIRONMENTS ONLY and GLOBALENVIRONMENTS ONLY options of the REFERENCE command list the procedures within which an identifier is used. If individual procedures are properly sequenced, then the ENVIRONMENTS and GLOBALENVIRONMENTS options give the procedure names and sequence numbers where the references are located. Because references are sorted first by procedure and then by sequence number, all references within a given procedure are grouped together.

If the source is not properly sequenced, the *AT <sequence number>* option does not work. The form *AT <sequence number> IN <procedure specification>* has been provided especially for cases where the specified procedure is sequenced properly and contains nested blocks, procedures that redeclare some of its identifiers, or both.

Environment ranges work regardless of the sequencing of the source. Sequence number ranges produce undefined results if the source is improperly sequenced.

The EXPAND and LIST commands, the TEXT option, and the EXPAND option in the DECLARATIONS command perform binary searches on the SYMBOL file to obtain needed text. Therefore, these constructs work only if the needed text is obtained from the currently loaded symbol file and if that file is properly sequenced.

Use with COBOL74

Any valid COBOL74 identifier is accepted as a valid identifier when COBOL74 XREF files are loaded. The identifier qualification terms IN and OF also can be used with record specifications. This use of IN and OF specifies a particular record identifier that contains or owns the given identifier.

The EXPAND command is not available when INTERACTIVEXREF is run against COBOL74 programs.

Use with FORTRAN and FORTRAN77

FORTRAN does not keep track of environments for INTERACTIVEXREF; therefore, no environment information is included in the header line. The ENVIRONMENTS and GLOBALENVIRONMENTS options are not available, nor are any of the environment ranges, identifier environment ranges, or identifier qualifications – with the exception of AT FIRST and AT <sequence number>.

The EXPAND command is not available when INTERACTIVEXREF is run against FORTRAN and FORTRAN77 programs.

A number (FORTRAN or FORTRAN77 label) or an identifier containing dollar signs (\$) is accepted as a valid identifier when FORTRAN and FORTRAN77 XREFFILES are loaded.

Use with PASCAL

A number (Pascal label) is accepted as a valid identifier when Pascal XREF files are loaded.

Example INTERACTIVEXREF Program

The following program is used for the examples in this section. This program has several procedures, and redeclares some of its identifiers.

```
00001000 $SET XREFFILES
00001100 BEGIN
00001200 REAL B, C, MEAN, STRG;
00001300 INTEGER I;
00001400 ARRAY R[1:3];
00001500 DEFINE ONEE = 1 #,
00001600     TWO = 2 #,
00001700     THREE = 3 #,
00001800     MORE = FOURR #,
00001900     FOURR = 4 #,
00002000 EQUATION= R[ONEE] + R[TWOO] * R[THREE] - R[MORE]/R[ONEE] ;
00002200 PROCEDURE ONE;
00002300     BEGIN
00002400     MEAN:=2;
00002500     FOR I:= 0 STEP 1 UNTIL 5 DO
00002600         MEAN:= MEAN * B;
00002700         C:= C * MEAN;
00002800         STRG:= EQUATION;
00002900     END ONE;
00003100 PROCEDURE TWO;
00003200     BEGIN
00003300     INTEGER B;
00003500 PROCEDURE THREE;
00003600     BEGIN
00003700     INTEGER C;
```

```
00003800      C:= 5;
00003900      MEAN:= MEAN + C;
00004000      END THREE;
00004200      MEAN:= MEAN / 3;
00004300      R[1]:= 1;
00004400      R[2]:= 2;
00004500      R[3]:= R[ONEE] + R[TWOO];
00004600      B:= 28;
00004700      I:= 5;
00004800      C:= 43;
00004900      THREE;
00005000      END TWO;
00005400      B:= 3;
00005500      C:= 25;
00005600      ONE;
00005700      TWO;
00005800      B:=5;
00005900      ONE;
00006000      END.
```


Section 7

KEYEDIO Support

The COBOL74 and RPG indexed sequential-access method permits a file sequenced by keys to be processed in both random and sequential modes. The COBOL74 and RPG indexed sequential-access method is provided by the SYSTEM/KEYEDIO library (KEYEDIO) and has the same capabilities as the ISAM intrinsic in the SYSTEM/PLISUPPORT library. However, files created by PLISUPPORT cannot be referenced by KEYEDIO, nor can files created by KEYEDIO be referenced by PLISUPPORT. You can obtain access to KEYEDIO procedures as follows:

- For COBOL74 and RPG programs, through code that is generated by compilers.
- For ALGOL, DCALGOL, and Pascal programs, through procedures in the GENERALSUPPORT system library. Refer to “KEYEDIO Procedures” later in this section.

This section explains ISAM files that have FILEORGANIZATION value equal to INDEXED or INDEXEDNOTRESTRICTED. The KEYEDIO system is compatible with KEYEDIOII – ISAM files that have a FILEORGANIZATION value equal to KEYEDIOII or KEYEDIOIISET. KEYEDIOII includes several new features that increase system performance over that of KEYEDIO and is documented in the *KEYEDIOII Reference Manual*. For information about ISAM files that have a FILEORGANIZATION value equal to PLIISAM, refer to Section 10, “PL/I Indexed Sequential-Access Method (PLIISAM).”

Physical Structure of KEYEDIO Files

KEYEDIO files consist of three logical areas contained within one physical file: coarse tables, fine tables, and data. The size of these areas increases during the life of a file as records are added and deleted. The areas are not distinct from one another and are intermixed throughout the file. Each block of data in a KEYEDIO file contains information for a single type of area. Control information describing the block and how to access it is appended to the beginning of each block in the file.

Coarse Tables

Coarse tables contain key values that describe fine tables or other coarse tables. One entry exists in the coarse tables for each fine table. Several coarse tables are created if more fine tables are present than can be indexed by a single coarse table. These coarse tables are then ordered by another level of coarse tables. This hierarchy continues until only one table remains at the top level; this table is called the root table. A coarse table entry consists of a key value equal to the largest key in the next lower table and a pointer to that table.

Fine Tables

Fine table blocks contain one key entry for each record. A key entry consists of a key value and a pointer to the data record associated with that key value.

Data Blocks

Data blocks contain your records. The records are stored in these blocks exactly as they were written, but the record size is increased if you specify that relative keys are to be used. This addition is internal to the file only and need not be used when calculating record size (MAXRECSIZE). A data block is accessed by going through fine tables.

Locating Data

To find a specific record, your key value is first compared against the key values in the coarse tables. The first key value in the coarse table that is greater than or equal to your key value is used to locate the coarse table at the next level. This process continues until a fine table is encountered. The fine table is then searched for your key value, and the data referenced by that key entry is returned.

Because fine tables are sequentially ordered, they are linked together so that only the fine tables and the data areas need to be read when the file is being accessed sequentially.

A set of coarse and fine tables is created for each key you define. Also, coarse and fine tables are created for the relative key.

There is a limit of 48 keys per KEYEDIO data file. If the number of keys exceeds 48, one of the following error messages is given:

FILE CONTAINS TOO MANY KEYS (GREATER THAN 48) FOR KEYEDIO TO HANDLE

USER OPEN REQUEST DECLARES TOO MANY KEYS (GREATER THAN 48) FOR
KEYEDIO TO HANDLE

File and KEYEDIO Library Management

All KEYEDIO file management routines are contained in the SYSTEM/KEYEDIO library. Information about all the users of the file is also contained in the library stack. The library mechanism permits multiple users to access the same file and also provides links for all users of the same file to the same library stack. Your position in the file is maintained by a *current record pointer* that points to the fine table entry corresponding to your current record in the file.

When a file is opened, if the FILEORGANIZATION file attribute is equal to INDEXED or INDEXEDNOTRESTRICTED, the user program is linked to the SYSTEM/KEYEDIO library routines instead of to the normal FIBSTACK procedures.

The KEYEDIO open routine performs certain checks to ensure file integrity. Refer to the *A Series File Attributes Programming Reference Manual* for more information.

If the file is declared to be of type INDEXED, all keys declared when the file was created must be declared by your program each time the file is opened, and these keys must match exactly. If the file is of type INDEXEDNOTRESTRICTED, keys not known – not declared – by your program are still updated.

When the file is updated, recovery information is stored so that file integrity can be maintained in the event of abnormal termination. Because of the overhead involved, this information is not saved when the file is being created.

KEYEDIO files can be used by more than one user program at a time. Any number of user programs can be reading the file, but when one of the programs attempts to update – add, delete from, or rewrite – the file, other users of the file are locked out for the duration of the update transaction. This lockout feature permits more than one program to have the file open in update mode, while also preserving the integrity of the data. No mechanism exists for locking out other users for more than a single transaction; record level lockout is not provided.

The KEYEDIO library keeps track of the number of programs currently reading and writing to the file. For the file to be updated, both the reader and writer counts must equal zero (0). For the file to be read, the write count must equal zero (0).

Removing and Installing a KEYEDIO Library

The KEYEDIO library is written and compiled in NEWP. The base library is frozen permanently. Because the base library is frozen, you must perform the following steps if the SYSTEM/KEYEDIO library needs to be removed from the active library list:

1. Find the mix number of the KEYEDIO library by entering the LIBS (Library Task Entries) system command. Refer to the *A Series System Commands Operations Reference Manual* for more information.
2. Enter `<mix number> THAW` at the ODT to remove the KEYEDIO library from the list.

Note: *KEYEDIO terminates only when no tasks are using it.*

The KEYEDIO library job summary is then printed.

After the old KEYEDIO library has terminated, you can install a new KEYEDIO library by using the SL (Support Library) system command.

KEYEDIO Program Interface

Only COBOL74 and RPG can create files that define keys within a record. All languages can use the INDEXEDNOTRESTRICTED value of the FILEORGANIZATION attribute and normal sequential or random I/O statements to access or create files sequentially or with relative keys. Files created and later opened

as INDEXEDNOTRESTRICTED and used with keys have all keys updated during an update, even if you have not specified all the keys.

For RPG and COBOL, the procedures provided by the KEYEDIO library can be used only through normal compiler I/O constructs. For languages such as ALGOL and Pascal, the KEYEDIO operations are available through procedures provided by the GENERALSUPPORT library.

Indexed KEYEDIO File Attributes

There are two kinds of attributes discussed in this section: file attributes that must be set in a special way when indexed files are created and accessed (FILEORGANIZATION, EXCLUSIVE, BUFFERS, and BLOCKSIZE), and attributes that are internal to SYSTEM/KEYEDIO and can only be accessed by the compilers (ISAMKEYS and ACCESSMODE).

To ensure maximum processing efficiency and minimum use of save memory, the BLOCKSIZE and BUFFERS attributes should be set carefully. Refer to the *A Series File Attributes Programming Reference Manual* for descriptions of these and other file attributes.

Increasing the number of buffers used by KEYEDIO reduces the time needed to process an indexed file at the expense of increased usage of save memory. In contrast, reducing the number of buffers increases processing time and decreases save memory usage.

Increasing the block size to allow for one- or two-level access to data in the indexed file decreases processing time but also increases usage of save memory. Decreasing block size to allow for three- or four-level data access increases processing time and decreases save memory usage.

Consider the needs of your particular installation when choosing BLOCKSIZE and BUFFERS values. Suggestions for how to choose BUFFERS and BLOCKSIZE values are given in the following paragraphs.

Accessing a KEYEDIO file using a record size other than the record size with which the file was created could cause file corruption or faults in the KEYEDIO library. To prevent one of these results, KEYEDIO verifies that the values specified by a program for the MAXRECSIZE and UNITS file attributes are identical to those specified at the time the file was created. If a mismatch is detected, the program is discontinued with the following message:

```
RECORD LENGTH MISMATCH (FILE MAXRECSIZE=nnn; PROGRAM MAXRECSIZE=mmm)
```

The value of *nnn* is the MAXRECSIZE value assigned at file creation, and the value of *mmm* is the MAXRECSIZE value specified by the program.

To open a file in which the key structure is not known, set the file attribute DEPENDENTSPECS to TRUE and use the procedure ISMGETKEYSTRUCTURE to get key information. See "ISMGETKEYSTRUCTURE Procedure" later in this section for details.

Setting the FILEORGANIZATION Attribute

The FILEORGANIZATION attribute can be set in one of two ways for an indexed file: indexed files that have relative keys have FILEORGANIZATION attribute equal to INDEXEDNOTRESTRICTED. (RPG files have relative keys by default.) Indexed files that do not have relative keys have FILEORGANIZATION attribute equal to INDEXED. The FILEORGANIZATION attribute must be set each time the indexed file is opened. Refer to the *A Series File Attributes Programming Reference Manual* for more information about the FILEORGANIZATION attribute.

Setting the EXCLUSIVE Attribute

The EXCLUSIVE attribute can be set to TRUE only for files that are accessed by only one user at a time, or for files that are accessed by only users currently operating in the same subsystem. For more information about the EXCLUSIVE attribute, refer to the *A Series File Attributes Programming Reference Manual*.

Setting the Value of the BUFFERS Attribute

You can control the number of buffers used by the KEYEDIO library in processing an indexed file.

A program can indicate the number of buffers KEYEDIO is to use by setting the value of the BUFFERS attribute of the indexed file. The value of the BUFFERS attribute is used by the library to determine how many buffers to allocate for processing that indexed file.

Impact of Number of Buffers on Processor Time

Increasing the number of buffers used by KEYEDIO reduces the time needed to process an indexed file at the expense of increased usage of save memory. Reducing the number of buffers decreases save memory usage but increases processing time. In general, programs doing random accesses to the indexed file are more sensitive to the number of buffers than programs doing serial accesses.

The effect of changing the number of buffers on the time needed to process a file tends to be proportional to the reciprocal of the number of buffers; that is, reducing the number of buffers lengthens the processing time more than increasing the number of buffers by the same amount decreases the time. For this reason, decisions to decrease the number of buffers below the KEYEDIO default of 10 buffers should be the result of careful consideration and measurement.

Impact of Number of Buffers on Save Memory

Increasing the number of buffers increases the save memory usage of the KEYEDIO library; save memory is equal to the number of buffers multiplied by the actual block size and is explained later in this section. You should take into account the overall performance of the system, as well as the processing time of the specific application, when deciding to increase the number of buffers.

The value specified for the `BUFFERS` attribute at file creation time is especially important because this value becomes the permanent default number of buffers to be allocated whenever the file is used later. The time needed to create the file is usually not as strongly affected by the number of buffers as later uses of the file are. For this reason, specifying the proper number for the permanent default is generally of greatest importance when specifying the `BUFFERS` value to use for file creation.

Rules for Determining the Number of Buffers Used

KEYEDIO determines how many buffers to use for processing an indexed file according to the following rules:

1. First, the value of the `BUFFERS` attribute specified when the file is created is stored permanently in the file itself. This value is used by the KEYEDIO library both when creating the file and as the default number of buffers to allocate each time the file is subsequently opened.

If the value of the `BUFFERS` attribute is not specified at file creation time or the value specified for the `BUFFERS` attribute is 2 or less, KEYEDIO uses the default value 12 for the number of buffers both for file creation and as the default number of buffers when the file is subsequently opened for use by a single user.

For files with a `BUFFERS` value greater than 2, KEYEDIO allocates 10 more buffers than the number of buffers specified in the creating program. For example, if program A sets the attribute `BUFFERS` of file F to 5, and then opens the file, `HEADERBUFFERS` is 13, and 15 buffers are allocated.

2. Second, the value of the `BUFFERS` attribute specified when an existing indexed file is opened is used to indicate the number of buffers the KEYEDIO library should use in processing that file. KEYEDIO determines the number of buffers to use for an existing file in the following way:

When the file is not being used by any other programs at the time it is opened, and if the number of buffers specified is 3 or greater, the number of buffers allocated is 2 more than specified by the opening program, or it is the number of buffers allocated when the file was created, whichever is greater.

For example, if program A has closed file F and program B reopens the file with `BUFFERS` equal to 15, then 17 buffers are allocated.

When the file is being used by other programs at the time it is opened, either of the following can be true:

- If the original number of buffers allocated at file creation is more than 2 greater than the number of buffers allocated by the additional opening program, then KEYEDIO allocates 2 additional buffers.
- If the number of buffers allocated at file creation is more than 2 less than the number of buffers allocated by the additional opening program, then KEYEDIO allocates 4 more buffers than the difference between program-specified buffers and the original creation buffers.
- In certain languages the number of buffers is a default value determined by the compiler (as described in the following).

When there are multiple users of an indexed file, the buffers are allocated in a common pool by the KEYEDIO library and are shared by all users.

For example, if program B has file F open with BUFFERS equal to 15, and program C sets F.BUFFERS to 20 and then opens the file, 9 more buffers (2 + 20 – 13) are allocated for a total of 26.

If program A sets BUFFERS equal to 7, then HEADERBUFFERS becomes 15 and 17 buffers are allocated while A is running. If program B reopens the file with BUFFERS equal to 15, 17 buffers are allocated. If program C sets F.BUFFERS equal to 20 while B is still running, then 7 more buffers (2 + 20 – 15) are allocated, for a total of 24.

Some compilers automatically set the value of BUFFERS even when the program has not specified a value for BUFFERS. In particular, the COBOL74 compiler sets BUFFERS to 2 if a value is not specified; the RPG compiler sets BUFFERS to 1 if a value is not specified for an indexed file.

The compiler-set defaults do not interfere with the default assignment of 10 buffers at file creation time for an indexed file—because the compiler default is 2 or less—or with the assignment of 10 buffers by default when a single access is made to an already existing indexed file. The compiler default has an impact when many programs access an indexed file. Each new program accessing the indexed file increments the number of buffers in use by the compiler default setting.

The maximum value that can be set for the BUFFERS file attribute is 63. The maximum total number of buffers used by the KEYEDIO library is 255. Once this limit is reached, additional buffers are not allocated, regardless of the BUFFERS specifications of later users.

If the BUFFERS attribute of an indexed file is interrogated, the value returned is the value currently established for that file by the program, not the total number of BUFFERS that are actually being used by KEYEDIO at that time.

Choosing a Value for the BLOCKSIZE Attribute

The actual block size used by KEYEDIO is different from the block size provided by you because space must be added to round the record size to an exact multiple of 6 characters, to provide room for the relative keys of an INDEXNOTRESTRICTED file, and to provide for the 10 words of header information in each block. Actual block size is used to calculate how much save memory is actually occupied by the KEYEDIO file. If the actual block size that has been calculated is not satisfactory to the programmer, it might be necessary to adjust the specified block size; that is, the BLOCKSIZE attribute value.

Make sure the block size is large enough to store more than one key for each block. If the block size is not large enough to store more than one key, an error occurs.

The specified block size is saved in the KEYEDIO file and is returned as the value of the BLOCKSIZE file attribute when the indexed file is open. If this attribute is interrogated when the file is closed, it always returns the value of 30, which is the value that the KEYEDIO library uses when creating the file. (This is a side effect of the fact that the KEYEDIO library manipulates the file using DIRECT I/O.)

Effect of Block Size on Processor Time

The proper specification of block size is extremely important to the performance of applications that use indexed files because the actual block size (calculated according to the algorithm described in “Calculating Actual Block Size”) is used not only for storing the data but also as the size of the key index tables used to access the data. The size of these tables and the number of records in the file determine how many tables must be searched in order to find a particular record. Each additional table that must be searched increases the processor and I/O time that is required to access a record.

The most efficient access is obtained when only a single table must be searched in order to find the key. A single-table search requires that the block size be large enough to hold the keys for all the records in the file; thus, this block size is usually not a practical choice except for files with a small number of records.

The next most efficient access is obtained when only two tables—a coarse table and a fine table—must be searched to find the key. A two-table search requires a block size large enough to hold a number of keys equal to the square root of the number of records in the file. A block size of this value is generally the most suitable choice for all but very small or very large indexed files. A block size smaller than this square root value requires multiple table accesses and noticeably increases the time required for random accesses to the file.

Effect of Block Size on Save Memory

The buffers used by KEYEDIO occupy save memory. The amount of save memory to be used for a given indexed file can be approximated by multiplying the actual block size (calculated according to the algorithm given in “Calculating Actual Block Size”) by the number of buffers to be used for the file.

If the save memory requirements for block sizes that provide one- or two-level access to data are too great, a new block size should be calculated that provides three- or four-level access. This block size can be calculated using the algorithm given under “Calculating User-Specified Block Size (2 Level)”; but at step 2 compute the cube root or fourth root of the number of records instead of the square root.

Calculating Actual Block Size

The actual block size used by KEYEDIO is different from the BLOCKSIZE attribute specified by you because space must be added to round the record size up to an exact multiple of 6 characters, to provide room for the relative keys of an INDEXEDNOTRESTRICTED file, and to provide for the 10 words of header information in each block.

KEYEDIO uses the following algorithm to compute the actual block size for a file:

1. Divide the specified BLOCKSIZE by the specified record size (MAXRECSIZE), truncating any remainder. This gives the records per block value.
2. Round the specified record size up to the next multiple of 6 characters, if it is not already an exact multiple of 6 characters. Convert this record size to the number of words required to hold the record by dividing by six.
3. If this is an INDEXEDNOTRESTRICTED file, add 1 (word) to the record size to allow space for the relative key.
4. Compute a trial block size by multiplying the record size in words (calculated in steps 2 and 3) by the specified records per block (calculated in step 1). Then add 10 words to provide space for the header information in each block.
5. Calculate the actual block size by rounding the trial block size (from step 4) up to the next multiple of 30 words, if it is not already an exact multiple of 30 words.

Once the actual block size has been calculated, as many records as can fit are placed in each block. That is, if the rounding process of step 5 adds enough space to the block for additional records, that space will be used, and the actual records per block will be greater than the specified records per block calculated in step 1.

Calculating User-Specified Block Size (2 Level)

To calculate the proper block size for an indexed file, assuming the two-level table search is desired, make the following calculation:

1. Calculate the number of records the file will contain over its lifetime.
2. Compute the square root of the number of records. Then multiply this value by an *adjustment* factor to allow for the fact that not all the tables will be completely filled. The result of this computation is the desired number of keys per block.

The value of the adjustment factor is determined by the way the file is created and updated. If the file is created sequentially with the entries for all the keys in ascending order, and few records will be added later, a small adjustment factor of 1.1 can be used. If the file is created sequentially, but more records are to be added, use an adjustment factor of about 1.3 (or greater, if many records will be added). If the file is created with the entries for some of the keys occurring in random order, use an adjustment factor of 2.0.

3. Compute the size of the largest key entry by performing the following steps:
 - Find the size of the largest key in the record.
 - Round this size up to the next multiple of 6 characters, if it is not already a multiple of 6 characters.
 - Add 6 characters to provide space for the key entry's pointer to the data record.
4. Compute the desired block size by multiplying the desired number of keys per block (from step 2) by the size of the largest key entry (from step 3).

5. Round this desired block size up to the next multiple of the record size, if it is not already a multiple of the record size. This last step ensures that the block size chosen is suitable for storing the data records as well as the keys.

The block size calculated by this procedure provides two-level access, but its impact on the system must be determined before deciding that this block size is the correct block size to use. Consider, in particular, the effects of the block size on memory usage. Refer to "Effect of Block Size on Save Memory" in this section.

Calculating Actual Area Size

KEYEDIO uses the following algorithm to calculate the actual AREASIZE of KEYEDIO files:

1. Converts the declared AREASIZE value to the number of blocks by dividing the AREASIZE value by the number of records per block.
2. Multiplies the result by the actual block size in segments.
3. If the resulting AREASIZE value is too small, assigns a size sufficient for one block per area.

The KEYEDIO file does not actually contain an area equal to the product of AREASIZE times AREAS, because many of the blocks in the file are used to hold index tables for the keys.

KEYEDIO Procedures

The GENERALSUPPORT library makes KEYEDIO library procedures available indirectly to programs in languages such as ALGOL and Pascal that have no direct interface to the KEYEDIO library. The KEYEDIO procedures that are exported from the GENERALSUPPORT library are described in the following subsections.

The library and the desired procedure should be declared as follows.

```
LIBRARY ISAMLIBRARY (LIBACCESS = BYFUNCTION,  
                    FUNCTIONNAME = "GENERALSUPPORT");  
REAL PROCEDURE ISMGETKEYSTRUCTURE (ISAMFILE, KEYINFO, OPTION, OFFSET);  
    VALUE OPTION, OFFSET;  
    FILE ISAMFILE;  
    ARRAY KEYINFO[0];  
    REAL OPTION, OFFSET;  
    LIBRARY ISAMLIBRARY;  
  
REAL PROCEDURE ISMOPEN (ISAMFILE, FILEINFO, OPENTYPE);  
    VALUE OPENTYPE;  
    FILE ISAMFILE;  
    ARRAY FILEINFO[0];  
    REAL OPENTYPE;  
    LIBRARY ISAMLIBRARY;
```

```
REAL PROCEDURE ISMCLOSE(ISAMFILE,CLOSETYPE);
  VALUE CLOSETYPE;
  FILE ISAMFILE;
  REAL CLOSETYPE;
  LIBRARY ISAMLIBRARY;

REAL PROCEDURE ISMSTART(ISAMFILE,KEYOFREF,KEYLEN,RECORD,CHOOZ);
  VALUE KEYOFREF,KEYLEN,CHOOZ;
  FILE ISAMFILE;
  REAL KEYOFREF,KEYLEN;
  ARRAY RECORD[0];
  REAL CHOOZ;
  LIBRARY ISAMLIBRARY;

REAL PROCEDURE ISMSEQUENTIALWRITE(ISAMFILE,RECORD);
  FILE ISAMFILE;
  ARRAY RECORD[0];
  LIBRARY ISAMLIBRARY;

REAL PROCEDURE ISMSEQUENTIALREAD(ISAMFILE,RECORD);
  FILE ISAMFILE;
  ARRAY RECORD[0];
  LIBRARY ISAMLIBRARY;

REAL PROCEDURE ISMRANDOMWRITE(ISAMFILE,RECORD);
  FILE ISAMFILE;
  ARRAY RECORD[0];
  LIBRARY ISAMLIBRARY;

REAL PROCEDURE ISMREWRITE(ISAMFILE,OPTION,RECORD);
  VALUE OPTION;
  FILE ISAMFILE;
  REAL OPTION;
  ARRAY RECORD[0];
  LIBRARY ISAMLIBRARY;

REAL PROCEDURE ISMDELETE(ISAMFILE,OPTION,RECORD);
  VALUE OPTION;
  FILE ISAMFILE;
  REAL OPTION;
  ARRAY RECORD[0];
  LIBRARY ISAMLIBRARY;

REAL PROCEDURE ISMRANDOMREAD(ISAMFILE,KEYOFREF,RECORD);
  VALUE KEYOFREF;
  FILE ISAMFILE;
  REAL KEYOFREF;
  ARRAY RECORD[0];
  LIBRARY ISAMLIBRARY;
```

KEYEDIO Support

```

REAL PROCEDURE ISMSETUPLIMIT(ISAMFILE,KEYOFREF,RECORDLEN,RECORD);
  VALUE KEYOFREF,RECORDLEN;
  FILE ISAMFILE;
  REAL KEYOFREF,RECORDLEN;
  ARRAY RECORD[0];
  LIBRARY ISAMLIBRARY;

```

The opening of keyed files requires two kinds of information for which no provision is made in nonkeyed files: key information and file access information.

Key Information

The key information describes the keys declared by your program. Each key is one word of information in an array parameter. The key information word format is shown in Table 7-1.

Table 7-1. Key Word Format

Name	Field	Value	Meaning
KEYFLAGF	[46:01]		Relative or keyed key
		0	Relative key
		1	KEYEDIO key
ALTERNATEKEYF	[45:01]		Alternate or primary
		0	Primary key
		1	Alternate key
DUPLICATEF	[44:01]		Duplicates
		0	No duplicates
		1	Duplicates
KEYORGANIZATIONF	[43:01]		Key organization
		1	Ascending. This key must be set. The value 0 (zero) for descending keys is not valid.
KEYSIGNPOSITIONF	[39:04]		Sign information
		0	No sign: alphanumeric data
		1	Leading separate: numeric data, leading separate sign
		2	Trailing zone: numeric data, trailing zone

continued

Table 7-1. Key Word Format (cont.)

Name	Field	Value	Meaning
		3	Leading zone: numeric data, leading zone sign
		4	Trailing separate: numeric data, trailing separate sign
		5	Operand
		6	Two's complement
KEYTYPEF	[35:04]		Type of key
		0	Word
		2	HEX field
		4	HEX or EBCDIC field
		8	ASCII or EBCDIC
KEYLENGTHF	[31:16]		Length in KEYTYPEF units
KEYOFFSETF	[15:16]		Offset in record in KEYTYPEF units

File Access Information

The ACCESSMODE attribute determines the way the file is accessed. The ACCESSMODE attribute is also contained in the file description. The values and mnemonics of the ACCESSMODE attribute are shown in Table 7-2.

Table 7-2. File Access Values

Value	Mnemonic	Meaning
0	SEQUENTIALACCESS	File access is sequential only
1	RANDOMACCESS	File access is random only
2	DYNAMICACCESS	File access is sequential and random

Results Returned

Results are returned from the operating system, the GENERALSUPPORT library, or KEYEDIO. The results returned by the procedures in GENERALSUPPORT are preceded by an asterisk (*). The results returned by the GENERALSUPPORT procedures will also have a value of 0 (zero) in bits 26:10 of the result. Results from KEYEDIO depend on the type of operation performed, such as OPEN, READ or

WRITE. Most of the frequently returned results are returned for each operation. Refer to the descriptions of I/O result enumerated values and open, close, and respond results in the *A Series File Attributes Programming Guide*.

ISMGETKEYSTRUCTURE Procedure

ISMGETKEYSTRUCTURE is used to inquire on the structure of the ISAM file.

For files in which the MAXRECSIZE is not known, the file attribute DEPENDENTSPECS should be set to TRUE before calling this procedure.

The ISMGETKEYSTRUCTURE procedure uses the following parameters to return the requested key information:

ISMGETKEYSTRUCTURE (ISAMFILE, KEYINFO, OPTION, OFFSET)

- ISAMFILE – The user's file.
- KEYINFO – An array that contains file and key information. The meanings of the values returned in word 0 (zero) of the array are as follows:

Field	Value	Meaning
[26:01]		Relative key option
	0	No relative key
	1	Relative key
[25:01]		Record unit
	0	Words
	1	Characters
[19:04]		Record length option
[15:08]		Start key pointer; index into this array for the start key
[7:08]		Number of keys

The meanings of the values returned in word 1 of the array are as follows:

Field	Meaning
[31:16]	Block size
[15:16]	Maximum record size

Words 2 through n+1 (1 greater than the number of keys) contain 1 word of key information for each key. See "Key Information" earlier in this section.

- OPTION – Indicates the type of information to be returned in the KEYINFO array.

Value	Meaning
0	Get all the key structure
1	Get primary key information

continued

continued

Value	Meaning
2	Get relative key information
3	Get key located at offset

- OFFSET – Indicates the record offset of the key to be reported on. OFFSET is only used when OPTION equals 3.

Error Results

The following results are returned by the GENERALSUPPORT procedure ISMGETKEYSTRUCTURE (note the *). Bit 0:1 also has a value of 1 and bit 26:10 has a value of 0 for these results.

- *[2:1] 1 = OPTION was not a valid value.
- *[5:1] 1 = The key was not found.

ISMOPEN Procedure

ISMOPEN opens a file that has its FILEORGANIZATION attribute equal to INDEXED or INDEXEDNOTRESTRICTED. If a new file is being created, the file is initialized and the key information is saved. Initialization includes creation of a key root table for each key defined by your program. If your program is accessing an existing file, your file declaration and key information are checked against those of the existing file, and your current record pointer and current key of reference are established.

The information in the FILEINFO parameter is used only when a new output file is opened. If file attributes other than the default values are necessary, the values can be set according to the information found in the *A Series File Attributes Programming Reference Manual*. The record unit, block size, and maximum record size values must be supplied in the FILEINFO parameter.

The ISMOPEN procedure uses the following parameters and returns an open result:

ISMOPEN (ISAMFILE, FILEINFO, OPENTYPE)

- ISAMFILE – The user’s file.
- FILEINFO – Information contained in the array is only returned when the OPENTYPE value is 2 (OUTPUT). When you open a file, the following information should be available in word 0 (zero):

Field	Value	Meaning
[25:01]		Record unit
	0	Words
	1	Characters
[7:08]		Number of nonrelative keys

The meanings of the values returned in word 1 of the array are as follows:

Field	Meaning
[31:16]	Block size
[15:16]	Maximum record size

Words 2 through n+1 (1 greater than the number of keys) contain 1 word of key information for each key. See “Key Information” earlier in this section.

- **OPENTYPE**— Specifies how the file is to be opened. **OPENTYPE** has the same value as the **FILEUSE** attribute.

Value	Meaning
1	Input
2	Output
3	Input/Output

If the open result is 1, the open was successful. Other odd numbered results are errors detected in the **ISMOPEN** procedure. These errors are given in the following text. For even-numbered results, refer to the explanation of the **OPEN** results in the *A Series File Attributes Programming Reference Manual*.

Error Results

Note that bit 0 is included in the value.

Field	Value	Meaning
*[3:4]	3	The length of FILEINFO was less than that required for the number of keys given.
	5	OPTION was not a valid value.
	7	An Error in key setting.
	9	The file was already open.
	11	Two primary keys were found.
	13	No primary key was found.
	15	The keys overlap.

ISMCLOSE Procedure

ISMCLOSE flushes buffers, unlocks any remaining locks (present due to abnormal termination), and closes the file.

The **ISMCLOSE** procedure requires the following parameters to return a **CLOSE** result:

ISMCLOSE (**ISAMFILE**, **CLOSETYPE**)

- ISAMFILE – The user’s file.
- CLOSETYPE – Specifies how the file is to be closed.

Value	Meaning
0,1	Rewind
2	No rewind
3	Save
4	Lock
5	Purge
6	Crunch

If the close result is 1, the close was successful. Other odd-numbered results are errors detected in the ISMCLOSE procedure. These errors are given below. For even-numbered results, refer to the explanation of the CLOSE Results in the *A Series File Attributes Programming Reference Manual*.

Error Results

Value	Meaning
*[2:1]	1 OPTION was not a valid value.
*[3:1]	1 File already closed.

ISMSTART Procedure

ISMSTART positions your current record pointer to the logical record currently in the file whose key satisfies the comparison. If the comparison is not satisfied by any record in the file, a “RECORD NOT FOUND” result is returned – bits 0 and 9 SET. A successful START operation establishes the RECORDKEY parameter as the key of reference. The key of reference is the key that is used on any subsequent sequential operations. No data is transferred during a start operation.

The ISMSTART procedure requires the following parameters to return a START result:

ISMSTART (ISAMFILE, KEYOFREF, KEYLEN, RECORD, CHOOZ)

- ISAMFILE – The user’s file.
- KEYOFREF – Specifies the key to be used for the START operation. If the KEYFLAGF (bit 46) is SET, a keyed START is indicated, and KEYOFREF specifies the key information format. If the KEYFLAGF is not SET, a relative START is indicated and the value contained in KEYOFREF is used as the relative key. The values of CHOOZ and KEYLEN are checked against the value stored in KEYOFREF.
- KEYLEN – The length of the key to be used in the START operation. KEYLEN must be less than or equal to the key-length field of the KEYOFREF. If KEYLEN is less than the key-length field of the KEYOFREF, a partial key start is indicated.
- RECORD – The user’s record area.

KEYEDIO Support

- **CHOOZ**—Specifies the type of start to be done, as shown in the following:

Value	Meaning
0	Start equal. Finds the key with the same value as the key in your record.
15	Start greater than or equal to. Finds the first key with a value greater than or equal to the key in your record.
20	Start greater than. Finds the first key with a value greater than the key in your record.

The **START** result values are listed as follows. The results with an asterisk (*) are returned by the **GENERALSUPPORT** procedure **ISMSTART**. The other results are returned by **KEYEDIO**.

Field	Value	Result [26:10]	Mnemonic	Meaning
[0:01]	1			An error occurred
	0			No error occurred
*[2:1]	1	0	NOERROR	The value in CHOOZ was not valid. Bit 0 also equals 1.
[9:1]	1	95	RECORDNOTFOUND	No key met the conditions Bit 0 will also equal 1.
		98	KEYISINVALID	The key is invalid. Bit 0 also equals 1.

ISMSEQUENTIALWRITE Procedure

ISMSEQUENTIALWRITE updates the file with your record and updates all key tables. Recovery information is saved before the key tables are updated. A **WRITE** operation physically updates the keyed file. **ISMSEQUENTIALWRITE** does not affect the current record pointer.

The **ISMSEQUENTIALWRITE** procedure requires the following parameters to return a sequential **WRITE** result:

ISMSEQUENTIALWRITE (**ISAMFILE**, **RECORD**)

- **ISAMFILE**—The user's file.
- **RECORD**—The user's record area.

Field	Value	Result [26:10]	Mnemonic	Meaning
[0:1]	1	100	DIFFERENTLENGTHRECORD	This record length is not the same as the record length in the file.
[5:1]	1	97	DUPLICATEKEYS	A duplicate key was found, and duplicate keys are not allowed.
[6:1]	1	101	PRIMARYKEYOUTOFORDER	The key was out of order. Sequential WRITE operations require the current key to be greater than the previous key. Bit 0 also equals 1.

The sequential write result values can be found in the *A Series File Attributes Programming Reference Manual* in the explanation of the STATE general file attribute.

ISMSEQUENTIALREAD Procedure

ISMSEQUENTIALREAD reads the record specified by the current record pointer. The current record pointer is then updated to point to the next record in the file.

The ISMSEQUENTIALREAD procedure requires the following parameters to return a sequential READ result:

ISMSEQUENTIALREAD (ISAMFILE, RECORD)

- ISAMFILE – The user's file.
- RECORD – The user's record area.

The sequential read result values are listed as follows:

Field	Value	Result [26:10]	Mnemonic	Meaning
[0:1]	0			No errors
	1	100	DIFFERENTLENGTHRECORD	This record length is not the same as the record length of the file.
[9:1]	1	46	ENDOFFILE	End of file. Bit 0 is also equal to 1.

ISMRANDOMWRITE Procedure

ISMRANDOMWRITE physically updates the KEYEDIO file with your record and updates all key tables. Recovery information is saved before the key tables are updated. ISMRANDOMWRITE does not affect the current record pointer.

The ISMRANDOMWRITE procedure requires the following parameters to return a random WRITE result. ISMRANDOMWRITE uses the primary key by default.

ISMRANDOMWRITE (ISAMFILE, RECORD)

- ISAMFILE – The user's file.
- RECORD – The user's record area.

Field	Value	Result [26:10]	Mnemonic	Meaning
[0:1]	1	100	DIFFERENTLENGTHRECORD	This record length is not the same as the record length of the file.
[5:1]	1	97	DUPLICATEKEYS	A duplicate key was found but duplicate keys are not allowed.

The random write result values can be found in the *A Series File Attributes Programming Reference Manual* in the explanation of the STATE general file attribute.

ISMRANDOMREAD Procedure

ISMRANDOMREAD reads the record specified by RECORDKEY. The key specified by RECORDKEY becomes the key of reference, and the current record pointer is updated to point to the next record.

The ISMRANDOMREAD procedure requires the following parameters to return a random READ result:

ISMRANDOMREAD (ISAMFILE, KEYOFREF, RECORD)

- ISAMFILE – The user's file.
- KEYOFREF – Specifies the key to be used for the read. If the KEYFLAGF (bit 46) is SET, a keyed read is indicated, and KEYOFREF has the key information word format. If the KEYFLAGF is RESET, a relative read is indicated, and the value contained in KEYOFREF is used as the relative key.
- RECORD – The user's record area.

The random READ result values are listed as follows:

Field	Value	Result [26:10]	Mnemonic	Meaning
[0:01]	0			No errors
	1	100	DIFFERENTLENGTHRECORD	This record length is not the same as the record length of the file.
[9:1]	1	95	RECORDNOTFOUND	Record not found. Bit 0 is also equal to 1.
		98	KEYSINVALID	The key is invalid. The KEYFLAGF was 0 indicating a relative key, but the file was not a relative file. Bit 0 is also equal to 1.

ISMREWRITE Procedure

ISMREWRITE rewrites the record specified by the primary key. If a serial rewrite is done, the next record specified by the current record pointer is rewritten. If a random rewrite is done, the primary key specifies the record to be rewritten. ISMREWRITE does not affect the current record pointer.

The ISMREWRITE procedure requires the following parameters to return a REWRITE result:

ISMREWRITE (ISAMFILE, OPTION, RECORD)

- ISAMFILE – The user’s file.
- OPTION – Specifies whether a serial or random rewrite is to be done.

Value	Meaning
0	Serial rewrite.
1	Random rewrite.

- RECORD – The user’s record area.

The REWRITE result values are listed in the following table. The results with an asterisk (*) are returned by the GENERALSUPPORT procedure ISMREWRITE. The other results are returned by KEYEDIO.

KEYEDIO Support

Field	Value	Result [26:10]	Mnemonic	Meaning
[0:1]	0			No errors
	1	99	LASTIOMUSTBEREAD	The last i/o was not a READ. A random I/O was attempted when the ACCESSMODE was SEQUENTIAL.
*[2:1]	1	0	NOERROR	The value in OPTION was not valid. Bit 0 is also equal to 1.
[6:1]	1	96	PRIMARYKEYSNOTEQUAL	The primary key was changed. Only alternate keys and data can be changed with ISMREWRITE. Bit 0 is also equal to 1.
[9:1]	1	95	RECORDNOTFOUND	The record was not found. No record met the key conditions. Bit 0 is also equal to 1.
		98	KEYSINVALID	The key is invalid. The KEYFLAGF was 0 indicating a relative key, but the file was not a relative file. Bit 0 is also equal to 1.

ISMDELETE Procedure

ISMDELETE deletes the specified record. The previous I/O operation must have been a successful READ if a serial delete is to be done. A serial DELETE removes the record previously read. A random DELETE removes the record specified by the primary key. ISMDELETE does not affect the current record pointer.

The ISMDELETE procedure requires the following parameters to return a DELETE result:

ISMDELETE (ISAMFILE, OPTION, RECORD)

- ISAMFILE – The user’s file.
- OPTION – Specifies whether a serial or random delete is to be done.

Value	Meaning
0	Serial delete
1	Random delete

- RECORD – The user’s record area.

The DELETE result values are listed in the following table. The results with an asterisk (*) are returned by the GENERALSUPPORT procedure ISMDELETE. The other values are returned by KEYEDIO.

Field	Value	Result [26:10]	Mnemonic	Meaning
[0:1]	0			No errors
	1	99	LASTIOMUSTBEREAD	The last i/o was not a read. A random i/o was attempted when the ACCESSMODE was SEQUENTIAL.
*[2:1]	1	0	NOERROR	OPTION was not a valid value. Bit 0 is also equal to 1.
[6:01]	1			Primary keys are not equal
[9:1]	1	95	RECORDNOTFOUND	The record was not found. No record met the key conditions. Bit 0 will also equal 1.
		98	KEYISINVALID	The key is invalid. The KEYFLAGF was 0 indicating a relative key, but the file was not a relative file. Bit 0 will also equal 1.

ISMSETUPLIMIT Procedure

ISMSETUPLIMIT defines the upper bounds of the file. This logical end-of-file (EOF) is set only for the key defined by KEYOFREF. The user’s RECORD contains the value of

the upper bound. If UPPERLIMIT has a value and an attempt is made to access beyond this limit, an end-of-file (EOF) condition is returned.

The ISMSETUPLIMIT procedure requires the following parameters to return a set-upper-limit result:

ISMSETUPLIMIT (ISAMFILE, KEYOFREF, KEYLEN, RECORD)

- ISAMFILE – The user’s file.
- KEYOFREF – Contains the key information used to carry out key comparisons. The key length must be set in [31:16]. Refer to the key information under “KEYEDIO Procedures.”
- KEYLEN – Specifies the length of the entire record.
- RECORD – The user’s record area.

The set-upper-limit result value is listed as follows:

Field	Value	Result [26:10]	Mnemonic	Meaning
[9:1]	1	98	KEYISINVALID	KEYOFREF did not match any of the keys. Bit 0 will also equal 1.

The KEYEDIO File Structure

The following text describes the structure of the KEYEDIO file.

Segment 0 (Zero) of the File

When a KEYEDIO file is created, the following information about the file is saved in segment 0 of the file. Words in segment 0 are specified as offsets from the standard block information area at the start of every block. Since the block information area is currently 10 words long, word 0 of segment 0 is actually located at word 10 of the block.

Word 0

Describes the physical characteristics of the file. The values stored in this word are calculated at file creation time based on your file declarations and space requirements needed for maintaining the file.

Field	Meaning
[47:08]	The offset into the recovery area where the contents of the new record are stored for recovery of a rewrite operation.
[39:08]	The default number of buffers to allocate when this file is opened. This default is established by the setting of the BUFFERS attribute when the file was created.

continued

continued

Field	Meaning
[31:16]	The size of the records contained in the file in words. This value might be different from your declared MAXRECSIZE because of relative keys. If your program has specified relative keys, the record size is adjusted to allow relative keys.
[15:16]	The block size of the file in words. This value is different from the user's declared block size.

Word 1

Contains information about the keys and the program that created the file. This word is copied directly from the FILEINFO [0] parameter in ISAMOPEN at file creation time.

Field	Value	Meaning
[47:04]		File format level (currently 1)
[43:04]		Status of file
	0	Closed
	1	Open input
	2	Open output
	3	Open input/output
	9	Locked
[39:08]		Language of program opening the file; same values as in the MCP language table.
[31:04]		Type of access to use on file
	0	Sequential
	1	Random
	2	Dynamic
[27:01]		Deleted record flag
	0	Deleted records are not visible
	1	Deleted records are visible
[26:01]		Presence of relative keys
	0	No relative keys
	1	Relative keys
[25:01]		Record units
	0	Words
	1	Characters
[15:08]		Relative index into FILEINFO of first key of first key attribute (in words)

continued

continued

Field	Value	Meaning
[19:04]		Record length flag
	0	Fixed
	1	Variable
[07:08]		Number of keys

Word 2

Specifies the timestamp of the last update. The timestamp is used in recovery. The current value of TIME (6) is stored in this word whenever the file is about to be updated. Every block that is written because of the update contains this timestamp. Refer to "Recovery Procedures" in this section.

Word 3

Specifies the segment that describes the keys – the key information table as follows:

Field	Meaning
[39:20]	Length of the key information
[19:20]	Relative segment number of key information block

Word 4

Not currently used.

Word 5

Specifies the first block in the file that has never been used. All blocks beyond the specified one are available for use.

Word 6

Specifies the next available record slot. When a record is added, the new record is stored at the location specified by this word.

Field	Meaning
[43:24]	The relative segment number of the current block
[19:20]	The offset into the block of the next record location

Word 7

Specifies the last user record to be updated. This word is only valid if nonzero. (Refer to "Recovery" in this section.)

Field	Meaning
[47:04]	The type of update in process
[43:24]	The relative segment number of the last updated block
[19:20]	The offset into the block of the last updated record

Word 8

The next relative key to be allocated. If relative keys have been specified, this word contains the next relative key value to be allocated.

Word 9

The MINRECSIZE value specified when the file was created.

Word 10

The MAXRECSIZE value specified when the file was created.

Word 11

The BLOCKSIZE value specified when the file was created.

Word 12

The FRAMESIZE value specified when the file was created.

Word 13

The BLOCKSTRUCTURE value specified when the file was created.

Word 14

The UNITS value specified when the file was created.

Word 15

The EXTMODE value specified when the file was created.

Block Information Layout

A keyed file is made up of data and tables. Data and tables have the same size, and each block contains the following control information:

Word 0

Contains the following information about the block, table organization, and key size:

Field	Value	Meaning
[47:04]		Type of block
	1	Coarse table
	2	Fine table
	3	Data block
[43:01]		Table organization (coarse and fine tables)
	0	Descending order by key
	1	Ascending order by key
[15:16]		Size of each key entry (coarse and fine tables)

Word 1

Contains the following information about the keys for coarse and fine tables:

Field	Meaning
[47:16]	First key entry (word offset)
[31:16]	Number of keys currently in table
[15:16]	Maximum number of key entries that can fit in the table

Word 2

Address (relative segment number) of this block.

Word 3

Address (relative segment number) of the next sequential fine table. This word links fine tables together so that the file can be accessed sequentially.

Field	Meaning
[43:24]	Relative segment number of the first available block

Word 4

Address (relative segment number) of the previous sequential fine table. This word maintains the sequential ordering of fine tables.

Field	Meaning
[43:24]	Relative segment number of the first available block

Word 5

Timestamp of the block. This word is used in recovery.

Coarse Table Layout

Coarse tables are made up of keys and table pointers. The key value is the value of the last key contained in the block specified by the table pointer. The table pointers are relative segment numbers into the file of the next table down in the structure. Table pointers are aligned on word boundaries.

Figure 7-1 shows a coarse table layout.

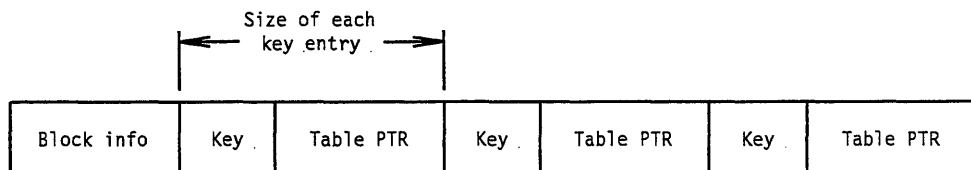


Figure 7-1. Coarse Table Layout

Fine Table Layout

Fine tables are made up of keys and pointers. The key value is the same value that the record specified by the data pointer contains. The pointers are relative segment numbers into the file and a word offset of where the data starts within the block. The data pointers have the following format:

Field	Meaning
[43:24]	The relative segment number of the block that contains the record
[19:20]	The word offset into the block of the beginning of the record

Figure 7-2 shows a fine table layout.

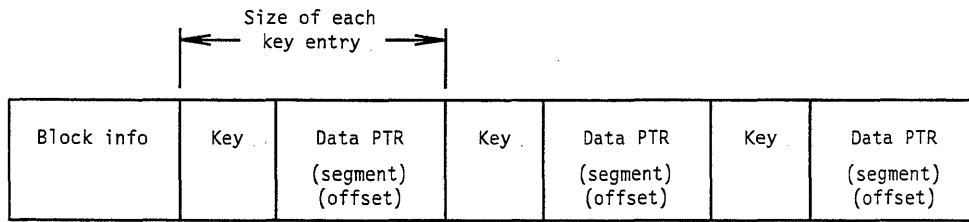


Figure 7-2. Fine Table Layout

Key Information Table Layout

The key information table contains 3 word entries that describe each key in the file and the first tables to be used in accessing the file with that key.

The first word of each key information table entry describes the key. It contains the same information in the same format as used for the ISAMKEYS file attribute discussed under “Indexed (KEYEDIO) File Attributes.”

The second word of each entry contains the relative segment address of the root table of the index tables for this key. If all the entries for this key will fit into a single table, the root table will be a fine table. Otherwise it will be a coarse table. The root table is the highest block in the hierarchical structure of index tables; it is the first table to be searched when accessing a file randomly.

The third word of each key information table entry contains the relative segment address of the first fine table for this key. It is used to find the first record when accessing the file sequentially.

Logical Layout of a KEYEDIO File

The logical layout of a KEYEDIO file is illustrated in Figure 7-3.

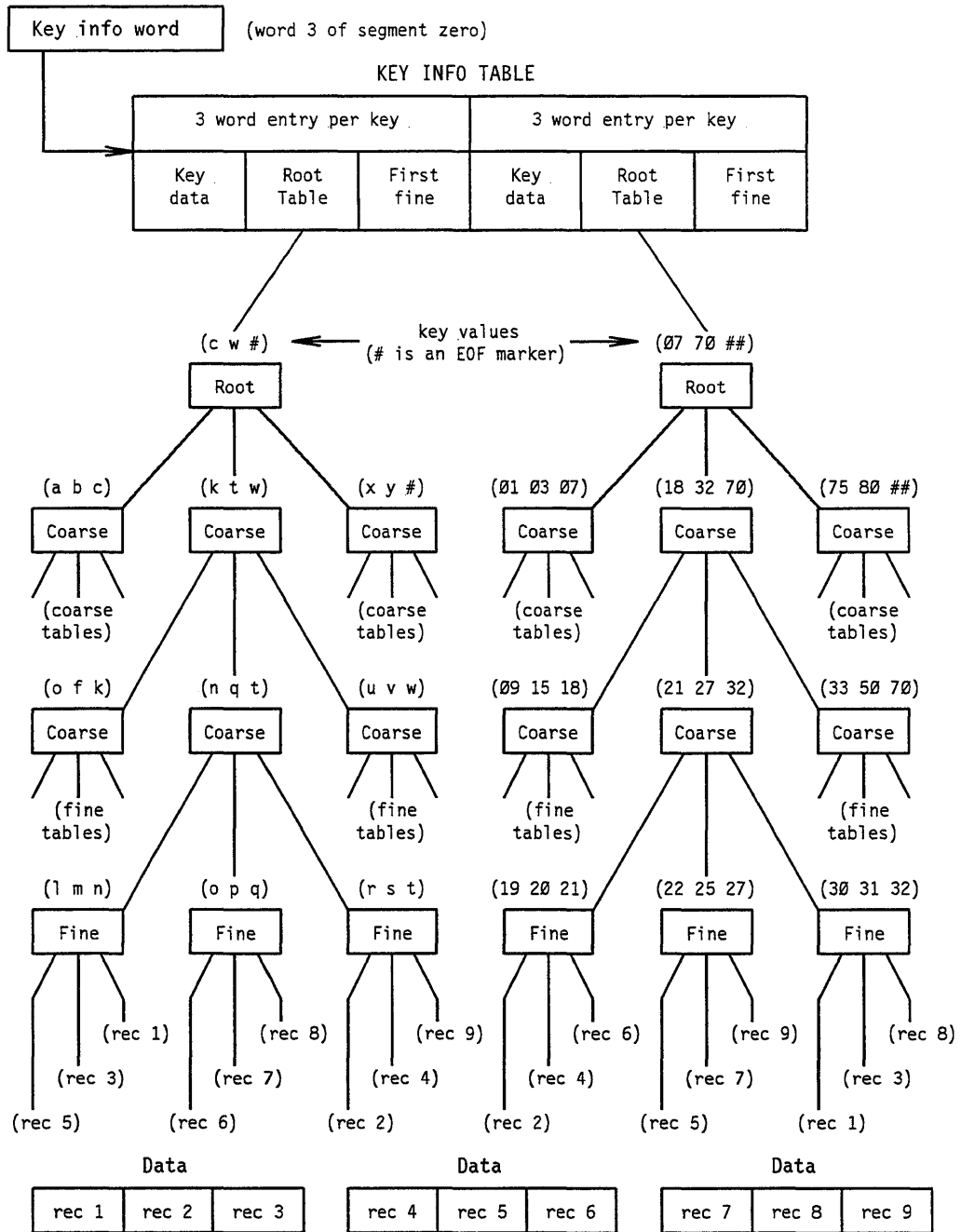


Figure 7-3. KEYEDIO File Layout

The end-of-file (EOF) marker is actually a key entry of all bits SET and a data pointer of zero.

Inserting Keys

Information contained in the block information is used for determining how to add the new key. The first entry in the table, the number of entries currently in the table, and the maximum number of entries that can fit in the table are saved in the block information. If the number of entries is less than the maximum number of entries, keys to the left of the new key are moved to the left, and the new key is inserted. To add a new key entry to a table that is already full, the table is split. When the split is made, all entries to the left of the new key and the new key go into one table, and all entries to the right go into another table.

Note: The tables are always right-justified.

Example of Inserting a Key:

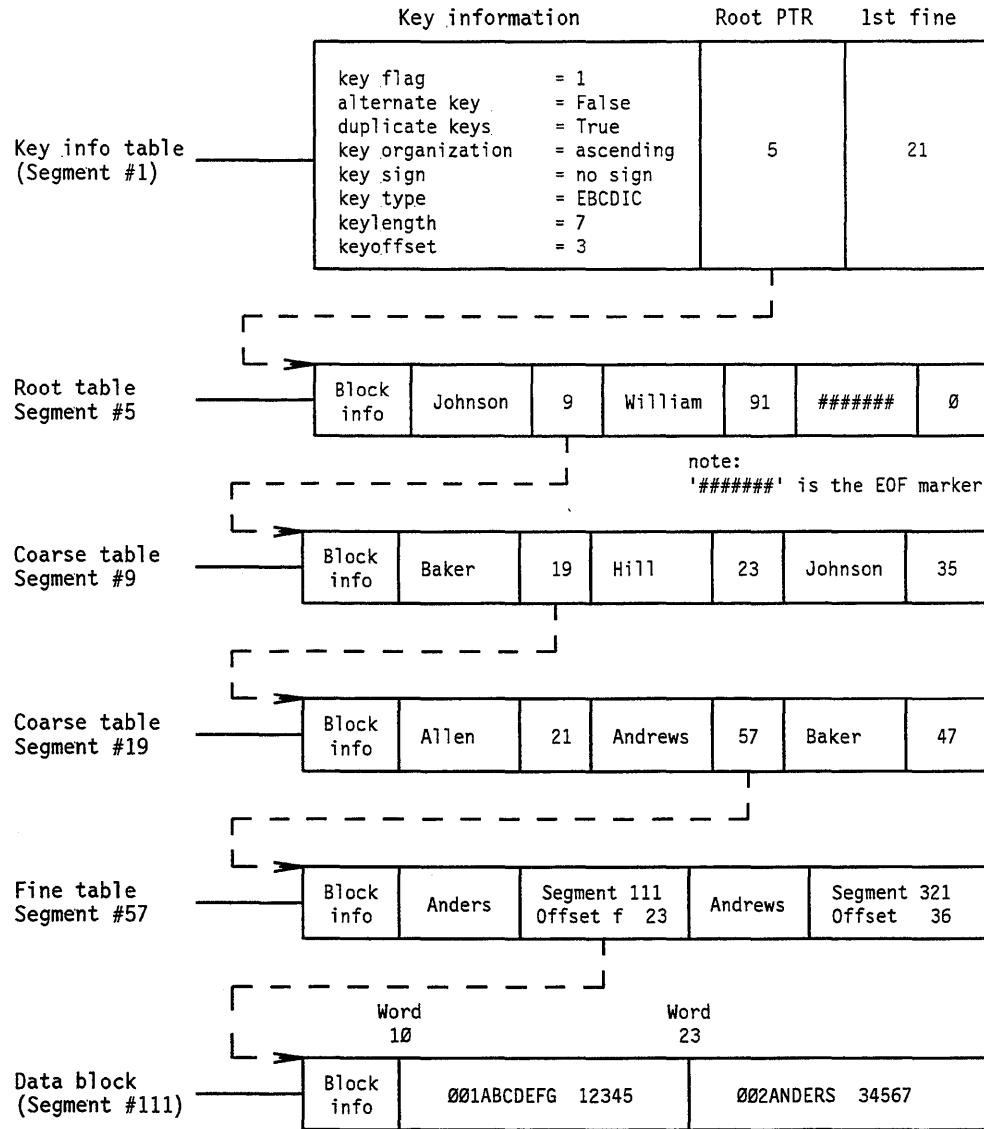
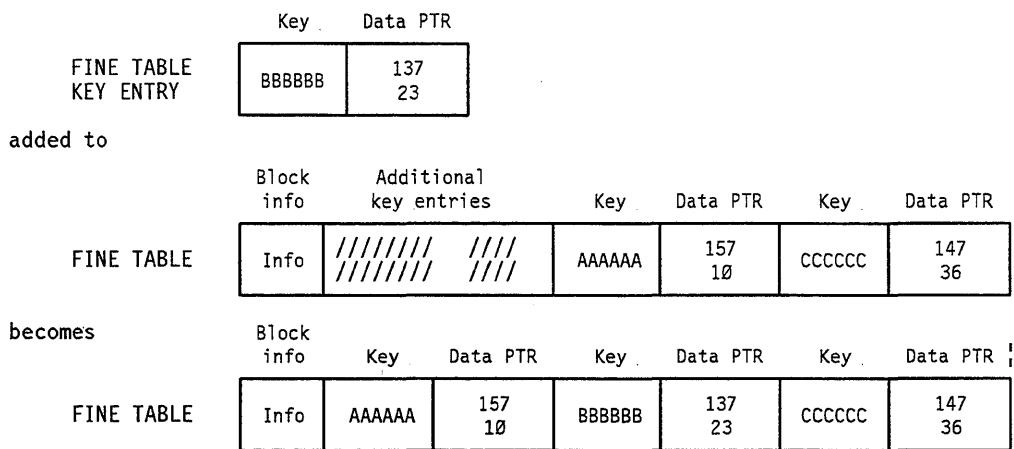


Figure 7-4. Inserting a Key

Example of Inserting a Key into a Full Table:

Example of inserting a key:



Example of inserting a key into a full table:

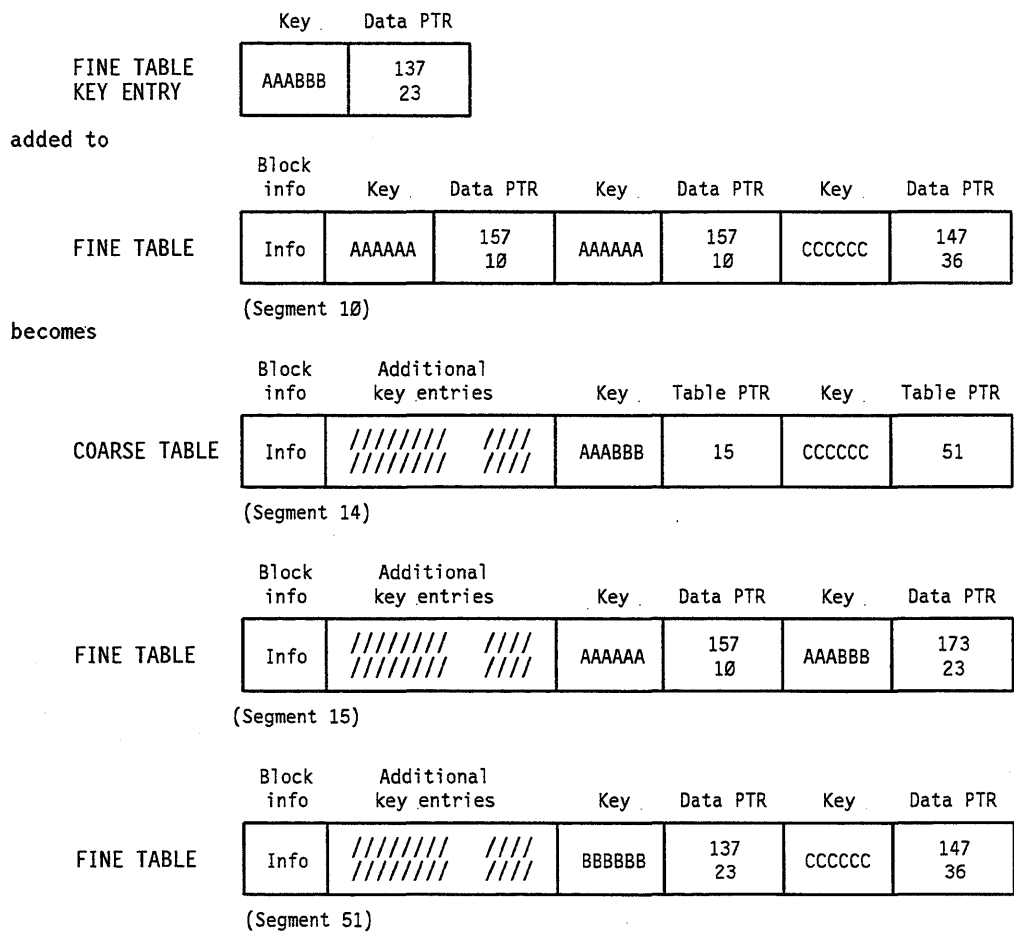


Figure 7-5. Inserting a Key into a Full Table

Recovery Procedures

To ensure that the file is always in a consistent state, recovery information is saved before any updating is done to the file. The recovery information consists of a timestamp and a pointer to the record that is being updated. Recovery information is not saved when the file is being created.

The following sequence of events takes place when a file is updated:

1. The recovery information is saved in segment 0 of the file.
2. Information is written to the file in a careful order so that data is not lost if the WRITE operations are not completed. The data record affected by the update is the last record changed.

If the update is terminated before step 2 is finished, the update is completed the next time the file is opened. The recovery process begins when the record referenced by the recovery information in segment 0 is read. The file is then updated with that record. During the update process, if a block is read that has a timestamp equal to that of the recovery timestamp, that block is treated as already reflecting the update. After the update is done, the recovery information is zeroed out, and segment zero is rewritten.

***Note:** Because enhancements were made for the Mark 3.7 release in the recovery procedure and in the way the recovery record is stored, the Mark 3.7 version or later of KEYEDIO should not be used to recover a file whose recovery record was stored by an earlier version of KEYEDIO. This will not be a problem if you make sure that all KEYEDIO files have been properly closed and do not need recovery before installing Mark 3.7 KEYEDIO.*

Recovery Messages and Warnings

The following warning message is displayed when a KEYEDIO file has Mark 3.3 PR1 or earlier recovery information:

```
FILE TOO OLD FOR HALTLOAD RECOVERY.  
FILE MUST BE RE-CREATED WITH 3.4.1 OR LATER KEYEDIO.
```

The recovery can still be attempted, if necessary. If KEYEDIO is unable to begin recovery with the information in the file, the following message is displayed:

```
INSUFFICIENT RECOVERY STATE.
```

If there is sufficient information to begin recovery but the recovery cannot be completed, the following message is displayed:

```
UNABLE TO RECOVER FILE.
```

KEYEDIO Support

For both of these conditions, you also receive the following message:

```
WARNING: FILE MAY BE CORRUPTED.  
THIS FILE SHOULD BE RELOADED USING 3.4.1 OR  
LATER KEYEDIO.  
'HI' TO CONTINUE, 'DS' TO ABORT.
```

Enter HI to resume processing and terminate the recovery attempt. Enter DS to discontinue the application.

Section 8

Mathematical Functions

This section describes the mathematical functions used in the A Series systems to support user-written mathematical intrinsics. Programs written in various languages can use these functions to perform mathematical operations. The compilers in which the functions are used cannot refer to the function by the names given to them in this section. (For example, in ALGOL, the function ARCTAN actually refers to the function ATAN described in this section.) The names used in this section are identical to those used in the *A Series FORTRAN Programming Reference Manual*. See “Permissible Argument Ranges” in this section for a list of function names used in various other languages.

The functions are grouped as single-precision, double-precision, and complex. A brief description of each function is given. In some instances this description is followed by the algorithm used in computing the function or notes regarding the derivation of the algorithm.

Certain constants such as π and e are used throughout the algorithms. These constants are defined under “Common Constants.” These values are stated in both single- and double-precision forms, where necessary. The appropriate value should be chosen according to whether the algorithm under consideration is single- or double-precision.

Several procedures are used in exponentiation and are called implicitly by the compilers. Because of their similarity, these procedures are grouped under the heading “Single-Precision Exponentiation,” “Double-Precision Exponentiation,” or “Complex Exponentiation.”

Single-Precision Functions

The following paragraphs describe the single-precision mathematical functions.

ALGAMA Function

The ALGAMA function accepts a positive real number and returns the natural logarithm of the GAMMA function at that number. The algorithm used varies depending on the value of argument x .

If x is less than 3.28, the following algorithm is used:

$$\text{ALGAMA}(x) = \text{ALOG}(\text{GAMMA}(x))$$

Mathematical Functions

If x is greater than or equal to 3.28, the calculation is more direct, relying on Stirling's approximation as follows:

$$\text{GAMMA}(x) = (e^{*-x} * x^{*(x-1/2)} * \text{SQRT}(2 * \pi)) g(x)$$

In the preceding calculation, the function $g(x)$ is an error polynomial.

ALOG Function

The ALOG function accepts any positive number and returns the natural logarithm (that is, the logarithm to base e) of that number. ALOG(x) returns a positive number if the argument x is greater than 1 and returns a nonpositive number if the argument x is between 0 and 1.

ALOG10 Function

The ALOG10 function accepts any positive number and returns the common logarithm—that is, the logarithm to the base 10 of that number. This function is computed by using the ALOG(natural logarithm) function in the following identity:

$$\text{ALOG10}(x) = \text{ALOG}(x) * \text{ALOG10}(e)$$

The value of the common logarithm of e is given under “Common Constants” later in this section.

ARCOS Function

ARCOS is the arccosine trigonometric function. ARCOS is also called the inverse cosine function. The ARCOS function accepts a number between -1 and 1 inclusive and returns the angle from the range 0 through $+pi$ that has that cosine.

The arccosine is calculated similarly to the arcsine function and uses the following identity:

$$\text{ARCOS}(x) = \pi/2 - \text{ARSIN}(x)$$

The value of $\pi/2$ is given under “Common Constants” later in this section.

ARSIN Function

ARSIN is the arcsine trigonometric function. ARSIN is also called the inverse sine. The ARSIN function accepts a number between -1 and 1 inclusive and returns the angle from the range $-\pi/2$ through $+\pi/2$ radians that has that sine.

ARSIN is calculated using a rational approximation for the argument x in the range 0 through 0.5 only. If x is outside this range, x is first reduced to being in the range in the following ways:

- If x is less than 0, the following identity is used:

$$\text{ARSIN}(x) = -\text{ARSIN}(-x)$$

- If x is greater than 0.86602, the following identity is used:

$$\text{ARSIN}(x) = \pi/2 - 2 * \text{ARSIN}(\text{SQRT}((1-x)/2))$$

- If x is greater than 0.5 and less than 0.86602, the following identity is used:

$$\text{ARSIN}(x) = \pi/4 + 1/2 * \text{ARSIN}(2 * x^{**3} - 1)$$

The value of $\pi/2$ is listed under “Common Constants” later in this section.

ATAN Function

ATAN is the arctangent function. ATAN is also called the inverse tangent. The ATAN function accepts any number and returns the angle from the range $-\pi/2$ through $+\pi/2$ radians that has that tangent.

ATAN2 Function

ATAN2 accepts any two numbers x and y and returns the arctangent of the quotient of those two numbers.

ATAN2 is defined for all real x and y values. This function is adapted to fall in the range of $-\pi$ through $+\pi$ by choosing it in a quadrant determined by the signs of x and y . In effect, this function is used in complex arithmetic as follows: given the complex number $x+iy$, ATAN2(x,y) returns the argument of that number between $-\pi$ and π . This function is calculated from the function ATAN, as follows:

- $\text{ABS}(x) < \text{ABS}(y)$ If $y > 0$ then $\text{ATAN2} = \text{ATAN}(x/y)$. If $y < 0$ then $\text{ATAN2} = \text{ATAN}(x/y) - \text{sign}(x) * \pi$.
- $\text{ABS}(x) \geq \text{ABS}(y)$ If $x \neq 0$ then $\text{ATAN2} = -\text{ATAN}(y/x) + \pi/2 * \text{sign}(x)$. If $x = 0$ then $\text{ATAN2} = \pi/2$.

COS Function

The COS function accepts a number that indicates the number of radians in an angle, and returns the cosine of that angle. The angle is first reduced to lie between 0 and $\pi/2$, using the following identities:

$$\begin{aligned} \text{COS}(-x) &= \text{COS}(x) \\ \text{COS}(2 * \pi + x) &= \text{COS}(x) \\ \text{COS}(\pi + x) &= \text{COS}(\pi/2 - x) \end{aligned}$$

Mathematical Functions

The final calculation is one of two polynomial approximations, depending on the value of the reduced argument.

COSH Function

The COSH function accepts a real number and returns the hyperbolic cosine of that number. Depending on the value of the argument x , $\text{COSH}(x)$ is computed by means of the following identities:

$$\begin{aligned}\text{COSH}(-x) &= \text{COSH}(x) \\ \text{COSH}(x) &= \text{EXP}(x - \text{ALOG}(2)) + .25/(\text{EXP}(x - \text{ALOG}(2)))\end{aligned}$$

COTAN Function

COTAN is the trigonometric cotangent function. The COTAN function accepts a number that indicates the number of radians in an angle, and returns the cotangent of that angle.

The method used for calculating the $\text{COTAN}(x)$ is identical to the method for calculating $\text{TAN}(x)$. However, because $\text{COTAN}(x)$ is the reciprocal of $\text{TAN}(x)$, the final calculation for $\text{COTAN}(x)$ is obtained by use of the following relationships:

Table 8-1. TAN/COTAN Calculation

Octant	COTAN(x)
0	T/R
1	R/T
2	(-R)/T
3	(-T)/R

ERF Function

ERF is the error function used in calculating probability. This function accepts any number and returns a value between -1 and 1. The error function is defined as follows:

$$\text{ERF}(x) = 2/\text{SQRT}(\text{pi}) * (\text{INTEGRAL}(e^{**((-t)**2)} dt) \text{from } 0 \text{ to } x)$$

This function is slightly different from the normal probability curve, Gauss's probability integral, which is defined as follows:

$$\text{phi}(x) = 1/\text{SQRT}(2*\text{pi}) * (\text{INTEGRAL}(e^{**(((t)**2)/2)} dt) \text{from } 0 \text{ to } x)$$

The relationship between ERF(x) and phi(x) is as follows:

$$\text{ERF}(x) = 2 * \text{phi}(\text{SQRT}(2) * x)$$

ERFC Function

ERFC is the complement of the ERF error function. ERFC(x) is defined as follows:

$$\text{ERFC}(x) = 1 - \text{ERF}(x)$$

EXP Function

The EXP function accepts any number less than 157.9 and returns *e* (the base of the natural logarithm) raised to that power as a real value. The EXP function is defined as follows:

$$\text{EXP}(x) = e^{**x}$$

The value of the constant *e* is given under “Common Constants” later in this section.

Single-Precision Exponentiation

Single-precision exponentiation is performed by the intrinsic RTOR—a real number to a real power.

GAMMA Function

The GAMMA function accepts any number less than or equal to 53.4, except 0 or a negative integer, and returns the value of the gamma function at that number. The GAMMA function is defined for positive numbers by the following integral:

$$\text{GAMMA}(x) = \text{INTEGRAL}((t^{**}(x - 1) * e^{**}(-t))dt) \text{ from } \emptyset \text{ to infinity}$$

RANDOM Function

The RANDOM function accepts a call-by-name integer and returns a pseudorandom real number that is greater than or equal to 0 and less than 1. The number is generated by the mixed congruent method, which is designed to give a uniform distribution.

RANDOM is the only function described in this section that takes a call-by-name input parameter. The parameter gives an initial value that is thereafter changed to give succeeding values by the procedure itself.

Mathematical Functions

You can obtain starting values to acquire a good sequence of pseudorandom numbers by picking odd integers close in value to 2^{19} , 2^{40} , or 2^{41} .

The procedure for RANDOM generates integers in the range 0 through $2^{39}-1$ and then returns those integers divided by 2^{39} . The integer variable N given to RANDOM is changed as follows:

$$\text{ABS}(N) := (A * \text{ABS}(N) + 116177073375) \text{ MOD } (2^{39})$$

The operator := is the replacement operator. A is a constant whose value depends on the sign of N (which is never changed), as follows:

- For nonnegative N: $A = 152587890725$
- For negative N: $A = 277626315293$

These values permit two different pseudorandom sequences depending on whether the starting value was positive or negative.

The random number is computed as follows:

$$\text{RANDOM}(N) = \text{ABS}(N) / (2^{39})$$

SIN Function

The SIN function accepts a number that indicates the number of radians in an angle and returns the sine of that angle. SIN(x) is always between -1 and 1 inclusive. The angle is first reduced to lie between 0 and $\pi/2$, by means of the following identities:

$$\begin{aligned}\text{SIN}(-x) &= -\text{SIN}(x) \\ \text{SIN}(2N\pi + x) &= \text{SIN}(x) \\ \text{SIN}(\pi + x) &= -\text{SIN}(\pi - x) \\ \text{SIN}(\pi/2 + x) &= \text{SIN}(\pi/2 - x)\end{aligned}$$

The final calculation is one of two polynomial approximations, depending on the value of the reduced argument.

SINH Function

The SINH function accepts a real number and returns the hyperbolic sine of that number. Depending on whether the value of the argument is less than 1 or not, SINH(x) is computed by means of either a polynomial approximation or the following identity:

$$\text{SINH}(x) = \text{EXP}(x - \text{ALOG}(2)) - 0.25 / (\text{EXP}(x - \text{ALOG}(2)))$$

SQRT Function

The SQRT function accepts any positive number and returns the square root of that number. The square root is always positive. The algorithm for deriving a square root is essentially the traditional Newton-Raphson method. However, an initial estimate is first derived.

TAN Function

The TAN function accepts a number that indicates the number of radians in an angle and returns the tangent of that angle. TAN(x) is either positive or negative depending on the argument x . To compute TAN(x), the argument x is reduced if it is outside the range 0 through π .

TANH Function

The TANH function accepts any number and returns the hyperbolic tangent of that number. TANH(x) for a real argument x is computed either by means of the following identities or by a rational approximation depending on the value of the argument:

$$\begin{aligned} \text{TANH}(-x) &= -\text{TANH}(x) \\ \text{TANH}(x) &= \frac{\exp(2x) - 1}{\exp(2x) + 1} \end{aligned}$$

Double-Precision Functions

The following paragraphs describe the double-precision mathematical functions.

Many double-precision functions are calculated in the same manner as the equivalent single-precision functions. For these functions, all references to single-precision functions are changed to references to double-precision functions, and all references to the arithmetic operations are assumed to be to double-precision operations.

DARCOS Function

DARCOS is the inverse cosine function. This function accepts a number between -1 and 1 inclusive and returns the angle from the range 0 through π that has that cosine.

DARCOS is calculated similarly to the arcsine function and uses the following identity:

$$\text{ARCOS}(x) = \pi/2 - \text{DARSIN}(x)$$

The value of $\pi/2$ is given under “Common Constants” later in this section.

DARSIN Function

DARSIN is the inverse sine function. This function accepts a number between -1 and 1 inclusive and returns the angle from the range $-\pi/2$ through $+\pi/2$ that has that sine.

DATAN Function

DATAN is the inverse tangent function. This function accepts a double-precision number and returns the angle that has that tangent. The argument x is reduced to the range $0 < x \leq 1$, where $1 = \tan(\pi/4)$.

DATAN2 Function

DATAN2 accepts any two numbers x and y and returns the arctangent of the quotient of those two numbers.

DATAN2 is defined for all real x and y values. This function is adapted to fall in the range of $-\pi$ through $+\pi$ by choosing it in a quadrant determined by the signs of x and y . In effect, this function is used in complex arithmetic as follows: given the complex number $x + iy$, DATAN2(x,y) returns the argument of that number between $-\pi$ and π .

DATAN2 is calculated from the function DATAN as follows:

- If y is equal to 0 , then

$$\text{DATAN2}(x,0) = \text{sign}(x) * \pi / 2$$

- If y is less than 0 , then

$$\text{DATAN2}(x,y) = \text{DATAN}(x/y) + \text{sign}(x) * \pi$$

- If y is greater than 0 , then

$$\text{DATAN2}(x,y) = \text{DATAN}(x/y)$$

In these calculations, $\text{sign}(x)$ is a function that has the value $+1$ if x is greater than or equal to 0 , otherwise the value is -1 .

DCOS Function

The DCOS function accepts a number that indicates the number of radians in an angle, and returns the double-precision value of the cosine of that angle.

DCOSH Function

The DCOSH function accepts a number and returns the double-precision value of the hyperbolic cosine of that number. Depending on the value of the argument x , DCOSH(x) is computed either directly from the definition by using the DEXP function or by a polynomial approximation.

The argument is reduced to the first quadrant by the method used in calculating the single-precision COS. The calculation then uses the same approximation as is used for DSIN and then uses the following identity:

$$DCOS(x) = DSIN(\pi/2 - x)$$

DERF Function

DERF is the double-precision error function used in calculating probability. This function accepts any number and returns a value between -1 and 1. DERF is defined the same as ERF, as follows:

$$DERF(x) = 2/\text{SQRT}(\pi) * (\text{INTEGRAL}(e^{(-t)^2} dt) \text{ from } 0 \text{ to } x)$$

DERFC Function

DERFC is the complement of the DERF double-precision error function. DERFC(x) is defined as follows:

$$DERFC(x) = 1 - DERF(x)$$

DEXP Function

The DEXP function is the double-precision exponential function. This function is calculated in the same manner as is the single-precision exponential function.

DGAMMA Function

The DGAMMA function accepts any number except 0 or a negative integer as an argument and returns the double-precision value of the gamma function at that number. The DGAMMA function is the double-precision equivalent of the GAMMA function, as follows:

$$DGAMMA(x) = \text{INTEGRAL}((t^{(x-1)})(e^{-t}) dt) \text{ from } 0 \text{ to infinity}$$

DLGAMA Function

The DLGAMA function accepts a number and returns the double-precision value of the natural logarithm – that is, the logarithm to base *e* of the GAMMA function at that number. The constant *e* is defined under “Common Constants” later in this section.

DLOG Function

The DLOG function accepts any positive number and returns the double-precision value of the natural logarithm – that is, the logarithm to base e of that number.

DLOG10 Function

The DLOG10 function accepts any number and returns the double-precision value of the common logarithm – that is, the logarithm to the base 10 of that number. This function is computed by means of the DLOG function in the following identity:

$$\text{DLOG10}(x) = \text{DLOG}(x) * \text{DLOG10}(e)$$

The value of DLOG10(e) is given under “Common Constants” later in this section.

DSIN Function

The DSIN function accepts a number that indicates the number of radians in an angle and returns the double-precision value of the sine of that angle. The argument is reduced to the first quadrant by the method used in calculating the single-precision sine. The calculation proceeds by means of a polynomial approximation.

DSINH Function

The DSINH function accepts a number and returns the double-precision value of the hyperbolic sine of that number. DSINH is computed either directly from the definition by means of the DEXPT function or by approximation, depending on the value of the argument – in the same manner as SINH.

DSQRT Function

The square root of a double-precision argument is computed by the following two steps:

1. The single-precision square root of the most significant half of the argument is computed by means of the algorithm for finding the single-precision square root.
2. One additional Newton-Raphson iteration, using the original double-precision argument, double-precision arithmetic, and the single-precision square root is computed. The exponent is shifted if necessary.

DTAN Function

The DTAN function accepts a number that indicates the number of radians in an angle and returns the double-precision tangent of that angle. DTAN(x) returns a positive or negative number depending on the value of x . To compute the tangent of an angle, the angle is reduced if it is outside the range 0 through π .

DTANH Function

The DTANH function accepts a number and returns the double-precision hyperbolic tangent of that number. The double-precision hyperbolic tangent is computed either directly from the definition by means of the intrinsic DEXP or by approximation, depending on the value of the argument (in the same manner as TANH).

Double Precision Exponentiation

Double precision exponentiation is performed in the same manner as single-precision exponentiation by means of two routines: RTOD (real-to-double) and DTOD (double-to-double). In either case, the result is double-precision.

Complex Functions

The following subsections describe the complex mathematical functions.

Definitions Used in Complex Function Descriptions

All the complex functions are derived by the use of the real functions. In these functions, several methods are used to write a complex number. When a complex number is described as a simple variable, the letter z is used. Several equivalent ways of writing z exist. In the following example, x and y are real numbers and i equals $\text{SQRT}(-1)$:

$$z = x + iy$$

The variable z can also be expressed as follows:

$$z = re^{i\phi}$$

The variables r and ϕ are the absolute value and the displacement, respectively. They are related to x and y as follows:

$$\begin{aligned}x &= r \cos(\phi) \\y &= r \sin(\phi) \\r^2 &= x^2 + y^2 \\ \tan(\phi) &= x/y\end{aligned}$$

These identities also indicate DeMoivre's formula as follows:

$$e^{i\phi} = \cos(\phi) + i\sin(\phi)$$

These basic relationships are used to determine most of the complex algorithms.

CABS Function

The CABS function accepts a complex number and returns the absolute value of that number. The absolute value of a complex number z is defined to be $ABS(r)$. Refer to “Definitions Used in Complex Function Descriptions” earlier in this section.

Therefore, given the following equation:

$$CABS(x + iy) = \text{SQRT}(x^{**2} + y^{**2})$$

If $ABS(x) \geq ABS(y)$, the following identity is used:

$$\text{SQRT}(1 + (y/x)^{**2}) * ABS(x)$$

If $ABS(x) < ABS(y)$, the following identity is used:

$$\text{SQRT}(1 + (x/y)^{**2}) * ABS(y)$$

The following special function is evaluated using iterations similar to Newton-Raphson iterations.

$$F(z) = \text{SQRT}(1 + z) - 1$$

The final computation is

$$CABS = F(z) * w + w$$

The variables z equals $(y/x)^{**2}$ or $(x/y)^{**2}$ and w equals $ABS(x)$ or $ABS(y)$.

CCOS Function

The CCOS function accepts a complex number and returns the cosine of that number. The cosine of a complex number z is calculated by using the identity for $COS(a+b)$ on the number $x+iy$. Then the following identities are applied:

$$COS(iy) = COSH(y)$$

$$SIN(iy) = iSINH(y)$$

These relationships are derived according to the definitions given under “Definitions Used in Complex Function Descriptions” earlier in this section. When the definitions of the hyperbolic sine and cosine are substituted in the equation, the algorithm becomes the following:

$$\begin{aligned} \text{CCOS}(x+iy) &= (+ \text{ or } -)1/2 \text{ SQRT}(1-\text{SIN}^{**2}(x)) (e^{**y} + e^{**(-y)}) - i/2* \\ \text{SIN}(x)*(e^{**y} - e^{**(-y)}) \end{aligned}$$

The value of x is taken modulo 2π before the sine function is applied. The negative sign is applied on the square root if the original x is in the second or third quadrants—that is, x is greater than or equal to $\pi/2$ and less than or equal to $3\pi/2$.

CEXP Function

The CEXP function accepts any complex number and returns the value of e raised to that number. The value of the exponent is calculated by the use of DeMoivre’s relationship. Refer to the information given under “Definitions Used in Complex Function Descriptions” earlier in this section. The basic identity that $\text{COS}^{**2}(x) = 1 - \text{SIN}^{**2}(x)$, is the following:

$$e^{**x+iy} = e^{**x} ((+ \text{ or } -)\text{SQRT}(1-\text{SIN}^{**2}(y))+i \text{ SIN}(y))$$

The value of y is taken modulo $2(\pi)$ before the sine function is applied. The negative sign on the square root is chosen if the original y is in the second or third quadrant.

CLOG Function

The CLOG function accepts a complex number and returns the natural logarithm—to base e —of that number. The complex natural logarithm of a number is calculated by means of DeMoivre’s relationship and the relationships among x , y , r , and ϕ as follows:

$$\text{CLOG}(x + iy) = \text{ALOG}(r) + i*\phi$$

In this algorithm, ϕ is chosen to fall in the principal range noted, and both r and ϕ are as previously defined. The natural logarithm is computed as a real number. The value of ϕ is computed by the real intrinsic ATAN2, which is designed for use in this application. Then the algorithm becomes the following:

$$\text{CLOG}(x+iy) = \text{ALOG}(\text{SQRT}(x^{**2} + y^{**2})) + i \text{ ATAN2}(y,x)$$

Because the complex logarithm is not a single-value function, the value returned by CLOG is in the range $-\pi$ through $+\pi$.

CSIN Function

The CSIN function accepts a complex number and returns the sine of that number. The sine of a complex number z is calculated by the use of the identity for $SIN(a+b)$ on the complex number $x+iy$. Then the identities $COS(iy) = COSH(y)$ and $SIN(iy) = SINH(y)$ are applied. Refer to the definitions under “Definitions Used in Complex Function Descriptions” earlier in this section. When the definitions of the hyperbolic sine and cosine are substituted in the equation, the algorithm is as follows:

$$CSIN(x+iy) = (+or-)1/2*SQRT(1-COS**2(x))*(e**y+e**(-y)) + i/2 COS(x)*(e**y-e**(-y))$$

The value of x is taken modulo 2π before the COS function is applied. The negative sign is applied on the square root if the original x was in the third or fourth quadrant.

CSQRT Function

The CSQRT function accepts a complex number and returns the complex square root of that number. The complex square root of a number is calculated first by means of DeMoivre’s relationship and then taking its square root, as follows:

$$CSQRT(z) = (r)**1/2(COS(phi/2) + i SIN(phi/2))$$

Using the half-angle formulas for the cosine and sine and rearranging the preceding relationship, the following equation is derived:

$$CSQRT(z) = SQRT(r(1+COS(phi))/2) + (SQRT(r(1-COS(phi))/2)*i)$$

The identity $x/r = COS(phi)$ and algebraic manipulation result in the algorithm that is used.

If $x \geq 0$ and $r = CABS(x+iy)$, then

$$CSQRT(x+iy) = SQRT((r+x)/2) + iy/(2 SQRT((r+x)/2))$$

If $x < 0$, then the trigonometric functions in polar form are complemented, and the following algorithm results, where $r = CABS(x+iy)$:

$$(SQRT(x+iy) = y/(2*SQRT((r+ABS(x))/2)) + i SIGN(y)*SQRT((r+ABS(x))/2)$$

SIGN(y) is a function that has the value +1 if y is not negative and the value -1 otherwise.

Complex Exponentiation

Exponentiation of a complex number is performed by two routines: CTOR, for complex numbers to a real power, and CTOD, for complex numbers to a double-precision power. The only difference between the double-precision power and the real power is that computations are performed by the use of the double-precision functions. Because the final result must be a complex number and no double-precision complex type is supported by any computers, exponentiation to a double-precision power might result in little increased accuracy at a high cost in time, depending on the particular case.

Common Constants

The following table lists common constants used in computing the mathematical functions. Real and, where necessary, double-precision values are given for each of the constants. Because fewer double-precision functions than real functions exist, some of the constants are unnecessary in the double-precision cases.

Table 8-2. Common Constants

Constant	Single-Precision Value	Double-Precision Value
pi	3.14159265359	3.1415926535897932384626
pi/2	1.57079632697	1.5707963267948966192313
pi/4	0.785298163397	
pi/6	0.523598775598	0.52359877559829887307711
3(pi)/4	2.35619449019	
SQRT(3)	1.732050807570	1.7320508075688772935275
SQRT(2)/2	0.707106781187	
ALOG(2)	0.693147180560	0.69314718055994530941723
ALOG(SQRT(2pi))	0.91893853321	
e	2.71828182846	2.7182818284590452353603
ALOG10(e)	0.434294481903	0.43429448190325182765113
LOG2(e)	1.4426950409	
TAN(pi/40)	0.0787017068246	
TAN(pi/20)	0.158384440326	
TAN(3pi/40)	0.240078759080	
TAN(pi/10)	0.324919696234	

Permissible Argument Ranges

Table 9–3 lists the permissible argument ranges for each of the mathematical functions. The word *All* signifies that all single-precision numbers – or double-precision numbers if the function is double-precision – are permitted. *All* is often modified in some obvious manner. If a function has more than one argument, the requirements for each are listed separated by commas (.). The notation (0,0) means that both the first and second arguments are 0 (zero).

Table 8–3. Permissible Argument Ranges

Function Name	Permissible Argument Range
ALGAMA	All positive
ARCOS	All <-1 and > +1
ARSIN	All <-1 and > +1
ATAN	All
ATAN2	All except (0,0)
COS	ABS < 10**24
COSH	All
COTAN	All
ERF	All
EXP	All
EXPONENT(RTOR)	All for exponent; all for base except negative numbers to nonintegral exponent
GAMMA	All except negative integers and 0
ALOG	All positive
ALOG10	All positive
RANDOM	ABS < 2**39
SIN	ABS < 10**24
SINH	All
SQRT	All nonnegative
TAN	ABS < 10**22
TANH	All
DATAN	All
DATAN2	All except (0,0)
DCOS	ABS < 10**24
DEXP	All

continued

Table 8-3. Permissible Argument Ranges (cont.)

Function Name	Permissible Argument Range
DLOG	All positive
DLOG10	All positive
DSIN	$ABS < 10^{**24}$
DSQRT	All nonnegative
EXPONENT	For both RTOD and DTOD:
(RTOD)	All for exponent
(DTOD)	All nonnegative for base
CABS	All
CCOS	All
CEXP	All
CLOG	All
CSIN	All, $-107 < y < 159$
CSQRT	All
EXPONENT	For both CTOR and CTOD:
(CTOR)	All for base;
(CTOD)	All real or double-precision for exponent

The following table lists the function names in various languages. The names used in this section are the FORTRAN names.

Table 8-4. Function Names

FORTRAN	FORTRAN77	ALGOL	COBOL	BASIC	PASCAL
ALGAMA		LNGAMMA			LNGAMMA
ALOG	ALOG	LN	LN	LOG	LN
ALOG10	ALOG10	LOG			LOG
ARCOS	ACOS	ARCCOS			ARCCOS
ARSIN	ASIN	ARCSIN			ARCSIN
ATAN	ATAN	ARCTAN	ARCTAN	ATN	ARCTAN
ATAN2	ATAN2	ARCTAN2			
CABS	CABS	CABS			

continued

Mathematical Functions

Table 8-4. Function Names (cont.)

FORTTRAN	FORTTRAN77	ALGOL	COBOL	BASIC	PASCAL
CCOS	CCOS	CCOS			
CEXP	CEXP	CEXP			
CLOG	CLOG	CLN			
COS	COS	COS	COS	COS	COS
COSH	COSH	COSH			COSH
COTAN		COTAN		COT	COTAN
CSIN	CSIN	CSIN			
CSQRT	CSORT	CSQRT			
DARCOS	DACOS	DARCCOS			
DARSIN	DASIN	DARCSIN			
DATAN	DATAN	DARCTAN			
DATAN2	DATAN2	DARCTAN2			
DCOS	DCOS	DCOS			
DCOSH	DCOSH	DCOSH			
DERF		DERF			
DERFC		DERFC			
DEXP	DEXP	DEXP			
DGAMMA		DGAMMA			
DLGAMA		DLGAMMA			
DLOG	DLOG	DLN			
DLOG10	DLOG10	DLOG			
DSIN	DSIN	DSIN			
DSINH	DSINH	DSINH			
DSQRT	DSORT	DSQRT			
DTAN	DTAN	DTAN			
DTANH	DTANH	DTANH			
ERF		ERF			ERF
ERFC		ERFC			
EXP	EXP	EXP	EXP	EXP	EXP
GAMMA		GAMMA			GAMMA
RANDOM	RANDOM	RANDOM			RANDOM

continued

Table 8-4. Function Names (cont.)

FORTRAN	FORTRAN77	ALGOL	COBOL	BASIC	PASCAL
SIN	SIN	SIN	SIN	SIN	SIN
SINH	SINH	SINH			SINH
SQRT	SQRT	SQRT	SQRT	SQR	SQRT
TAN	TAN	TAN		TAN	TAN
TANH	TANH	TANH			TANH

Section 9

PATCH Utility

The patch merge program (SYSTEM/PATCH) is a utility program used to merge one or more patch decks into a single patch deck on disk or pack. This disk file can then be used as the input CARD file for an ALGOL, BASIC, COBOL, COBOL74, DCALGOL, FORTRAN, FORTRAN77, Network Definition Language II (NDLII), Pascal, PL/I, or RPG compilation.

The PATCH utility merges all input patch records by sequence number. Only numeric or blank sequence numbers are accepted. The PATCH utility permits resequencing and patching into resequenced areas of a patch.

Running the PATCH Utility

The PATCH utility can be run through either Work Flow Language (WFL) or the Command and Edit (CANDE) message control system (MCS).

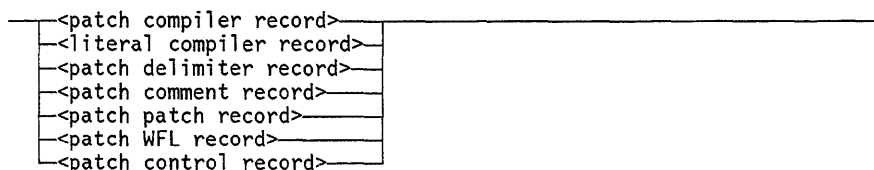
Using a WFL Job to Run the PATCH Utility

The following WFL job can be used to initiate the PATCH utility:

```
BEGIN JOB PATCH;  
  RUN SYSTEM/PATCH;  
  .  
  .  
  <file equation>;  
  .  
  .  
  DATA  
  .  
  .  
  <input record>  
  .  
  .  
  <i>END JOB.
```

PATCH Utility

<input record>



Explanation

<file equation>

Contains the necessary file equations for the files discussed later in this section. The files to be used in the \$.DISK, \$.DISK \$, \$.FILE, \$.PATCHDECK, and \$.INSERT options can also be file-equated in this section of the WFL job. Refer to the *A Series Work Flow Language (WFL) Programming Reference Manual* for a definition of file equation.

<input record>

Contains the data necessary to run the PATCH utility. The input record consists of patches, dollar records and comments.

Considerations for Use

When running the PATCH utility using a WFL job, you can interrogate the TASKVALUE attribute of the PATCH utility task to determine if errors were detected in the patches. If TASKVALUE is not equal to 1, errors have been detected.

Using CANDE to Run the PATCH Utility

The following CANDE command causes an interactive run of the PATCH utility to be initiated:

```
RUN *SYSTEM/PATCH; <file equation>
```

After the command is entered, the PATCH utility responds as follows:

```
ENTER INPUTS--
```

The information for the patch can then be entered. Any of the patch control records can be used when running the PATCH utility in interactive mode. The debug options are especially useful in interactive mode. An example of how to run the PATCH utility interactively is given under "Examples of PATCH Utility Input" later in this section.

Files Used by the PATCH Utility

The PATCH utility uses the following files, which can be file-equated in the file equation section of the input to SYSTEM/PATCH. The internal names for these files can have multiple nodes. If these files are not file-equated, they are assumed to be named CARD, LINE, NEWSOURCE, NEWTAPE, PATCH, PATCHES, SOURCE, and TAPE.

Any files referenced by the option \$.DISK, \$.DISK \$, \$.FILE, \$.PATCHDECK, or \$.INSERT can also be file-equated in the file equation section. The internal names for these files are described as follows. These file names can have only one node; that is, they cannot contain slashes.

File Name	Description
CARD	The input card file containing patches to be merged by the PATCH utility and other instructions to the PATCH utility.
LINE	The output printer file.
NEWSOURCE	The output disk file created when the PATCH file and the SOURCE or TAPE file are merged. For a SOURCE or TAPE file that is a BASIC, FORTRAN, or PL/I symbol file, the PATCH utility uses a specified NEWTAPE file, or lacking a NEWTAPE file, creates one. For all other SOURCE or TAPE files, if a file equation exists for both NEWSOURCE and NEWTAPE, NEWSOURCE is used. If NEWSOURCE or NEWTAPE is used alone, the specified name is used. NEWSOURCE is a synonym for NEWTAPE.
NEWTAPE	See NEWSOURCE. A synonym of NEWSOURCE. For a SOURCE or TAPE file that is a BASIC, FORTRAN, or PL/I symbol file, the PATCH utility uses a specified NEWTAPE file, or lacking a NEWTAPE file, creates one. For all other SOURCE or TAPE files, if a file equation exists for both NEWSOURCE and NEWTAPE, NEWSOURCE is used. If NEWSOURCE or NEWTAPE is used alone, the name specified is used.
PATCH	The output disk file containing the merged patches.
PATCHES	The output disk file containing the input specified by the \$.OUT option. Refer to “\$.OUT Option”.
SOURCE	The symbolic disk file to which the patches and \$ options of GO TO, SEQ, and MERGE, and the \$. options of INSERT, MOVE, COMPARE, LISTN, CYCLE, VERSION, and NEW are applied. If a file equation exists for both SOURCE and TAPE, SOURCE is used. If SOURCE or TAPE is used alone, the specified name is used. SOURCE is a synonym for TAPE.
TAPE	See SOURCE.

Patch Control Records

Seven categories of patch control records are acceptable as input to the PATCH utility. These categories are distinguished by a unique character or a blank immediately following the dollar sign.

The PATCH utility requires the following conditions:

- A \$# record must directly precede each patch.
- Within a patch (delimited by \$# records), all records not being resequenced must occur with increasing sequence numbers. Records that occur while \$SEQ is TRUE, or when the \$.INSERT (syntax 2) or the \$.MOVE (syntax 2) option is used, are not checked for the order of their sequence numbers.
- The \$ options SEQ, VOID, and VOIDT must be FALSE at the end of each patch.
- All input to the PATCH utility must be in uppercase characters.

The following text describes the different categories of patch control records.

Patch Compiler Control Records (\$ Records)

A compiler control record is one of the following:

- A record with a dollar sign (\$) in column 1—or in column 7 for COBOL, COBOL74, or COBOL85
- A record with a dollar sign in both columns 1 and 2—or in columns 7 and 8 for COBOL, COBOL74, or COBOL85
- A record with a blank in column 1 and a dollar sign in column 2—or a blank in column 7 and a dollar sign in column 8 for COBOL or COBOL74 but not for COBOL85

A dollar sign in any other column is not recognized. Normally, the PATCH utility protects a temporary compiler control record from being suppressed by a record with the same sequence number in a succeeding patch. This protection is removed for compiler control records that have nothing on them.

The interpretation of temporary and permanent compiler control records depends on the symbol input file equate (SOURCE or TAPE) as shown in the following table:

Equate	Column 1	Column 2	Compiler Control Record Type
TAPE	\$	Any character	Temporary
	Blank	\$	Permanent
SOURCE	\$	Any character except \$	Temporary
	\$	\$	Permanent

The PATCH utility handles the following compiler control options:

- DELETE
- SEQ
- SEQUENCE
- VOID

- VOIDT
- MERGE
- GO TO
- BUMP TO

These option functions are performed by the PATCH utility and are not passed to the compiler. For this reason, these options (except for MERGE) are erased from the record on which they appear before that record is written to the PATCH file. The PATCH utility creates SET and POP VOIDT records as needed to simulate the functions of these options. If a \$.COBOL74, \$.NDLII, \$.PASCAL, or \$.RPG record is included as input to SYSTEM/PATCH, then DELETE cards are created instead of VOIDT cards. SEQUENCE is a synonym for SEQ.

All other compiler options are passed to the compiler by way of the PATCH file, but are ignored by the PATCH utility. The following compiler options are checked for format, because their associated parameters could cause invalid actions if improperly specified:

- ERRORLIMIT
- INSTALLATION
- LEVEL
- LIMIT
- VERSION
- INCLUDE
- MAKEHOST

If no option value is specified, SET is assumed.

When handling permanent compiler options, PATCH passes \$ records and \$\$ records to the compiler with the \$ or \$\$ unchanged.

Refer to the appropriate language reference manual for the syntax and an explanation for all the compiler control options.

Patch Literal Compiler Records (\$& Records)

<literal compiler record>

— \$& —<compiler control record>—————|

\$& records are control records that you do not want the PATCH utility to handle. The PATCH utility replaces the ampersand (&) character with a blank and places the record into the PATCH file as shown in the following:

Example

```
$& SET SEQ 90000 + 1000
```

```
$ SET SEQ 90000 + 1000
```

Patch Delimiter Records (\$# Records)

<patch delimiter record>

```
— $# —<comment> —————  
                  |<control information>|
```

\$# records are patch delimiter records. Each individual patch within the input deck must be immediately preceded by a record with a **\$#** in columns 1 and 2—columns 7 and 8 when the COBOL control option is TRUE. The remainder of the record is for comments and control information.

The control information contains information necessary for the **\$.COUNT**, **\$.LABEL**, **\$.MARK**, and **\$.VERSION** options.

Patch Comment Records (\$: Records)

<patch comment record>

```
— $: —<comment> —————
```

\$: records contain comments about the patch. These records are listed if **LISTP** is TRUE and are written to the output **PATCHES** file if **OUT** is TRUE; otherwise, they are ignored. They can occur anywhere in the patch input. No limit exists on the number of records that can occur.

Patch Patch Records (\$- Records)

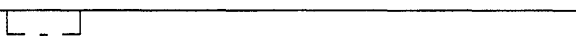
<patch patch record>

```
— $- —<patch> —————
```

\$- records are used to patch a patch. A patch patch record is treated as a regular record in that it must have a sequence number and can delete a record in a previous patch at that sequence number; however, it is not included in the **PATCH** file. The records let the original source filter through with the original patch number (if any) without changing an established patch or repunching the source and losing the patch number.

Patch WFL Records (\$* Records)

<patch WFL record>

— \$* —<WFL statement> 

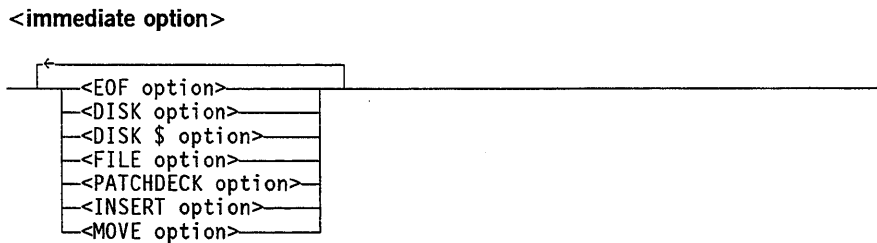
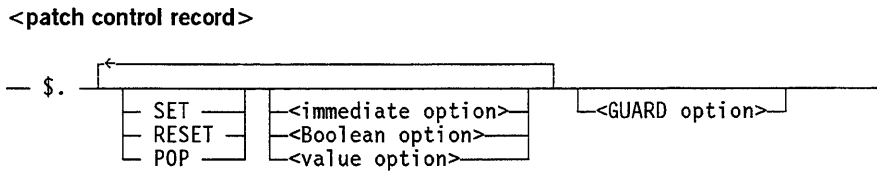
\$* records contain special WFL statements (without comments). The PATCH utility puts these records into an array and performs an *ALGOL ZIP WITH ARRAY* operation. The PATCH utility modifies the \$* records by placing a semicolon at the end of each statement and preceding each statement with a question mark (?) as shown in the following example. You can suppress this modification feature for that record by placing a hyphen (-) in column 80.

Example

```
$*RUN MYPROG ON MYPACK
```

```
?RUN MYPROG ON MYPACK;
```


Patch Control Records (\$. Records)

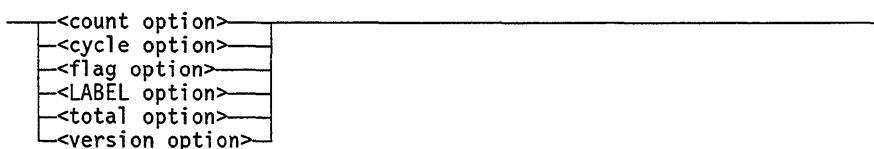


A Boolean option name is one of the following:

- BRIEF option
- COBOL option
- COBOL74 option
- COMPARE option
- COMPILE option
- CONFLICT option
- DELETE option
- DELIMOPT option
- DUMP option
- ERRLIST option
- EXECUTE option
- LABEL option
- LISTD option
- LISTI option
- LISTN option
- LISTP option
- MARK option
- MARKBLANK option

- NDLLI option
- NEW option
- OUT option
- PASCAL option
- RPG option
- SINGLE option
- SQUASH option

<value option>



Explanation

\$. records are control records to the PATCH utility that are similar to compiler control records. They are used to control the PATCH utility and are not included in the PATCH file. These records must be in all uppercase letters. The specific record options are described under “Patch Control Record Options” later in this section.

Additional \$. records are available if the PATCH utility was compiled with \$ SET DEBUG. For an explanation of these additional \$. records, refer to “Debug \$. Records” later in this section.

A \$. record must have a dollar sign (\$) in column 1 and a period (.) in column 2. The text can appear in columns 3 through 80 (9 through 80 if \$.COBOL or \$.COBOL74 is TRUE). Parsing of this text field is terminated by a percent sign (%). Unlike compiler options, no action is taken on options not specifically mentioned.

<immediate option>

Causes the PATCH utility to perform a function independent of the subsequent processing.

<Boolean option>

An option that is either enabled (TRUE) or disabled (FALSE). When enabled, it causes the compiler to apply an associated function to all subsequent processing until the option is disabled.

The keywords SET, RESET, and POP affect the setting of Boolean options in different ways. Each Boolean option has an associated stack in which up to 48 previous values of the option are saved. The management of the current value, along with the stack of previous values of the option, is as follows:

- If a Boolean option is the object of SET or RESET, the option is enabled or disabled (assigned a current value of TRUE or FALSE), respectively, and the previous value is pushed onto the stack. In other words, the option is assigned a new current value, and the previous values are saved in the stack.
- If a Boolean option is the object of POP, the current value of the option is discarded, and the previous values are moved up in the stack.
- If a Boolean option is not the object of an explicit SET, RESET, or POP, it is implicitly enabled (assigned a current value of TRUE), and all previous values saved on the stack for that option are discarded. In addition, the stacks for all other resettable standard Boolean options are discarded, and all such options are assigned a current value of FALSE. In other words, first the current value and all previous values of all resettable standard Boolean options are discarded; then the options appearing on the compiler control record are assigned a current value of TRUE.

<value option>

Nonimmediate options that cause the PATCH utility to store a value associated with a given function.

<guard option>

Has the same characteristics of an immediate option; that is, it performs a function independent of subsequent processing. However, the GUARD option must appear as the last option on the record.

Patch Control Record Options

The following text describes the \$. options in alphabetical order. These options must be in all uppercase letters.

\$.BRIEF Option

(Type: Boolean, Default: FALSE)

<\$.BRIEF option>

— \$.BRIEF —————|

The \$.BRIEF option is used with the \$.COMPARE option to suppress printing of more than six consecutive voided lines. The first record voided, the last record voided, and the number of records voided are printed instead.

\$.COBOL Option

(Type: Boolean, Default: FALSE)

<\$.COBOL option>

— \$.COBOL _____|

The \$.COBOL option tells the PATCH utility to expect input in COBOL format, that is, with sequence numbers in columns 1 through 6 and a dollar sign in column 7. When this option is TRUE, all special \$ records recognized by the PATCH utility must begin in column 7, but the record actually setting this option must have a \$. starting in column 1.

\$.COBOL74 Option

(Type: Boolean, Default: FALSE)

<\$.COBOL74 option>

— \$.COBOL74 _____|

The \$.COBOL74 option tells the PATCH utility to expect input in COBOL74 format—that is, sequence numbers in columns 1 through 6 and dollar signs in column 7. When the \$.COBOL74 option is TRUE, all special \$ records recognized by the PATCH utility must begin in column 7, but the record actually setting the \$.COBOL74 option must have a \$. starting in column 1. If the \$.COBOL74 option is TRUE, any \$DELETE or \$VOIDT records produce \$DELETE records. If COBOL74 is FALSE, \$VOIDT records are produced.

\$.COMPARE Option

(Type: Boolean, Default: FALSE)

<\$.COMPARE option>

— \$.COMPARE _____|

The \$.COMPARE option flags instances in which lines from two different patches are adjacent or interleaved at the same point in the source.

This option causes the PATCH utility to print a report comparing the PATCH utility input with the TAPE file. All patch records are listed. Source lines from TAPE are shown if they are deleted by the patch or if they appear immediately before or after a newly inserted line. Each line in the patch is identified in the right-most column of the report. Simple patch lines have the ordinal patch number; moved, resequenced, or inserted lines are identified with the original patch number—or as SOURCE—and with the modifying patch number. If lines from more than one patch fall adjacent to

each other with no intervening unmodified TAPE source lines, the final column is prefixed with a right angle bracket (>) character. These second-order conflicts often require investigation.

Because all \$ records from the patch decks are shown in the \$.COMPARE listing, the same flags appear if one patch makes insertions into an area voided by an earlier patch. If \$.BRIEF is TRUE, the listing of deleted sections of the TAPE file is abridged.

The \$.COMPARE option or the \$.LISTD option can be SET to list the patch delimiter \$# records but not the bodies of the patches. A COMPARE listing is produced only if the final value of \$.COMPARE is TRUE. However, any time that \$.COMPARE is TRUE while \$.LISTP is FALSE during the input phase, patch delimiter records are listed as if \$.LISTD were TRUE. Refer to “\$.LISTD Option” in this section.

\$.COMPILE Option

(Type: Boolean, Default: FALSE)

<\$.COMPILE option>

— \$.COMPILE _____|

The \$.COMPILE option causes the PATCH utility to automatically start the compilation of the TAPE or SOURCE file with the PATCH file if no fatal errors are discovered. See “Files Used by the PATCH Utility” for information on the selection of the SOURCE or TAPE file. The CARD and TAPE files are file-equated automatically by the PATCH utility. Other information for the compilation must be passed by \$* records supplied by you.

The \$.COMPILE and \$.EXECUTE options cannot be TRUE at the same time.

Example

```
$. SET COMPILE
$* COMPILE M/A WITH ALGOL LIBRARY
$* ALGOL FILE NEWTAPE (TITLE = S/M/A)
```

\$.CONFLICT Option

(Type: Boolean, Default: TRUE)

<\$.CONFLICT option>

— \$.CONFLICT _____|

The \$.CONFLICT option controls printing of patch conflicts – records deleted in previous patches by records in later patches. When the conflict option is set to TRUE, these conflicts are listed in the LISTP section of the output.

\$.COUNT Option

(Type: value, Default: FALSE)

<\$.COUNT option>

— \$.COUNT <integer>

If this option is SET or no value is specified, the \$.COUNT option must be followed by an unsigned integer number. If this option is RESET or POP, the \$.COUNT option must not be followed by a number. The number following the \$.COUNT option indicates a column on the \$# record and the number in that column indicates the number of records in the patch. The PATCH utility checks the number of records actually found against the number specified by the \$# record and issues an error if the numbers differ.

This option permits flexibility because different areas on the \$# record can be used to specify the record count for different patches. The \$ records are counted, the *non* \$ records are counted, and the \$. records with \$.MOVE or \$.INSERT options are counted.

\$.CYCLE Option

(Type: value, Default: FALSE)

Refer to “\$.VERSION and \$.CYCLE Options” in this section.

\$.DELETE Option

(Type: Boolean, Default: FALSE)

<\$.DELETE option>

— \$.DELETE <integer> , <integer> - <integer> - <integer>

When the \$.DELETE option is TRUE, the PATCH utility deletes the patches specified in the number list. Patches already processed are not affected. Each patch can have its \$.DELETE option specified as SET, RESET, or POP as desired. A deleted patch is listed if \$.LISTP is TRUE, but is otherwise ignored. Deleted patches are not included in the PATCHES file.

\$.DELIMOPT Option

(Type: Boolean, Default: FALSE)

<\$.DELIMOPT option>

— \$.DELIMOPT —————|

The \$.DELIMOPT option insulates succeeding patches from option changes in a particular patch. Whenever \$.DELIMOPT is SET—implicitly or explicitly—on a \$. record, the option on the record is SET, and the current value is recorded for each of the following options:

- \$.CONFLICT
- \$.LISTP
- \$.MARK
- \$.MARKBLANK
- \$.OUT

Whenever a \$# patch delimiter record is read while \$.DELIMOPT remains TRUE, all these options are restored to the values they had when \$.DELIMOPT was last TRUE. The stacked historical values of these options are RESET; thus, a POP is equivalent to a RESET for an option like \$.LISTP that has been restored by a \$# record read while \$.DELIMOPT was SET.

A patch control record can specify SET, RESET, or POP for the \$.DELIMOPT option. If the patch control record specifies POP for the \$.DELIMOPT option, and this changes the \$.DELIMOPT option from RESET to SET, the option values from the most recent occasion when \$.DELIMOPT was SET are restored.

The simplest way to use \$.DELIMOPT is to SET it as the last option before the first \$# delimiter in the input set. Then the specified or default values of the several options apply to each patch in turn, even if a prior patch has changed one or more options.

Example

In the following example, the \$.MARKBLANK option applies to the input for patch one, but not for patches two or three. The \$.MARK option applies to patches one and three but not to patch two.

```
$.SET COMPARE MARK DELIMOPT
$#PATCH ONE
$.MARKBLANK
.
.
$#PATCH TWO
$.RESET MARK
.
.
$#PATCH THREE
```

\$.DISK Option

(Type: immediate)

<\$.DISK option>

```
— $.DISK ————|
      |<file title>|
      |<intname>  |
```

The \$.DISK option tells the PATCH utility to get input from the specified file. This option performs tasks similar to the tasks performed by the \$.FILE option, but with the following exceptions:

- The input from the specified file cannot contain \$.DISK, \$.DISK \$, \$.PATCHDECK, or \$.FILE options.
- Mark field information from the input file is preserved even if the \$.MARK option is SET. The mark information from the input file is used only if the input file records are long enough to contain mark information.

If either the \$.COBOL or \$.COBOL74 option is SET, the record must contain at least 80 characters for mark field information to be retained. The mark information is taken from columns 73 through 80.

In all other cases, the mark information is taken from columns 81 through 90 of the record, and all records must be a minimum of 90 characters in length. Records that are not at least 90 characters long have a blank value in the mark field after the patch is merged.

Other \$. options can appear on the same \$. record after the options \$.DISK, \$.DISK \$, \$.FILE, and \$.PATCHDECK.

PATCH Utility

The files used in the \$.DISK, \$.DISK \$, \$.FILE, and \$.PATCHDECK options can be file-equated in the file equation section of the input. The internal name must consist of only one node—that is, no slashes (/) can be used.

Example

The following example tells the PATCH utility to get input from the file NEW/EMPLOYEE/NUMBERS:

```
$.DISK NEW/EMPLOYEE/NUMBERS
```

The following WFL job shows a PATCH utility run that file-equates the file used in the \$.DISK option:

```
BEGIN JOB PATCHER;  
RUN SYSTEM/PATCH;  
FILE TAPE(TITLE = SYMBOL/SOURCE ON PACK01);  
FILE MYPATCH(TITLE = PATCH/SOURCE/23 ON PACK02);  
DATA CARD  
$#PATCH SEPARATOR CARD 1  
$ SET LIST MERGE NEW  
$#PATCH SEPARATOR CARD 2  
$.DISK MYPATCH COMPARE  
?  
END JOB.
```

\$.DISK \$ Option

(Type: immediate)

Refer to “\$.FILE, \$.DISK \$, and \$.PATCHDECK Options” in this section.

\$.DUMP Option

(Type: Boolean, Default: FALSE)

<\$.DUMP option>

— \$.DUMP —————|

If \$.DUMP is TRUE, and the first fatal error occurs in patch N ($N > 1$), the PATCH utility merges the first $n-1$ patches and locks them on a disk file. If the PATCH utility file had the title X/Y/Z, then this file has the title DUMP/X/Y/Z. The \$.DISK option can then be used to restart the merge operation without rereading the first $n-1$ patches.

\$.EOF Option

(Type: immediate)

<\$.EOF option>

— \$.EOF —————|

The \$.EOF option indicates the end of all input. \$.EOF can occur in any input file. The PATCH utility does not read any records following the EOF record.

The \$.EOF option is used to end an interactive run from a remote terminal. For more information about running the PATCH utility interactively, refer to the description under “Using CANDE to Run the PATCH Utility.”

SET or RESET context is ignored for \$.EOF.

\$.ERRLIST Option

(Type: Boolean, Default: TRUE)

<\$.ERRLIST option>

— \$.ERRLIST —————|

The \$.ERRLIST option is meaningful only during an interactive run from a remote terminal. If \$.ERRLIST is TRUE, all errors and warnings are displayed at the terminal. If \$.ERRLIST is FALSE, this listing is suppressed. If execution is not from a remote terminal, changing the value of \$.ERRLIST has no effect.

Refer to description of “Running the PATCH Utility” for more information about running the PATCH utility interactively.

\$.EXECUTE Option

(Type: Boolean, Default: FALSE)

<\$.EXECUTE option>

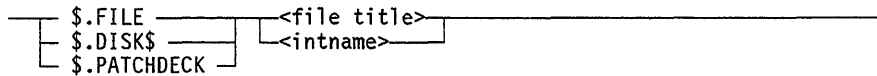
— \$.EXECUTE —<file name>—————|

The \$.EXECUTE option tells the PATCH utility to automatically start a specified program if no fatal errors are discovered. This option uses the \$* records in the same way the \$.COMPILE option does, except that no file equation occurs. The terminal END JOB control statement is supplied by the PATCH utility. EXECUTE and COMPILE cannot be TRUE at the same time.

\$.FILE, \$.DISK \$, and \$.PATCHDECK Options

(Type: immediate)

<\$.FILE option>
 <\$.DISK option>
 <\$.PATCHDECK option>



The \$.FILE, \$.DISK\$, and \$.PATCHDECK options are synonymous and are extensions of the CARD file. When one of these options is encountered, the PATCH utility reads from the specified file until it reaches end-of-file (EOF) or another \$.FILE, \$.DISK \$, \$.PATCHDECK, or \$.DISK option. If \$.LISTP is TRUE, the file title of the disk file from which a record is read is printed to the right of the sequence number in the printer listing.

The files used in the \$.DISK, \$.DISK \$, \$.FILE, and \$.PATCHDECK options can be file-equated in the file equation section of the input. The internal name must be only one node – that is, it cannot contain any slashes (/).

The \$.FILE option, and any options that are synonymous with this option, differs from the \$.DISK option in two important ways. The input from the \$.FILE option is marked if the \$.MARK option is set to TRUE. In addition, the specified input file may contain the \$.DISK option, the \$.DISK\$ option, the \$.PATCHDECK option, or the \$.FILE option and these options can be nested within other options up to 10 levels.

Examples

```
$.FILE MY/FILE ON MYPACK
$.PATCHDECK A/B
$.DISK $ MY/OTHER/FILE ON MYPACK
```

\$.FLAG Option

(Type: value, Default: FALSE)

<\$.FLAG option>

— \$.FLAG —<version number>— . —<cycle number>—————|

When the source file contains patch marks in the form of version and cycle numbers, the \$.FLAG option can be used in the COMPARE listing to call attention to neighboring or deleted lines that have marks greater than or equal to the specified values. Source lines whose marks begin *vv.ccc* or *vvccc* (where *v* and *c* are digits) are flagged if *vv* is greater than or equal to the version number and *ccc* is greater than or equal to the cycle number. If no cycle number is provided, the cycle number defaults to 0 (zero). The flag appears as an asterisk (*) preceding the mark field.

SET or RESET context is ignored for \$.FLAG. If more than one \$.FLAG value is provided, the last value specified is used throughout the COMPARE phase. \$.FLAG is ignored if \$.COMPARE is FALSE and is not fully effective if \$.BRIEF is TRUE.

Example

The following \$.FLAG option causes all source lines whose version number is greater than or equal to 33 and whose cycle number is greater than or equal to 320 to be flagged with an asterisk (*) preceding the mark field:

```
$.FLAG 33.320
```

\$.GUARD Option

(Type: immediate)

<\$.GUARD option>

```
— $.GUARD —<integer>— - —<integer>—<comment>—————|
```

<comment>

```
—<any character string>—————|
```

The \$.GUARD option causes all patch records within a specified sequence range to be flagged with the specified comment in a special report in the printer output. No more than 100 areas can be guarded in this manner. If a fatal error occurs, no \$.GUARD output is generated. The \$.GUARD option must be the last control option specified on the \$. record.

\$.INSERT Option

(Type: immediate)

Syntax 1

<#.INSERT option>

```
— $.INSERT —————<first>— - —<last>— AT —<baseinc>—————|
      |—————|
      |<file ID>|
```

Syntax 2

<\$.INSERT option>

```

— $.INSERT —————<first>— AT —<baseinc>—————|
      |—————|
      |<file ID>|
      |—————|
.
.
.
  <patch records to the inserted material>
.
.
| POP —————|
| RESET ————| INSERT —<last>—————|

```

<baseinc>

```

—<base>—————|
|<base>|—————|
| NEXT | | + —<increment>|

```

<file ID>

```

—<intname>—————|
| " —<title> " |

```

The \$.INSERT option serves two functions. The \$.INSERT option described in syntax 1 inserts a copy of a portion of the virtual TAPE file (the TAPE file plus previous patches) or a portion of an external file (indicated by <file ID>) at the specified base. The \$.INSERT option described in syntax 2 permits text that is being inserted to be patched.

Explanation

<file ID>

Specifies the file to be inserted. If the file ID is not specified, the virtual TAPE file is inserted.

The intname is used if the file was file-equated in the file equation section.

The title specifies an external file name.

<baseinc>

Specifies the sequence number at which the inserted text is to begin. If no increment option is specified, then the last value of the sequence increment is used. Since the base and increment used for the \$.INSERT option are the same as the base and increment used for handling the \$.SEQ \$ option and the \$.MOVE option, their values can be changed during the INSERT operation by a \$ <integer> record or + <increment> option featured in syntax 2. The base and increment are not reset to their default values until the next patch (\$# record).

NEXT

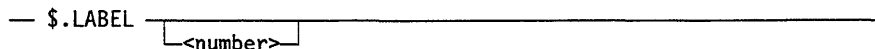
If the value of \$.SEQ is TRUE, then NEXT causes the the present value of the sequence base to be used as the base. If the value of \$.SEQ is FALSE, NEXT causes the sequence number of the last record in this patch plus the value of the increment to be used as the base.

An INSERT operation cannot be done while \$.VOID is TRUE, while \$.MERGE is FALSE, or while a \$.MOVE is being done. \$.VOID cannot be SET, MERGE cannot be RESET, a MOVE operation cannot be done, a \$ GO TO cannot occur, and \$.SEQ cannot be changed during an INSERT operation. \$ records cannot occur in text inserted from an external file. If the INSERT is from an external file, then \$.VOIDT can be SET when the INSERT begins but cannot be changed during the INSERT. If the INSERT is not from an external file, then \$.VOIDT cannot be in a SET state at any time during the INSERT. INSERT options cannot be nested – that is, no INSERT options can appear in the file to be inserted. The range to be inserted cannot overlap the destination range if the INSERT is from the virtual TAPE file. The destination range cannot overlap sequence numbers in the virtual TAPE file.

\$.LABEL Option

(Type: value, Default: FALSE)

<\$.LABEL option>



If the option value is SET or no value is specified, the \$.LABEL option must be followed by an unsigned integer. If the option value is POP or RESET, the \$.LABEL option must not be followed by this unsigned integer. This integer indicates the column on the \$# record where the LABEL information for that patch begins. The LABEL information is terminated by the first blank character. If \$.COBOL or \$.COBOL74 is TRUE, this LABEL information is right-justified in column 80 for a maximum length of 8. If \$.COBOL or \$.COBOL74 is FALSE, the LABEL information is prefaced by a percent character (%) and right-justified in column 72. If a nonblank character is present in the destination field of the record to be labeled, the LABEL information for that record is suppressed.

Example

In the following example, the patch records would be labeled %MYPATCH right-justified in column 72 if \$.COBOL or \$.COBOL74 is FALSE. If \$.COBOL or \$.COBOL74 is TRUE, the patch records would be labeled MYPATCH, right-justified in column 80.

```
$.SET LABEL 3
$#MYPATCH
```

\$.LIST Option

(Type: Boolean, Default: FALSE)

<\$.LIST option>

— \$.LIST —————|

The \$.LIST option tells the PATCH utility to list the created PATCH file—if no fatal errors occurred in the LINE file.

\$.LISTD Option

(Type: Boolean, Default: FALSE)

<\$.LISTD option>

— \$.LISTD —————|

If \$.LISTD or \$.COMPARE is TRUE and \$.LISTP is FALSE, all patch delimiter `$#` records and certain control `$.` records (but not the bodies of the patch) are listed. Each delimiter record is shown with the ordinal patch number assigned to that patch. This number is the number used in the CONFLICT and COMPARE listings to refer to individual input patches. The control records `$.BRIEF` , `$.CYCLE` , `$.FLAG` , and `$.VERSION` are listed if \$.LISTD or \$.COMPARE is TRUE when the control record is processed.

If \$.LISTD and \$.LISTP are TRUE, the entire input is listed.

\$.LISTI Option

(Type: Boolean, Default: TRUE)

<\$.LISTI option>

— \$.LISTI —————|

If \$.LISTI is TRUE, the PATCH utility lists input inserted from external files, as specified by the \$.INSERT option, in the LISTP section of the LINE file.

\$.LISTN Option

(Type: Boolean, Default: FALSE)

<\$.LISTN option>

```
— $.LISTN [ <integer> , <integer> ]
```

If \$.LISTN is TRUE, then specified ranges of the virtual NEWTAPE file are listed in the LINE file. If the specified range is empty, then the complete virtual NEWTAPE file is listed.

If the value of \$.LISTN is FALSE, then specified ranges of the virtual NEWTAPE file are not listed in the LINE file. If the specified range is empty, then none of the virtual NEWTAPE file is listed. You can reset all or parts of ranges that were previously set.

The virtual NEWTAPE file is the NEWTAPE file the PATCH utility creates or the NEWTAPE file the PATCH utility would create if NEW were TRUE.

The default value for the \$.LISTN option is as if \$.LISTN had been specified FALSE with an empty sequence range—that is, no portion of the virtual NEWTAPE file is listed.

\$.LISTP Option

(Type: Boolean, Default: TRUE)

<\$.LISTP option>

```
— $.LISTP _____
```

If \$.LISTP is TRUE, the PATCH utility lists the input to the LINE file. All \$, \$#, \$:, \$&, \$*, \$-, and \$. records are listed in this section as they are found.

\$.MARK Option

(Type: Boolean, Default: FALSE)

<\$.MARK option>

```
— $.MARK _____
```

If \$.MARK is TRUE, the PATCH utility places mark-level information in columns 81 through 90 of the file records. If the file is an ESPOL symbolic file, columns 81 through 88 of the merged patch are marked.

If \$.VERSION is FALSE, the mark number is taken as the first item immediately following the first nonblank character string on each \$# record.

If \$.VERSION is TRUE, the PATCH utility concatenates the version number, cycle number, and patch number to form the mark number. The patch number can be 1 to 4 digits long and is taken as the first item immediately after the first nonblank character string on the \$# record. Refer to “\$.VERSION and \$.CYCLE Options” later in this section for more information about the use of the \$.VERSION and \$.CYCLE options.

The information retained by \$.MARK is not stored into record images that contain a dollar sign (\$) in column 1.

Examples

In both of the following examples, all records from the patch – except records read in from a file specified by a \$.DISK option – contain 33.320.056 in columns 81 through 90 in the merged patch. These records are also labeled %XYZ in columns 69 through 72. The PATCH utility also checks that this patch has exactly 12 records in it. Refer to “\$.LABEL Option” and “\$.COUNT Option” in this section.

```
$.MARK LABEL 5 COUNT 3
$#12XYZ 33.320.056
```

```
$.MARK LABEL 5 COUNT 3
$.VERSION 33.320
$#12XYZ 56
```

\$.MARKBLANK Option

(Type: Boolean, Default: FALSE)

<\$.MARKBLANK option>

— \$. MARKBLANK _____|

The \$.MARKBLANK option is a conditional form of the \$.MARK option. If \$.MARKBLANK is TRUE, and if the mark field (columns 81 through 90 or columns 81 through 88 for ESPOL files) contains all blanks, then the PATCH utility inserts the mark-level information as it does in the \$.MARK option. If the mark field of the input record already contains nonblank data, the field contents are retained. Any record read from a file with a MAXRECSIZE value that is less than the number of columns necessary to completely contain the mark field automatically has a blank mark field. In this case \$.MARKBLANK is equivalent to \$.MARK.

If both \$.MARKBLANK and \$.MARK are TRUE, \$.MARKBLANK is still effective.

\$.MOVE Option

(Type: immediate)

Syntax 1

<\$.MOVE option>

```
— $.MOVE —<first>— - —<last>— TO —<baseinc>—————|
```

Syntax 2

<\$.MOVE option>

```
— $.MOVE —<first>— TO —<baseinc>—————|
```

.
.

.

.

<patch records to the moved material>

```
— $. [ POP ] MOVE —<last>—————|
     [ RESET ]
```

<baseinc>

```
— [ <base> ] + [ <increment> ]—————|
   [ NEXT ]
```

The \$.MOVE option moves portions of the virtual TAPE file – that is, the TAPE file plus previous patches – to a range beginning at the specified base. The PATCH utility places SET and POP VOIDT around the range to be moved and creates a copy of the moved text at the new range. The \$.MOVE option described in syntax 2 permits text that is being moved to be patched.

Explanation

<baseinc>

Specifies the sequence number at which the moved text is to begin. If no increment is specified, then the last value of the sequence increment is used. Since the base and increment used for the \$.MOVE option are the same as the base and increment used for handling the \$.SEQ \$ option and the \$.INSERT option, their values can be changed during the MOVE operation by a \$ integer record or \$ + <increment> option featured in syntax 2. They are not reset to their default values until the next patch (\$# record).

NEXT

If the value of \$.SEQ is TRUE, then NEXT causes the present value of the sequence base to be used as the base. If the value of \$.SEQ is FALSE, NEXT causes the sequence

PATCH Utility

number of the last record in this patch plus the value of the increment to be used as the base.

A MOVE operation cannot be done while VOIDT or VOID is TRUE, while \$.MERGE is FALSE, or while an INSERT operation is being performed. \$.SEQ cannot be changed and a \$ GO TO cannot occur while a MOVE operation is being performed. \$.MOVE options cannot be nested. The range to be moved cannot overlap the destination range, and the destination range cannot overlap sequence numbers in the virtual TAPE file.

\$.NDLII Option

(Type: Boolean, Default: FALSE)

<NDLII option>

— \$.NDLII _____|

The \$.NDLII option tells the PATCH utility to expect input in NDLII format—that is, sequence numbers in columns 73 through 80 and a dollar sign in column 1. If \$.NDLII is TRUE, any \$DELETE or \$VOIDT records produce \$DELETE records. If \$.NDLII is FALSE, \$VOIDT records are produced.

\$.NEW Option

(Type: Boolean, Default: FALSE)

<\$.NEW option>

— \$.NEW _____|

If \$.NEW is TRUE and the PATCH utility finds no fatal errors, the PATCH file is merged with the SOURCE or TAPE file to create the NEWSOURCE or NEWTAPE file. See “Files Used by the PATCH Utility” for information on selection of SOURCE or TAPE. The NEWTAPE file contains no \$ records; even \$ records passed through as \$& records are not included. The blocking factors of the NEWTAPE file are the same as those that a compiler-created NEWTAPE would have.

\$.OUT Option

(Type: Boolean, Default: FALSE)

<\$.OUT option>

— \$.OUT _____|

When \$.OUT is TRUE, \$, \$*, \$#, \$:, \$-, and \$. records with \$.MOVE or \$.INSERT options and regular patch records are written to the output disk file PATCHES. This file is locked after all input has been processed.

\$.PASCAL Option

(Type: Boolean, Default: FALSE)

<\$.PASCAL option>

— \$.PASCAL _____|

The \$.PASCAL option tells the PATCH utility to expect input in the Pascal format, that is, with sequence numbers in columns 73 through 80 and a dollar sign (\$) in column 1. When \$.PASCAL is TRUE, any \$DELETE and \$VOIDT records produce \$DELETE records. When \$.PASCAL is FALSE, \$VOIDT records are produced.

\$.PATCHDECK Option

Refer to “\$.FILE, \$.DISK \$, and \$.PATCHDECK Options” in this section.

\$.RPG Option

(Type: Boolean, Default: FALSE)

<\$.RPG option>

— \$.RPG _____|

The \$.RPG option tells the PATCH utility to expect input in RPG format; that is, with sequence numbers in columns 1 through 5 and a dollar sign (\$) in column 6. When \$.RPG is TRUE, all special \$ records recognized by the PATCH utility must appear in column 6. But the record actually setting the \$.RPG option must have a \$. starting in column 1. If RPG is TRUE, any \$DELETE or \$VOIDT records produce \$DELETE records.

\$.SINGLE Option

(Type: Boolean, Default: TRUE)

<single option>

— SINGLE _____|

When \$.SINGLE is TRUE, the PATCH utility single-spaces the output to the LINE file. When \$.SINGLE is FALSE, this output is double-spaced.

If the PATCH utility was compiled with the compiler user option DOUBLE set to TRUE, the default value is FALSE.

\$.SQUASH Option

(Type: Boolean, Default: TRUE)

<\$.SQUASH option>

— \$.SQUASH _____|

When \$.SQUASH is TRUE, each patch in the LISTP listing is separated by a line of equal signs (=). When \$.SQUASH is FALSE, each patch is listed beginning on a new page.

\$.TOTAL Option

(Type: value, Default: FALSE)

<\$.TOTAL option>

— \$.TOTAL _____|
 └─<number>┘

If the specified option value is SET, or if no value is specified, \$.TOTAL must be followed by an unsigned integer. This integer specifies the total number of patches in the input. When all input has been processed and the value of \$.TOTAL is TRUE, the PATCH utility checks the number of patches actually found against the number specified. If the two numbers do not agree, a fatal error is issued, and the PATCH file is not locked.

\$.VERSION and \$.CYCLE Options

(Type: value, Default: FALSE)

<\$.VERSION Option>

<\$.CYCLE Option>

— \$.VERSION — <version number> _____|
 └─ . —<cycle number> ┘
— \$.RESET VERSION _____|
— \$.CYCLE — <cycle number> _____|

When \$.VERSION and \$.CYCLE are used, the PATCH utility concatenates the version number, cycle number, and patch number, separated by periods, to form the mark number. The PATCH utility does not use the separators in this concatenation if the TAPE file is an ESPOL symbolic.

Both a version number and a cycle number must be specified. A cycle number can be specified with either the \$.VERSION option or the \$.CYCLE option. The version number or cycle number can be changed separately at any time.

The patch number is taken as the first item immediately after the first nonblank character string on the \$# record. The patch number can only be changed at the beginning of each patch by the \$# record.

All three numbers (version number, cycle number, and patch number) must be in the correct range. The version number can be two digits, the cycle number can be three digits, and the patch number can be four digits.

The mark field in the PATCH file record contains the version number, cycle number, and patch number. Periods are used as separators if the patch number is three digits or less, (that is, the mark field is created as *vv.cc.nnn*. If the patch number is four digits, periods are not used as separators—that is, the mark field is created as *vvccnnnn* and column 90 is blank. When the mark field is listed, the PATCH utility inserts periods between the fields for readability purposes.

\$.RESET VERSION restores the default mechanism as if no \$.VERSION or \$.CYCLE had appeared.

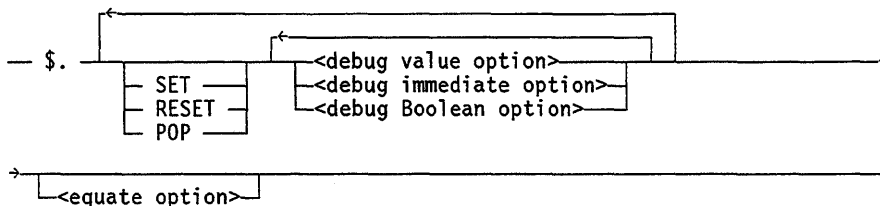
The \$.VERSION and \$.CYCLE options are ignored unless \$.MARK or \$.MARKBLANK is TRUE.

Debug \$. Records

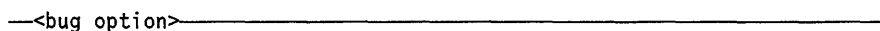
The compile-time option `DEBUG` is available to facilitate the debugging and development of the PATCH utility. Various \$. records exist that can be used only if the PATCH utility is compiled with `$ SET DEBUG`. These additional \$. records allow the flow of control through many critical procedures and the values of many important variables to be traced at will. These options are especially useful when the PATCH utility is run interactively through a remote terminal.

The following syntax diagrams show the \$. options that can be used if the PATCH utility has been compiled with `$ SET DEBUG` in the symbolic. The text following the diagrams give the syntax and an explanation for each option. All these options (except \$.CANDE) can be used in WFL jobs.

<debug option>



<debug value option>



<debug immediate option>

— <discard option> —————|
 └─<end option>┘

<debug Boolean option>

— <pdump option> —————|
 └─<CANDE option>┘

Debug Options

The following paragraphs describe all the debug options in alphabetical order.

\$.BUG Option

(Type: value, Default: FALSE)

<\$.BUG option>

— \$.BUG —————|
 └─<integer> , <integer> —————┘
 └─<integer> - - <integer> ┘

\$.BUG specifies SET, RESET, or POP for the \$.BUG option as indicated by <integer> or all the \$.BUG options indicated by the range <integer>–<integer>. The specific action of each \$.BUG option is subject to change and can be found in the BUG DIRECTORY near the beginning of the symbolic.

\$.CANDE Option

(Type: Boolean, Default: FALSE)

<\$.CANDE option>

— \$.CANDE —————|

This option is valid only when you run the PATCH utility interactively from a remote terminal through CANDE. Changing the value of CANDE has no effect when the PATCH utility is initiated through a WFL job. When \$.CANDE is TRUE, you can enter text that has sequence numbers by typing the sequence numbers first.

Example

The first group of records is equivalent to the second group in the following example:

```

$.CANDE
$# PATCH 005
$.SET COMPARE
500$ SET VOIDT
01000$ POP VOIDT

$#PATCH 005
$.SET COMPARE
$ SET VOIDT           00000500
$ POP VOIDT           00001000
    
```

\$.DISCARD Option

(Type: immediate)

<\$.DISCARD option>

— \$.DISCARD _____|

The \$.DISCARD option causes the PATCH utility to close and purge the LINE printer file so that all printer output up to this point is eliminated.

\$.END Option

(Type: immediate)

<\$.END option>

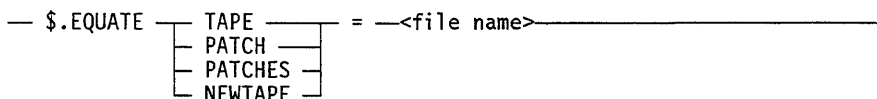
— \$.END _____|

The \$.END option indicates the end of all input to be merged for this particular set of patches. If no errors occur, the PATCH utility then creates the PATCH file and does the COMPARE operation and other optional output functions that have been specified. The utility then starts reading from the primary input file—CARD file or remote file—and expects input for another PATCH utility run. This capability permits multiple patches to be entered in one PATCH utility run. The \$.EOF option is used to indicate that no more patches are to follow.

\$.EQUATE Option

(Type: special)

<\$.EQUATE option>



The \$.EQUATE option causes the PATCH utility to change the name of one of the four files specified to the name given. An ON <family name> clause cannot be specified with the file name. At least one blank must precede the file name.

This option must be the last option on the \$. record. When used with the \$.END option, multiple patches can be entered against different pieces of software with different PATCH, PATCHES, and NEWTAPE files created in one PATCH utility run. Separate printer files are created for each run.

If the \$.EQUATE option is not used, the PATCH utility looks for a file named TAPE and creates files named PATCH, PATCHES, and NEWTAPE.

SET and RESET context are ignored for the \$.EQUATE option.

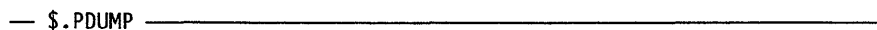
Example

```
$.EQUATE TAPE = MYFILE  
$.EQUATE PATCH = MYFILE/PATCH
```

\$.PDUMP Option

(Type: Boolean, Default: FALSE)

<\$.PDUMP option>



If \$.PDUMP is TRUE, the PATCH utility performs a program dump when any error is encountered.

Examples of PATCH Utility Input

This section shows two examples of running the PATCH utility. The first example uses a WFL job. The second example uses CANDE to initiate an interactive session.

The following example shows a WFL job initiating the PATCH utility. This example declares two internal files, INCL1 and INCL2, which are used in the \$.INSERT options.

```

BEGIN JOB PATCH;
  RUN SYSTEM/PATCH;
  FILE TAPE(TITLE=SYMBOL/PATCH ON SYSPACK);
  FILE PATCH(TITLE= PATCH/NEWPATCH);
  FILE PATCHES(TITLE= PATCH/NEWPATCHES);
  FILE INCL1 (TITLE=(USCODE)INCLUDE/FILE/1);
  FILE INCL2 (TITLE=(USCODE)INCLUDE/FILE/2);
  FILE NEWTAPE(TITLE=SYMBOL/NEW/PATCH ON SYSPACK);
DATA CARD

$.SET NEW COMPARE BRIEF EXECUTE LIST
$:THE FOLLOWING $* COMMANDS CAUSE MY/PROGRAM ON MYPACK TO BE RUN
$:USING KARD/FILE AS THE INPUT CARD FILE.
$*RUN MY/PROGRAM ON MYPACK
$*FILE CARD(KIND=DISK,TITLE=KARD/FILE)

$# D O L L A R   C A R D S
$ SET MERGE
$ SET NEW
$ SET LISTP
$ SET LINEINFO
$ SET SEQERR NEWSEQERR
$:THESE ARE SOME STANDARD $ CARDS FOR A COMPILE
$.MARK TOTAL 5  OUT DELETE 5

$# FIRST_PATCH  31.099.001
$.PATCHDECK MY/PATCH/FILE ON MYPACK

$# NEXT_PATCH  31.099.002
  :
  :
  <patch records>
  :
  :
$.VERSION 32.020

$# COMMENT  001
$.MOVE 50000-52000 TO 600100+100
$.MOVE 800000 TO NEXT+20
  :
  :
  <patches to the moved material>
  :
  :
```

PATCH Utility

```
$.POP MOVE 8001000
$.INSERT INCL1 0-500 AT 9000000 + 30
$.INSERT INCL2 6000-7000 AT NEXT
$.INSERT "MY/THIRD/INCL/FILE ON MYPACK" 61000 AT NEXT+300
:
:
<patches to the inserted material>
:
:
$.RESET INSERT 75000
$.INSERT 2100-2500 AT NEXT % THIS IS A COMMENT

$# COMMENT 2
$:THIS IS A COMMENT ABOUT THIS PATCH
$.FILE A/B
$.POP MARK RESET LISTP
$.DISK $ MY/OTHER/A/B ON MYPACK2
$.EOF
$:THIS CARD WILL NOT BE READ BY THE PATCH UTILITY
$:NOR WILL THIS ONE
?
END JOB.
```

The following example shows an interactive run of the PATCH utility. Lines that are preceded by a right angle bracket (>) indicate a system response. In this example, two patches are entered to show the use of the \$.END option. After the \$.END option, the PATCH utility displays the number of errors found, the number of warnings found, and the number of conflicts encountered. The word ENCOUNTERED is truncated to E. It then expects input for another patch. The \$.EOF option is used to indicate the end of the run.

```
RUN *SYSTEM/PATCH

> RUNNING <mix number>
> ?
>ENTER INPUTS--

$:START OF FIRST INTERACTIVE PATCH RUN
$.EQUATE TAPE = SYMBOL/SOURCE
$#PATCH SEPARATOR CARD 1
$ SET LIST MERGE NEW
$# PATCH SEPARATOR CARD 2
$.FILE PATCH/SOURCE/23
$:END OF FIRST PATCH RUN
$.END

><mix number> DISPLAY:OK TO COMPILE
> 0 ERROR(S) WERE FOUND 0 WARNING(S) WERE ISSUED 0 CONFLICTS WERE E

$:START OF 2ND PATCH RUN
$.EQUATE TAPE = MY/SYMBOL
$.EQUATE PATCH = OUT/PATCH
```

```
$.EQUATE NEWTAPE = OUT/NEWTAPE
$.CANDE
$# PATCH 005
$.SET COMPARE
500$SET VOIDT
01000$ POP VOIDT
$# PATCH 006
$.DISK MY/PATCH
$:END OF RUN
$.EOF
```

```
><mix number> DISPLAY:OK TO COMPILE
> 0 ERROR(S) WERE FOUND 0 WARNING(S) WERE ISSUED 0 CONFLICTS WERE E
> ET=6:39.4 PT=2.4 IO=6.1
```


Section 10

PL/I Indexed Sequential-Access Method (PLIISAM)

This section describes a set of software routines that implement indexed-sequential access methods of storage and retrieval of data records. Indexed sequential access method (ISAM) permits a keyed file to be processed in both random and serial fashion. This section is intended for use as a reference document for experienced programmers.

This section explains ISAM files that have a FILEORGANIZATION value equal to PLIISAM. For information about ISAM files that have a FILEORGANIZATION value equal to INDEXED or INDEXEDNOTRESTRICTED, refer to Section 7, “KEYEDIO Support.” For information about ISAM files that have a FILEORGANIZATION equal to KEYEDIOII or KEYEDIOIISSET, refer to the *KEYEDIOII Reference Manual*.

This section contains some information that applies to both standard and primitive ISAM and other information that applies only to primitive ISAM. A discussion of the differences between standard and primitive ISAM is given directly following this introduction. Most of the material concerning standard ISAM, including lists of exception codes, is contained in the *A Series PL/I Reference Manual* and in the *A Series COBOL ANSI-74 Programming Reference Manual Volume 2: Product Interfaces*.

You can access the ISAM facility by using only the COBOL (with or without the \$ANSI74 option), PL/I, and ALGOL compilers. The COBOL74 and RPG compilers do not use ISAM; they use KEYEDIO. Capabilities similar to ISAM are also available through Data Management System II (DMSII). For further information, refer to the *A Series DMSII Data and Structure Definition Language (DASDL) Programming Reference Manual* and the DMSII User Language Interface Programming Guide.

The support procedures for ISAM are contained in the SYSTEM/PLISUPPORT library. All documentation notes pertaining to ISAM appear under the heading PLISUPPORT. To initiate PLISUPPORT, use the SL (Support Library) system command. Refer to the *A Series System Commands Operations Reference Manual*.

Program Interface for Primitive and Standard ISAM

The following paragraphs define the two ways of invoking ISAM, the primitive method and the standard method.

Primitive ISAM

A set of ISAM procedures for creating and updating ISAM files is listed later in this section under “Implementation Information for Primitive ISAM Procedures” and “ISAM Procedures.” Primitive ISAM involves direct calls of the procedures that

perform the ISAM functions rather than use of special language syntax. Parameters must be passed explicitly. Each procedure returns a value indicating its results. In primitive ISAM, the program must detect exception conditions by interpreting the result word whose contents are explained at the end of this section. ISAM exception conditions do not cause program termination when primitive ISAM is used. Primitive ISAM permits the highest amount of selection and control. ALGOL must use primitive ISAM. COBOL can select either primitive or standard ISAM. PL/I must use standard ISAM.

Standard ISAM

The standard interface simplifies programming effort by permitting you to use normal, higher level language input/output (I/O) statements such as READ and WRITE rather than requiring you to directly invoke the ISAM procedures. No loss of efficiency results when standard ISAM is used. PL/I must use the standard method. COBOL can select either the standard or primitive method. PL/I and COBOL each have unique implementation features.

The standard interface includes the use of ISAM file options. Functionally, ISAM file options are similar to file attributes but they exist only for ISAM files. Unlike file attributes, ISAM file options cannot be assigned a value or be modified by control cards or programmatic file attribute statements. Each ISAM file option is assigned a value when the file is created and opened as OUTPUT. Examples of COBOL ISAM file options include the settings for KEYSPERENTRY, AREAOVERFLOW, and FILEOVERFLOW. Some keyed file options for PL/I include KEYLENGTH, KEYORDER, FILEOVERFLOW, and WAITUPDATEIO.

Standard ISAM deals with exception conditions differently from primitive ISAM. Primitive ISAM procedures return an information word to reflect the results of the ISAM invocation. In standard ISAM, the compiler emits code for observing the results and initiating appropriate action. ISAM exception conditions can occasionally cause program termination in standard ISAM. Tables listing the exception codes for standard ISAM are given in the *A Series COBOL ANSI-74 Programming Reference Manual Volume 2: Product Interfaces* and the *A Series PL/I Reference Manual*.

Structure of ISAM Files

An ISAM file consists of the following logical sections within one physical file:

- The prime data area
- The prime data area overflow space
- The data overflow area
- The file overflow area

These sections are defined in the following paragraphs.

Prime Data Area

The prime data area holds all the keyed data records that are entered when the file is first created. You can determine the maximum size of this area (in records) by multiplying the values of two file attributes: AREAS and AREASIZE. The number of prime data area rows is specified by the value of the AREAS attribute. The number of records in each prime data area row is specified by the value of the AREASIZE attribute. File space is automatically reserved by the ISAM program for nondata purposes (coarse and fine tables); the amount reserved is determined by the values of the attributes AREAS and AREASIZE.

ISAM files do not assume the default values of a normal file for AREAS and AREASIZE because these values should be carefully chosen for optimum performance. The values associated with AREAS and AREASIZE must be specified when creating (and opening as OUTPUT) an ISAM file, but they do not need to be specified at any other time.

When an ISAM file is created, all unused space contained in the final row of the prime data area is incorporated into overflow space for the final row, and totally unused prime data area rows are incorporated into the file overflow area.

Data Overflow Area

The data overflow area of the file is the unoccupied data area. Records added after file creation are always placed in an overflow area. Two types of physical overflow areas are provided: the prime data area overflow space and the file overflow area.

Prime Data Area Overflow Space

Area overflow space can be provided in each row containing prime data and is specified when the file is created (it is opened as OUTPUT at file creation time.) In standard ISAM, row overflow space is indicated by a file option such as AREA_OVERFLOW in COBOL. In primitive ISAM, row overflow space is indicated through a parameter to the ISOPEN procedure. When records are deleted, the occupied space can be returned to the overflow pool in the prime data area row in which the record resides. The deleted record option, set to ON when the file is created and opened as OUTPUT, specifies the disposition of the space occupied by deleted records.

File Overflow Area

A file overflow area outside of the prime data area can also be specified when the file is created and opened as OUTPUT. Again, a file option in standard ISAM or the ISOPEN procedure in primitive ISAM indicates the size of the overflow area. Records are placed in the file overflow area only after all available overflow space in the specific prime data area row in which the record would normally reside has been filled.

Tables for Locating Data

Two levels of tables are used by the ISAM procedures: fine tables and coarse tables. Each prime data area row of the file contains a fine table. The fine table is a list of keys

and associated file addresses. One key (and address) is placed in the table for each n records, where n is a program-selected value when the ISAM file is first created. The fine table is stored at the physical end of its corresponding file area.

The entire file has one coarse table that contains pairs of keys and addresses. Each key entry is identical to the first key entry of the corresponding fine table, and the address entry is the address of the fine table rather than the address of a data record. The coarse table is stored at the physical beginning of the file overflow area. Therefore, an ISAM file always has at least one physical row of file overflow space.

Data Record Links

ISAM data records are linked together in a logical sequence. Each record automatically contains both a forward and backward link to its logical successor and predecessor. A link is an address of a data record. The first data record contains a backward link that is zero (0), and the last data record contains a forward link that is zero (0). Forward links are used to locate data records. Both forward and backward links are used to insert and delete data records. Data record links are the innermost level of file structure in an ISAM file.

The coarse table serves to locate a fine table; the fine table, in turn, locates a data record. ISAM uses the data record links in following the trail to the desired record when necessary. Data records are not physically moved to accommodate additions and deletions. Instead, the data record links are modified, so that file changes are handled in a logical rather than physical fashion. Because links are physically contained in every data record, ISOPEN must increase the record size to provide space for the links. Increasing record size is accomplished by rounding the original record size up to the nearest full word and then adding one more word to contain the links.

ISAM Management of Overflow Areas

When an ISAM file is created, unoccupied space can be reserved in each prime data area row, and at least one entire row can be reserved for overflow records. The fine table that corresponds to a row also contains information that provides a link to the next available unoccupied space. Overflow space that is reserved when an ISAM file is created is allocated in serial fashion. If deleted record space is subsequently made available for reuse (an option selected by the program), the deleted records are linked into the available record chain for the corresponding area and are reassigned on a last-in, first-out (LIFO) basis. Record space made available for reassignment is reused before unused space is assigned.

The coarse table contains the link to the next available space in the file overflow area. Space assignment in the file overflow area is the same as overflow assignment in a prime data row. New records are not placed in the file overflow area if they can be placed in the prime data area. A given record is never eligible for placement in more than one prime data area row, and the only alternative placement for it is in the file overflow area.

Planning for ISAM Files

ISAM provides a specific set of capabilities that you must consider during preparation for application programs and systems. You can make trade-offs to favor a particular course of action. All features of the ISAM procedures are not available to every mode of operation and every language. The following paragraphs discuss the factors that you should consider when planning for ISAM files.

Maximum Number of Records

The maximum number of records that can be contained in a single ISAM file is 16,777,215. Some of this space is required for a coarse table, fine tables, and an INFO record. Data records can occupy the remaining space.

Coarse Table Size

One coarse table is created for the entire file.

Coarse table size is determined by the number of prime data area rows used during file creation; however, the number of rows used cannot exceed the value of the AREAS attribute because the coarse table cannot expand. Not more than 999 prime data area rows can be requested because at least 1 row is required for file overflow. The default value of AREAS is 1. One entry is made in the coarse table for each prime data area row. The table is contracted when fewer prime data area rows are used than specified. Key length also directly affects table size.

If the coarse table size exceeds 393,210 bytes, an error message is issued. The following explains how to compute the number of bytes needed for a coarse table.

Computing Coarse Table Size

Use the following formula to compute the coarse table size. All units are bytes (8-bit characters).

Coarse table size =

$$\begin{aligned} &(\text{number of table entries} * (\text{key length} + 3) \\ &+ 24 + \text{BLOCKSIZE} - 1) \text{ DIV } \text{BLOCKSIZE} * \text{BLOCKSIZE}. \end{aligned}$$

Record space loss to coarse table =

$$\text{coarse table size DIV } \text{BLOCKSIZE} * \text{number of records per block}.$$

Fine Table Size

One fine table is created for each prime data area row. A prime data area is a row of the ISAM file that was written into while the file was being created. All other rows of the

PL/I Indexed Sequential-Access Method (PLIISAM)

file are allocated to file overflow and do not contain fine tables. In any ISAM file, all fine tables have identical size.

A fine table ratio is specified to determine the number of entries in each fine table. The ratio can range between 1 and 63; the default value is 1. When duplicate records are permitted, only the first record in the duplicate set is eligible for entry in the fine table or is counted in meeting the fine table ratio. Key length also directly affects fine table size. Space for the fine table is automatically allocated for the fine table by the ISAM program.

If the fine table size exceeds 393,210 bytes, an error message is given. The following explains how to compute the number of bytes needed for fine table size.

Computing Fine Table Size

All units are bytes (8-bit characters).

Fine table size =

$$\begin{aligned} & (\text{number of table entries} * (\text{key length} + 3) \\ & + 24 + \text{BLOCKSIZE} - 1) \text{ DIV } \text{BLOCKSIZE} * \text{BLOCKSIZE}. \end{aligned}$$

Record space loss to fine table =

$$\begin{aligned} & \text{fine table size DIV } \text{BLOCKSIZE} * \text{number of records per block} \\ & * \text{number of fine tables}. \end{aligned}$$

INFO Record Size

The first record of each ISAM file is a special record that contains attribute information about that particular ISAM file. This INFO record is essential for proper access of the data records of the ISAM file. The current length of the INFO record is 7 words. One data record space is normally required to contain the INFO record, but more can be used if the data record length is less than 7 words (42 bytes).

AREAS and AREASIZE Values

The file attributes AREAS and AREASIZE are more important to ISAM files than to non-ISAM files. You must specify both attributes when you are creating a new ISAM file, but these attributes are not needed at other times. The default value is 1 for both attributes. The value of AREAS indicates the number of prime data area rows expected for the ISAM file; the value of AREAS cannot exceed 999. When an ISAM file is first created, a few more areas than needed should be specified. The value of AREASIZE, as specified by the program, indicates the number of data records per prime data area row.

The value of the AREASIZE attribute is automatically increased by the ISAM program to allow the fine table to be written in the same area (or row) of the file as the data it represents. The value of the AREASIZE attribute is also increased by the number of overflow records per area that is specified by a file option in standard ISAM or by an

ISOPEN procedure in primitive ISAM. Similarly, the value of the AREAS attribute is increased by the number of file overflow areas specified by a file option in standard ISAM or by an ISOPEN procedure in primitive ISAM. This increase is very similar to allowing the file to expand via the FLEXIBLE attribute; the increase does not affect the size of the coarse table. When a given ISAM file is closed, the AREAS and AREASIZE attributes are reset to their original values.

Minimum Record Size (MINRECSIZE)

ISAM does not provide for variable-length records. Therefore, the value of the MINRECSIZE attribute should be 0 or identical to the value of the MAXRECSIZE attribute.

Maximum Record Size (MAXRECSIZE)

The value chosen for the MAXRECSIZE attribute is entirely dependent upon the needs of the program and the absolute limits allowed by the system. ISAM increases the value of the MAXRECSIZE attribute by at least 1 word (6 bytes) and at most 11 bytes. The program should not assign a new value to the MAXRECSIZE attribute of a given ISAM file except when the file is first created and opened. When the ISAM file is closed, the MAXRECSIZE attribute is reset to its original value. The maximum usable values for MAXRECSIZE are 65,534 words or 65,535 characters.

BLOCKSIZE Attribute

The BLOCKSIZE attribute is used in association with the MAXRECSIZE attribute to determine the number of records per block. The value of BLOCKSIZE is always changed by ISAM. When the program specifies a nonzero value for BLOCKSIZE, ISAM retains the number of records per block specified by the program. The value of BLOCKSIZE is increased to accommodate larger records. When the program specifies a BLOCKSIZE of zero (0), ISAM computes a new value for BLOCKSIZE in order to conserve disk storage space. After the value MAXRECSIZE has been increased, ISAM computes the smallest number of records that exactly fits into a multiple of 30-word disk segments. The BLOCKSIZE attribute is reset to its original value when the file is closed.

EXCLUSIVE USE Attribute

The EXCLUSIVE USE attribute is a Boolean attribute that is set to TRUE by ISAM when an ISAM file is opened as INPUT-OUTPUT. Programs cannot share ISAM files unless all programs open the file as INPUT only.

Fine Table Ratio

You can select any value from 1 through 63 for the fine table ratio; a value of 1 signifies a table entry for each record, and a value of 63 signifies a table entry for each 63 records. The most successful choice is highly data- and program-dependent. A small

PL/I Indexed Sequential-Access Method (PLIISAM)

value is generally appropriate when records are often accessed randomly rather than sequentially. In order to improve or restore performance, you can reorganize the file after a number of changes have occurred.

Key Length

Some key modes permit specific maximum key lengths of 5, 6, or 11 bytes. Refer to the mode of key description under "ISOPEN Procedure" for a full list of key modes. Character keys of 4-bit or 8-bit characters are limited only by the 14-bit field that contains key length. The maximum usable key lengths are 8-bit characters for 1020 bytes and 4-bit characters for 508 bytes (1016 hexadecimal characters). Shorter keys yield faster performance and smaller fine and coarse tables.

Key Offset

ISAM provides a 15-bit field that permits an offset of 32767 for 8-bit characters and 16382 for 4-bit characters.

Practical Considerations

In an unstable environment, ISAM files can become corrupted—a condition in which the coarse table, fine tables, or data record links do not concur. Corruption of ISAM files can occur when the physical file on disk has not yet been updated to reflect the changes that have been made to buffers in memory. If an event occurs that prevents the updated buffers from being written into the physical file, the file can become corrupted. Use of standard ISAM helps to prevent this situation.

In those cases where the program terminates prematurely because of an invalid index, a divide-by-zero error, a termination, or some other reason, standard ISAM closes the ISAM file in an orderly manner, while primitive ISAM might not be able to close the file properly. However, the possibility of file corruption exists only after the file has been opened as INPUT-OUTPUT and additions or deletions have occurred. File damage is by no means inevitable, and two file options, WAITUPDATEIO and PHYSICALUPDATE, are available to further reduce such a possibility. Refer to the descriptions of the WAITUPDATEIO and PHYSICALUPDATE options discussed under "ISOPEN procedure."

ISAM files cannot be specified as output files to the SORT utility, except in PL/I. They must be read and written by INPUT or OUTPUT procedures. Other system software might also encounter similar situations when attempting to process ISAM files in a direct fashion without use of the ISAM procedures.

ISAM files cannot be implicitly opened using the PRESENT or AVAILABLE file attributes. ISAM files must be explicitly opened using the OPEN verb in standard ISAM or the ISOPEN verb in primitive ISAM.

ISAM files can be accessed simultaneously by several programs, if they all open the file as INPUT. Only one program can access the file while it is open as OUTPUT or INPUT-OUTPUT.

Because direct I/O is used by ISAM procedures to access the data, the ISAM file must be a direct file. In primitive ISAM, the program must declare the file as direct. The compiler properly declares the file in the standard method. The direct arrays used by the ISAM procedures are created by ISOPEN and returned by ISCLOSE. The program does not need other direct arrays to access the data.

ISAM files cannot be used in an Interprogram Communication (IPC) environment in which the file is passed from one task to another.

Implementation for Primitive ISAM Procedures

ISAM is implemented by a set of procedures in the PLISUPPORT support library. Symbolics for these procedures are contained in the PLISUPPORT symbol file. The procedures called directly from programs are as follows:

- ISOPEN – Open and set up file.
- ISCLOSE – Close file.
- ISREAD – Randomly read a record.
- ISWRITE – Add a record to the file.
- ISREADNEXT – Read the next sequential record.
- ISREWRITE – Rewrite the record just read.
- ISKEYWRITE – Randomly rewrite a record.
- ISDELETE – Delete a record.

You must use these ISAM procedures to open, close, create, and access ISAM files. You cannot use these procedures to access non-ISAM files. Non-ISAM file OPEN, CLOSE, READ, and WRITE statements and accesses by means of file attributes are not disallowed; however, the use of any of them can cause unexpected results that might be detrimental to the integrity of the ISAM file. The ISCLOSE procedure should be used to close the ISAM file. Using an implicit CLOSE statement on an ISAM file to exit a block does not properly save the file.

A \$SET INSTALLATION 1 record must appear at the beginning of the symbol file in all ALGOL programs that invoke ISAM procedures.

ISAM Procedures

The following procedures comprise ISAM. Each procedure is defined in terms of its function or functions within the general ISAM operating method and in terms of its interaction, if any, with other ISAM procedures. The standard program interface does not require direct use of these procedures and their parameters. However, the primitive program interface uses these procedures directly.

ISOPEN Procedure

The ISOPEN procedure opens an ISAM file for INPUT, OUTPUT, or INPUT-OUTPUT. ISAM files require additional information not provided for non-ISAM files. The ISOPEN procedure uses and creates the additional information according to the method of file opening. Non-ISAM files cannot be opened by this procedure.

Use one of the following program-calling sequences to open an ISAM file:

```
ALGOL: RS := ISOPEN(FILE,VALUE,STACK);
```

```
COBOL (PRIMITIVE): COMPUTE RS = ISOPEN (FILE, VALUE, STACK).
```

The following parameters are used in the ISAM calling sequence for ISOPEN:

- **RS**
The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
- **FILE**
The ISAM file being opened. FILE must be declared as a DIRECT FILE in ALGOL and COBOL.
- **VALUE**
A numeric value that specifies how the ISAM file is to be opened:
 - Open as INPUT.
 - Open as OUTPUT.
 - Open as INPUT-OUTPUT.

INPUT and INPUT-OUTPUT require an existing ISAM file. OUTPUT always means creation of a new file.

OUTPUT requires specification of additional file information. High-order bits in the VALUE parameter are used to convey certain information necessary for file creation. Bits and fields contained in this parameter are as follows:

Field	Description
47:1	Separate key (PL/I only).
46:15	Offset of the key, in bytes, from the start of the record. It is the TRUE (zero-relative) offset. A value of zero means the start of the record.
31:2	Open action (open as INPUT or as INPUT-OUTPUT only). 0 - Open the file. 1 - Use PRESENT attribute to open. 2 - Use AVAILABLE attribute. 3 - Not used.
29:14	Actual key length in bytes.

continued

PL/I Indexed Sequential-Access Method (PLIISAM)

continued

Field	Description
15:4	Mode of key. Values are as follows: 0 - BINARY (6-byte maximum) 1 - 8-bit character 2 - 8-bit unsigned numeric (11 bytes maximum) 3 - 8-bit MSD signed numeric (11 bytes maximum) 4 - 8-bit LSD signed numeric (11 bytes maximum) 5 - 4-bit characters 6 - 4-bit unsigned numeric (5 bytes maximum) 7 - 4-bit MSD signed numeric (6 bytes maximum) 8 - 4-bit LSD signed numeric (6 bytes maximum)
11:1	Duplicate key option. If this field is zero (0), records with duplicate keys cannot be added to the file. If this field is 1, duplicates are chained in first-in, first-out (FIFO) sequence. A duplicate key condition exists when the keys in two records are equal.
10:1	Deleted record option. If this field is zero (0), deleted records are physically delinked and their record space becomes available for reuse. If this field is 1, deleted records are flagged by having 4"FF" (all bits ON) placed in the first byte of the record. Records marked as deleted can be retrieved using READNEXT if bit 2 of this parameter word equals 1.
9:1	Sequence option. If this field is zero (0), the file is in ascending sequence. If this field is 1, the file is in descending sequence.
8:6	Fine table ratio. During file creation, this field controls the number of entries made in the fine tables and specifies the number of unique records to be added to the file between fine table entries.
2:1	See deleted record option. If this field is zero (0), deleted records are not visible to the program. If this field is 1, deleted records can be visible to the program if the deleted record option is set to 1 and the READNEXT attribute is used.
1:2	Open type (previously described). 0 - invalid 1 - INPUT 2 - OUTPUT 3 - INPUT-OUTPUT

PL/I Indexed Sequential-Access Method (PLIISAM)

- **STACK**

Specifies the first of four consecutive words in the programs stack. The location of the first word is used by ISOPEN to build data descriptors in all four words. The location is retained in the file information block (FIB) for use as long as the file remains open. The program must provide the space by declaring the four consecutive stack locations preferably with four type REAL variables in ALGOL or four usage COMP-1 variables in COBOL. The four words are not usable by the program while the ISAM file is open. A program reference to any of the four words during the time the file is open causes an "INVALID OPERATOR" message.

Additional file information is conveyed to ISOPEN by use of the first of the four consecutive words.

Field	Description
47:24	Number of overflow records per prime record area row. This field is used only when the file is opened as OUTPUT. At file creation, this field is used to increase the value of the AREASIZE attribute specified for the file. The new, larger value of AREASIZE becomes a permanent attribute of the file. Unoccupied space, large enough to contain the number of records specified by this field, is allocated in each row of the file.
23:1	Wait update I/O field. If 1, this field causes ISAM procedures to wait for I/O completion of all outstanding I/Os before returning to the program. This field cannot be set to 1 if the ANSI74 option is set in COBOL.
22:1	Physical update I/O field. If 1, this field causes the ISAM procedures to initiate I/Os for all buffers and tables that have been modified and need to be rewritten. This field cannot be set to 1 if the ANSI74 option is set in COBOL.
21:6	Unused at present but reserved for future implementation.
15:16	Number of file overflow area rows. This field is used only when the file is opened as OUTPUT. At file creation, this field is used to increase the value of the AREAS attribute specified for the file. The new, larger area becomes a permanent attribute. Any areas represented by this field are not used to contain prime data. Prime data area rows unused at file creation are, however, placed in the file overflow area pool. Therefore, when in doubt, it is better to make the AREAS attribute larger.

ISCLOSE Procedure

The ISCLOSE procedure closes an ISAM file in an orderly fashion. The normal CLOSE statement is not sufficient to close an ISAM file properly. Certain additional file information is saved within the file by the ISCLOSE procedure, and the four consecutive stack words are cleared or restored. Non-ISAM files cannot be closed by this procedure.

Use the following program calling sequences to close an ISAM file:

```
ALGOL: RS := ISCLOSE (FILE, TYPE);
```

```
COBOL (PRIMITIVE): COMPUTE RS = ISCLOSE (FILE, TYPE).
```

- **RS**
The result word returned to the program. RS is type **BOOLEAN** in ALGOL and **COMPUTATIONAL** in COBOL.
- **FILE**
The ISAM file to be closed.
- **TYPE**
A numeric value that specifies how the ISAM file is to be closed:
 - Close the file and release it from the program. This numeric value indicates a normal close. The file does not remain on disk unless it has been previously locked.
 - Close the file with a lock. The file is entered into the directory and remains on disk. Any previous file with a duplicate name can be removed.
 - Close the file and purge its entry from the directory. Any disk space occupied by the file becomes available for reassignment by the system.

ISREAD Procedure

The ISREAD procedure reads a record in a random fashion using the program-supplied key. If the program-supplied key matches a record in the ISAM file, the matching record is returned. When no matching record exists, the next logically sequential record is returned.

The ISAM file must be opened as **INPUT** or **INPUT-OUTPUT** by the ISOPEN procedure before the ISREAD procedure can be used to read records. The ISREAD procedure cannot be used to read non-ISAM files.

Use the following program calling sequences to read an ISAM file:

```
ALGOL: RS := ISREAD (FILE, KEY, AREA);
```

```
COBOL (PRIMITIVE): COMPUTE RS = ISREAD (FILE, KEY, AREA).
```

- **RS**
The result word returned to the program. RS is type **BOOLEAN** in ALGOL and **COMPUTATIONAL** in COBOL.
- **FILE**
The ISAM file.
- **KEY**
The key identifying the record to be read. For ALGOL, KEY must be a pointer. For COBOL, KEY must be a data item.

PL/I Indexed Sequential-Access Method (PLIISAM)

- AREA

The area to contain the record to be read. The AREA must be at least as large as the record. For ALGOL, AREA must be a pointer. For COBOL, AREA must be a data item.

ISWRITE Procedure

The ISWRITE procedure writes a record, using the provided key, from the provided area. This procedure never overwrites or rewrites previously existing records but always adds (or attempts to add) records to the file.

When the ISAM file is opened as OUTPUT, a new file is created. The ISWRITE procedure is used to create coarse and fine tables in addition to placing records into the ISAM file. Records must be presented in the sequence specified by the program when the ISAM file is created. Duplicate record acceptance depends on the setting of the duplicate key option.

When the ISAM file is opened as INPUT-OUTPUT, a previously existing file is utilized. Records need not be presented in any special sequence. The records are written into area overflow or file overflow space and appropriately linked into the ISAM file.

The file must be opened as OUTPUT or INPUT-OUTPUT by the ISOPEN procedure before the ISWRITE procedure can be used. The ISWRITE procedure cannot be used for non-ISAM files.

Use the following program calling sequences to write to an ISAM file:

```
ALGOL: RS := ISWRITE (FILE, KEY, AREA);
```

```
COBOL (PRIMITIVE): COMPUTE RS = ISWRITE (FILE, KEY, AREA).
```

- RS

The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.

- FILE

The ISAM file.

- KEY

The key identifying the record. The value associated with KEY must match the value in the key location in the record. For ALGOL, KEY must be a pointer. For COBOL, KEY must be a data item.

- AREA

The record to be written. The area must be at least as large as the record. For ALGOL, AREA must be a pointer. For COBOL, AREA must be a data item.

ISREADNEXT Procedure

The ISREADNEXT procedure reads the next logically sequential record. The record returned to the program is the record whose key immediately follows in sequence after the most recent record obtained by ISREAD or ISREADNEXT.

The ISAM file must be opened as INPUT or as INPUT-OUTPUT by the ISOPEN procedure before the ISREADNEXT procedure can be used. Non-ISAM files cannot be accessed by the ISREADNEXT procedure.

The purpose of the ISREADNEXT procedure is to provide a sequential processing capability. When ISREADNEXT is used in combination with ISREAD and ISREWRITE, records can be sequentially processed and updated for all or part of any ISAM file. ISREADNEXT can be used to read an entire ISAM file in a sequential manner.

Use the following program calling sequences for the ISREADNEXT procedure:

```
ALGOL: RS := ISREADNEXT(FILE, AREA);
```

```
COBOL (PRIMITIVE): COMPUTE RS = ISREADNEXT (FILE, AREA).
```

- RS

The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.

- FILE

The ISAM file.

- AREA

The area to contain the record to be read. The area must be at least as large as the record. For ALGOL, AREA must be a pointer. For COBOL, AREA must be a data item.

ISREWRITE Procedure

The ISREWRITE procedure replaces the record previously read with the data currently in the record area. This procedure permits records to be updated. Either the ISREAD or ISREADNEXT procedure must immediately precede the ISREWRITE procedure. The key contained in the record to be rewritten must match the key in the record that was read by the immediately preceding file operation.

The file must be an ISAM file and must be opened as INPUT-OUTPUT. Additional records cannot be added to the file by the ISREWRITE procedure.

PL/I Indexed Sequential-Access Method (PLIISAM)

Use the following program calling sequences for the ISREWRITE procedure:

```
ALGOL: RS := ISREWRITE(FILE, AREA);
```

```
COBOL (PRIMITIVE): COMPUTE RS = ISREWRITE (FILE, AREA).
```

- RS
The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
- FILE
The ISAM file.
- AREA
The record to be written. The area must be at least as large as the record. For ALGOL, AREA must be a pointer. For COBOL, AREA must be a data item.

ISKEYWRITE Procedure

The ISKEYWRITE procedure provides a random access update capability for ISAM files. It replaces a currently existing record from the file with the record provided by the program.

The file must be an ISAM file and must be opened as INPUT-OUTPUT. The ISKEYWRITE procedure provides update capability and does not add additional records to the file.

Use the following calling sequences for the ISKEYWRITE procedure:

```
ALGOL: RS := ISKEYWRITE(FILE, KEY, AREA);
```

```
COBOL (PRIMITIVE): COMPUTE RS = ISKEYWRITE (FILE, KEY, AREA).
```

- RS
The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
- FILE
The ISAM file.
- KEY
The key identifying the record to be replaced. The value associated with KEY must match the value in the key location in the record passed in the AREA parameter. For ALGOL, KEY must be a pointer. For COBOL, KEY must be a data item.
- AREA
The record to be written. The area must be at least as large as the record. For ALGOL, AREA must be a pointer. For COBOL, AREA must be a data item.

ISDELETE Procedure

The ISDELETE procedure drops or deletes records from the file. When duplicate records are allowed, the first (the oldest) record is deleted. This procedure provides random access delete capability.

The file must be an ISAM file and must be opened as INPUT-OUTPUT. The ISDELETE procedure deletes the first (that is, oldest) record with a matching key. The record can be physically or logically deleted depending on the DELETED RECORD option.

Use the following calling sequences for the ISDELETE procedure.

```
ALGOL: RS := ISDELETE(FILE, KEY);
```

```
COBOL (PRIMITIVE): COMPUTE RS = ISDELETE (FILE, KEY).
```

- **RS**
The result word returned to the program. RS is type BOOLEAN in ALGOL and COMPUTATIONAL in COBOL.
- **FILE**
The ISAM file.
- **KEY**
The key identifying the record to be deleted. For ALGOL, KEY must be a pointer. For COBOL, KEY must be a data item.

ISAM I/O Result Information

The ISAM procedures return result values to the calling program. These result values indicate success or failure of the program request. Each value returned is a 48-bit word. In primitive ISAM, the result word is type BOOLEAN in ALGOL and COMP-1 or COMP in COBOL. In standard ISAM, PL/I uses CONDITION CODES and COBOL uses FILE STATUS. The *A Series PL/I Reference Manual* describes CONDITION CODES. FILE STATUS is described in the *A Series COBOL ANSI-74 Programming Reference Manual Volume 2: Product Interfaces*.

Primitive ISAM Result Information

The value returned when primitive ISAM is used is a 48-bit word that is not 0 (zero) when an exception condition exists and zero (0) when no exception condition occurs. Specific, individual bits are used to indicate the exception condition. If several different exceptions occur, the corresponding bit is ON for each condition and creates the possibility of reporting back several exceptions for a single request. The rightmost and least significant bit (bit 0) is used for a specific purpose. Bit 0 is ON when any exception condition occurs and OFF when no exception exists. The remaining bits convey the following meanings:

Bit	Meaning
1:1	A hardware error, — for example, a parity error — occurred while processing the request. Another bit (7, 8, 9, or 10) is turned ON to further define the problem.
2:1	An attempt was made to read or write beyond end-of-file (EOF).
3:1	No record was found in the file whose key matches the requested key.
4:1	No space is available in the file to contain the record just written. (This condition applies for adding records to the file; it does not apply to file creation.)
5:1	A request was made to add a record to the file, and the key contained in the record matched a record that existed in the file. Refer to bit 6.
6:1	A record was added to the file and the key of the record matched an existing record of the file. The duplicate key option permits or does not permit this situation. When duplicates are allowed, both bit 5 and bit 6 are ON to indicate that a duplicate record has been added. Refer to bit 5.
7:1	A hardware error occurred while reading a data record. Bit 1 is also ON.
8:1	A hardware error occurred while writing a data record. Bit 1 is also ON.
9:1	A hardware error occurred while reading an ISAM table. Bit 1 is also ON.
10:1	A hardware error occurred while writing an ISAM table. Bit 1 is also ON.
11:1	This bit is not used.
12:1	An attempt was made to open the ISAM file, and the parameters passed to ISOPEN failed to meet one or more requirements. The first of four stack words parameter must be a Stuffed Indirect Reference Word (SIRW). The file must be declared in a block that will be entered no sooner than the block in which the four stack words reside. The file must not reside in a different stack from the program performing the ISOPEN. The block containing the four stack words must also contain a file, an array, or something that causes a tag-6 word for the block. The tags of all four stack words must be zero (0). The key must be defined to be contained in the records, have a size greater than zero (0), and have a valid mode.
13:1	An attempt was made to open a non-ISAM file.
14:1	Either the file has not been opened, or the open type does not permit the request. (For example, an ISWRITE is not permitted on a file opened as INPUT.)

continued

PL/I Indexed Sequential-Access Method (PLIISAM)

continued

Bit	Meaning
15:1	An ISREWRITE was requested, and the key of the record being rewritten does not match the key in the last record read, or the previous request was not an ISREAD or an ISREADNEXT.
16:1	The ISAM file is being created, and the record just written did not maintain proper file sequence. Records must be presented in sequence during file creation. A duplicate record also causes this bit to be ON when duplicates are not allowed.
17:1	The value of AREAS specified is not large enough to contain the data records written in the prime data area during file creation.
18:1	ISOPEN is requested to open an already open file.
19:1	In an IPC environment, one program closed an ISAM file, and another attempted an I/O after the file was closed.
20:1	An ISWRITE is requested, and the key supplied does not match the key contained in the supplied record.
21:1	The ISAM file is not a direct file.
22:1	An attempt was made to write a record containing the deleted record indicator (hex FF) in the first byte.
23:1	This bit indicates a PL/I program error condition. The program is requesting an I/O that is not allowed for keyed files. An ON condition is raised in the PL/I program.
24:1	This bit is ON if ISOPEN is requested — by way of an open action — to open the ISAM file using the PRESENT or AVAILABLE file attribute; this bit also indicates that the desired file could not be located. Refer to bit 43:8.
43:8	This bit contains the result of testing the PRESENT or AVAILABLE file attributes in the ISOPEN procedure. If the file could not be opened, bit 24:1 is also ON.
46:1	This bit is ON if the physical update I/O action is ON, the wait update I/O option is OFF, and an I/O error occurred as the result of doing an update I/O in the previous invocation of an ISAM procedure. Bit 1:1 is ON, and either bit 8:1 or bit 10:1 is ON.

Section 11

RLTABLEGEN Utility

General Information

This section describes the SYSTEM/RLTABLEGEN utility, which permits the system to accept nonstandard tape labels by constructing value arrays and compiling them into the operating system.

Installation Defined Tape Labels

The MCP LABEL recognition routine READALABEL recognizes a large class of installation-defined tape labels in addition to standard tape labels. Refer to the *A Series I/O Subsystem Programming Guide* for a discussion of standard tape LABEL formats.

To invoke the LABEL recognition routine, a description of the desired nonstandard labels must be written and passed through a table-building program called RLTABLEGEN. RLTABLEGEN places the information concerning nonstandard tape labels into a value array and then compiles the value array into the present MCP called SYSTEM/MCP. The new MCP is called SYSTEM/NEWMCP. The installation then changes the MCP to SYSTEM/NEWMCP through the CM (Change MCP) system command. Refer to the *A Series System Commands Operations Reference Manual* for a description of this command.

Installation-defined tape labels can be used in the same manner as standard tape labels. However, the following restrictions apply:

- The LABEL records must be a contiguous group of EBCDIC records at the beginning of the tape, and must be separated from the data by one tape mark.
- The LABEL type must be determined by examination of the first record only.
- The LABEL record size must not exceed 229 characters.
- The MCP does not recognize any user labels – for example, users' trailer label (UTL) and users' header label (UHL), – on these LABEL types.
- No up tape labels are recognized. That is, once the file is opened and the tape is positioned to the beginning of the data, the tape is treated as an unlabeled tape – that is, LABEL=OMITTEDEOF.
- Because READALABEL checks for the appearance of installation-defined labels before checking for standard labels, the recognition sequence must be sufficiently complete so that it does not interfere with standard LABEL recognition.

Running the RLTABLEGEN Utility

RLTABLEGEN accepts source-input records from the input file whose internal name is CARD. The output file generates a listing from the printer file named LINE.

The line printer listing contains a pseudo-reproduction of the input commands that RLTABLEGEN generates from the table produced by the input commands. Successful operation occurs when the essential information in the input records is contained in the output listing. The end of the output listing contains the value array generated by the program.

RLTABLEGEN automatically reconfigures the MCP by automatically starting the NEWP compiler. RLTABLEGEN requires that the present MCP have the title SYSTEM/MCP and that an MCP symbolic file having the title SYMBOL/MCP be available.

The following Work Flow Language (WFL) job can be used to run RLTABLEGEN:

```
<i> BEGIN JOB;  
    RUN SYSTEM/RLTABLEGEN; EBCDIC CARD  
    <RLTABLEGEN commands>  
<i> END JOB
```

Explanation

<i>

Specifies an invalid character. When RLTABLEGEN is run from the ODT or a remote terminal, the <i> variable is the question mark (?). When RLTABLEGEN is run from the card reader, the <i> variable can be any invalid punch. An <i> variable is optional when RLTABLEGEN is run from the ODT or a remote terminal; however, it is required when RLTABLEGEN is run from the a reader.

<rltablegen commands>

See "RLTABLEGEN Commands" for definitions of these commands.

Input to the RLTABLEGEN Utility

RLTABLEGEN input consists of a number of LABEL descriptions.

Label Description Format

A label description must consist of the following two divisions:

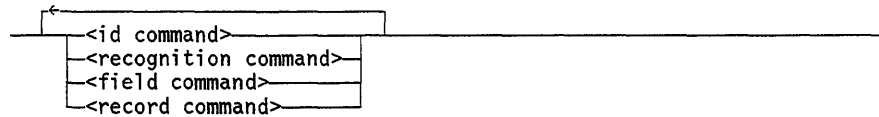
- The ID division
- The FIELD division

A label description is created using the commands described under "RLTABLEGEN Commands."

RLTABLEGEN Commands

The following commands can be used as input to RLTABLEGEN:

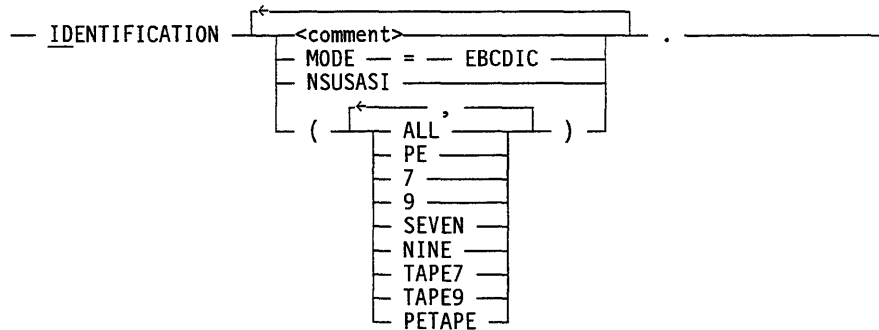
<rtablegen commands>



ID Command

The ID command, which starts the ID division, identifies the character code and the recording mode to be used for the LABEL description.

<id command>



Explanation

<comment>

Specifies an EBCDIC character string. It is used to document the program.

EBCDIC

Specifies that the EBCDIC mode is to be used for a tape label.

NSUSASI

Required for nonstandard ANSI tapes in which HDR2 and EOF2 LABEL records are omitted. (The first 4 characters of the second file header are HDR2, and the first 4

RLTABLEGEN Utility

characters of the second end-of-file (EOF) LABEL are EOF2.) Most forms of the IBM® disk operating systems (DOS) omit these LABEL records. If RLTABLEGEN is used for such tapes and this attribute is not specified, the tape file closes with a LABEL error because the MCP expects to find the nonpresent EOF2 record.

ALL

Identifies a tape label that can be used on a phase-encoded (PE) 9-track magnetic tape, a 7-track magnetic tape, and a 9-track magnetic tape.

7 SEVEN TAPE7

Identify a tape label that can be used on a 7-track magnetic tape.

9 NINE TAPE9

Identify a tape label that can be used on a 9-track magnetic tape.

PE PETAPE

Identify a tape label that can be used on a PE 9-track magnetic tape.

Examples

The following ID command identifies the LABEL description as EBCDIC for a PE 9-track magnetic tape or a 9-track magnetic tape:

```
ID MODE = EBCDIC FOR (PE, NINE) TRACK TAPES.
```

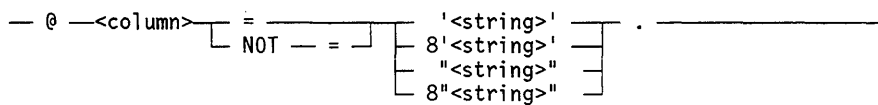
The following ID command identifies the nonstandard USASI LABEL description for PE 9-track, 7-track, and 9-track magnetic tapes as EBCDIC:

```
ID FOR NSUSASI LABELS (ALL) MODE = EBCDIC.
```

RECOGNITION Command

The RECOGNITION command, which belongs to the ID division, follows the ID command. It is used to describe the fields used to recognize this label. This command can appear as often as necessary.

<recognition command>



Explanation

<column>

An integer from 01 to 80 that specifies the column where the first character of the string appears.

'<string>'

"<string>"

Specifies a string of characters that fills the designated columns. These characters can include any alphanumeric or graphic character except the double quotation mark ("). However, if "<string>" contains an apostrophe ('), it must be enclosed in double quotations marks.

8'<string>'

8"<string>"

Specifies a string of 8-bit (EBCDIC) characters. These characters can include any alphanumeric or graphic character except the double quotation mark ("). If 8"<string>" contains an apostrophe ('), it must be enclosed in double quotation marks.

Examples

The following recognition command indicates VOL1 should be found in columns 1 through 4 of the tape label:

```
@ 01 = "VOL1".
```

The following recognition command indicates that a blank should be found in column 29 of the tape label:

```
@ 29 = " ".
```

FIELD Command

The FIELD command starts the FIELD division, which contains descriptions of the various fields found in the tape label. This command also denotes the beginning of the LABEL description and marks the end of the RECOGNITION command and the ID division.

RLTABLEGEN Utility

<field command>

— FIELDS —————
 └─<comment>┘

Explanation

<comment>

Specifies an EBCDIC character string.

Example

In the following example, the FIELD division is specified:

```
FIELDS BEGINNING OF LABEL DESCRIPTION.
```

RECORD Command

The RECORD command describes the various fields found in the label. Each field is used to store file attribute information. The file attributes BLOCKSIZE, CREATIONDATE, CYCLE, DENSITY, FILESECTION, MAXRECSIZE, MFID, FID, SERIALNO, MINRECSIZE, PARITY, UNITS, and VERSION are all discussed at length in the *A Series File Attributes Programming Reference Manual*.

The RECORD command, which belongs to the FIELD division, must follow the FIELD command. Label fields can be specified as bit, string, or number fields. Each field is used to store file attribute information.

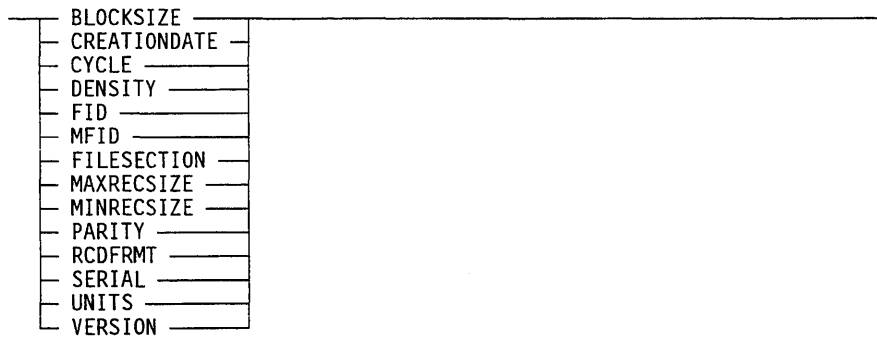
Each attribute value is of an assumed type. The character type (EBCDIC) is assumed to be the type specified by the MODE clause in the ID statement. Attributes of STRING type can be requalified to override the expressed mode but cannot be changed to type BINARY or NUMBER.

NUMBER or BINARY attributes can be requalified to any mode or type except STRING.

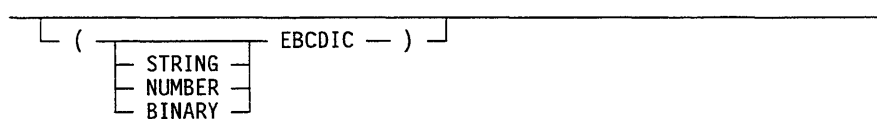
<record command>

← RECORD — ; —<record number> —————→
 └─<attribute field> @ —<column> FOR —<length> ┘
 └─<mode>┘
→ . —————→

<attribute field>



<mode>



Explanation

RECORD ; <record number>

Groups attribute fields according to the records in which they appear. Counting within records of columns and lengths begins at 1. Each record number must reference a higher record than the previous one.

<attribute field>

Specifies the file attribute with which the value in the field is associated. The file attributes are explained later in this section.

<column>

An integer from 01 through 80 that represents the starting column location within the current record.

<length>

An integer that represents the length of the field being described.

<mode>

Specifies the mode that is to be used.

EBCDIC

Specifies that EBCDIC mode is to be used. Information is in 8-bit characters.

STRING

Indicates that the attribute field contains a string of characters. Any alphanumeric or graphic character except the double quotation mark (") can be included.

The STRING attributes can be requalified to override the expressed <mode>; however, they cannot be changed to NUMBER or BINARY.

NUMBER

Indicates that the attribute field contains an integer—a digit string of no more than 11 digits.

The value of these attributes can be changed to any mode or type except STRING.

BINARY

Indicates that the attribute field contains a binary number.

The value of these attributes can be changed to any mode or type except STRING.

File Attributes

The following file attributes can be specified with the RECORD command:

BLOCKSIZE

Indicates the block length of the file. BLOCKSIZE is type NUMBER. The value of BLOCKSIZE is assumed to be an integer of no more than 11 digits and is significant only if FILETYPE 7 or 8 is specified when the file is opened.

CREATIONDATE

Returns the date a file was created. CREATIONDATE is type NUMBER. The value of CREATIONDATE is assumed to be an integer of no more than 11 digits and is significant only if FILETYPE 7 or 8 is specified when the file is opened.

CYCLE

Denotes the different generations of a permanent file. CYCLE is used in conjunction with the VERSION attribute to determine the genealogy of a file. CYCLE is type NUMBER. The value of CYCLE is assumed to be an integer of no more than 11 digits and is significant only if FILETYPE 7 or 8 is specified when the file is opened.

DENSITY

Automatically sets the field based on the density of the LABEL records. DENSITY is type BINARY. If this attribute is specified, it must reflect the density in its low-order 2 bits.

**FID
MFID**

FID is the last and MFID is the first identifier in the TITLE of a logical file on a tape containing multiple files. MFID and FID are used in the external name of that physical tape file.

Both FID and MFID are type STRING. The maximum length is 17 characters. Either field is optional. If both fields are missing or if both referenced fields are blank, the tape is labeled "untitled". All leading and trailing blanks are deleted from the two fields.

FILESECTION

Specifies the ISO, BSI, and ANSI file section number of the first header LABEL record. FILESECTION is type NUMBER. The value of FILESECTION is assumed to be an integer of no more than 11 digits and is significant only if FILETYPE 7 or 8 is specified when the file is opened.

MAXRECSIZE

Specifies the maximum size of records in the logical file. MAXRECSIZE is type NUMBER. The value of MAXRECSIZE is assumed to be an integer of no more than 11 digits and is significant only if FILETYPE 7 or 8 is specified when the file is opened.

MINRECSIZE

Specifies the minimum size of records in the logical file. MINRECSIZE is type NUMBER. The value of MINRECSIZE is assumed to be an integer of no more than 11 digits and is significant only if FILETYPE 7 or 8 is specified when the file is opened.

PARITY

This field is automatically set to the parity of the LABEL records; but if it is specified, the field must be in accordance with the manner in which this LABEL attribute works on A Series standard labels. PARITY is type BINARY. (If the low-order bit of the field is equal to one, standard parity is implied.)

RCDFRMT

Indicates the FILETYPE of the file. RCDFRMT is type BINARY. The low-order 8 bits of the field are assumed to be an EBCDIC letter. The following demonstrates the conversion from EBCDIC letter to FILETYPE value:

Letter	FILETYPE
F	0
D	1
V	2
I	4
Z	6

continued

RLTABLEGEN Utility

continued

Letter	FILETYPE
Any other letter	3

SERIAL

Indicates the SERIALNO value of the file. SERIAL is type STRING. The maximum length of the field is 6 characters. If no field is specified for SERIAL, the SERIALNO is assumed to be all zeros.

UNITS

Indicates whether or not the transfer of data in the file is word- or character-oriented. UNITS is type BINARY.

VERSION

Distinguishes successive iterations of the same generation of a permanent file. VERSION is used in conjunction with the CYCLE attribute. VERSION is type NUMBER. The value of VERSION is assumed to be an integer of no more than 11 digits and is significant only if FILETYPE 7 or 8 is specified when the file is opened.

Examples

The following RECORD command specifies the first record:

```
RECORD ; 1.
```

The following RECORD command specifies that the block length of a file is found in columns 6 through 10. It also declares the mode to be NUMBER and the character type to be EBCDIC.

```
BLOCKSIZE @06 FOR 5 (NUMBER EBCDIC).
```

The following RECORD command specifies that the creation date of a file is found in columns 43 through 47:

```
CREATIONDATE @43 FOR 5.
```

The following RECORD command specifies that the generation of a permanent file is found in columns 36 through 39:

```
CYCLE @36 FOR 4.
```

The following RECORD command specifies that the FID title is found in columns 5 through 21:

```
FID @05 FOR 17.
```

The following RECORD command specifies that the MFID title is found in columns 12 through 28:

```
MFID          @12 FOR 17.
```

The following RECORD command specifies that the ISO, BSI, and ANSI file section number of the first header LABEL record is found in columns 32 through 35:

```
FILESECTION   @32 FOR 4.
```

The following RECORD command specifies that the maximum record size of a logical file is found in columns 11 through 15:

```
MAXRECSIZE    @11 FOR 5.
```

The following RECORD command specifies that the minimum record size of a logical file is found in columns 16 through 19:

```
MINRECSIZE    @16 FOR 4.
```

The following RECORD command specifies that the parity that is used on a file is found in column 18:

```
PARITY        @18 FOR 1.
```

The following RECORD command specifies that the record format of a file is found in column 5:

```
RCDFRMT       @05 FOR 1.
```

The following RECORD command specifies that the transfer of data in a file is word- or character-oriented in column 24:

```
UNITS         @24 FOR 1.
```

The following RECORD command specifies that the distinguishing of successive iterations of the same generation of a permanent file is found in columns 40 and 41:

```
VERSION       @40 FOR 2.
```


Section 12

SORT Utility

This section describes the SORT utility. SORT is a procedure in the operating system that sorts a file or a set of records into a single file of ordered records. SORT can also merge a set of presorted files into a single ordered file. SORT can be activated through ALGOL, COBOL, COBOL74, COBOL85, FORTRAN77, PL/I, C, Pascal, Pascal83, the sort procedural interface, the Interactive Sort (ISORT) utility, or the SORT compiler. Refer to the *A Series Interactive Sort (ISORT) Operations Guide* for more information on the Interactive Sort utility.

The SORT compiler supports the MultiLingual System (MLS), allowing error and warning messages as well as header and trailer messages to be reworded or translated according to need. For more information, refer to the *A Series Message Translation Utility (MSGTRANS) Operations Guide*.

SORT accepts a number of parameters that specify where to get the input, the comparison technique to be used, where to place the sorted records, what system resources to use for the sort operations, and so forth.

All languages except the SORT compiler require you to provide a certain set of parameters – such as input, output, and comparison. The SORT compiler builds the calls on the SORT intrinsic, based on the information you enter in the SORT language. Refer to the *A Series SORT Language Programming Reference Manual* for a detailed description of the SORT compiler. This section does not discuss the SORT compiler.

SORT can be operated in four different modes:

- Disk-only mode
- Tape-only mode
- Integrated tape and disk (ITD) mode
- Memory mode

Sort operations can be restarted.

Sorting is performed in two phases:

- The sorting or stringing phase
- The merging phase

SORT begins by reading records from the input file or procedure and sorting them into groups, called *strings*, on the sort work files.

After the last input record is read, the merging phase begins. The strings of sorted records are merged into larger strings until the result is one string containing the

ordered records from the input file. The ordered records are then written to the output file or procedure, and the SORT utility terminates.

SORT Parameters

The following text describes the meaning of the SORT parameters and their effects on the operation of the SORT utility.

The SORT parameters should be chosen based on the relative priority of the job and the importance of total system efficiency. Knowing what particular sort is necessary and experimentation can provide optimized parameter values.

Refer to the appropriate volumes of the following manuals for more details on the SORT parameters:

- *A Series ALGOL Programming Reference Manual, Volume 1: Basic Implementation*
- *A Series C Programming Reference Manual*
- *A Series COBOL ANSI-85 Programming Reference Manual, Volume 1: Basic Implementation*
- *A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation*
- *A Series COBOL ANSI-68 Programming Reference Manual*
- *A Series FORTRAN77 Programming Reference Manual*
- *A Series Pascal Programming Reference Manual, Volume 1: Basic Implementation*
- *A Series Pascal ANSI-83 Programming Reference Manual*

Input Options

You can specify an input file or an input procedure to indicate how the records are to be provided to the SORT utility.

If an input file is specified, SORT invokes a standard input procedure that opens the file and reads the records. For efficiency, the input file should be blocked in increments greater than 500 words and should contain approximately 50 (or more) records per block. The input file must be closed when it is passed to SORT.

The correct values of the NEWFILE attribute for a file passed to the SORT utility are FALSE for an input file and TRUE for an output file. If the NEWFILE attribute is set to the wrong value, SORT changes it to the correct value before opening the file. This change is necessary when the same file is used for both the input and the output.

If your input procedure is specified as the input option, the procedure is called to provide the input records to SORT. Control is passed to the input procedure before the records are sorted. The records released from the input procedure are then sorted. The input procedure must not contain any SORT or MERGE statements.

SORT programs that contain lengthy input or output procedures can impact other jobs because much of the memory used by SORT is nonoverlayable (save) space. The use of input or output procedures should not be discouraged but should be considered in proper perspective for the job to be accomplished. In some cases, overall system performance can be improved by having the input procedure produce a file that is read by SORT and having SORT produce a file that is processed by the output procedures. The process of calling input or output procedures does not present an excessive burden to SORT or the system.

Output Options

You can specify an output file or an output procedure to indicate how the sorted records are to be returned from the SORT utility.

If an output file is specified, SORT invokes a standard output procedure that writes the sorted records to the output file. For efficiency, the output file should be blocked in increments greater than 500 words and should contain approximately 100 (or more) records per block. The output file must be closed when it is passed to SORT. The file is closed after the SORT statement is completed.

If you specify an output procedure, the procedure is called once for each sorted record and once for the end of output action. The output procedure must not contain any SORT or MERGE statements.

Compare Procedure

SORT calls the compare procedure to determine which of two records should be used next in the sorting process. Different keys or fields in the records can be compared; however, programs can be made more efficient by simplifying the individual keys and consolidating them into a single key. The comparison technique is determined entirely by you in the compare procedure.

In COBOL, the length and number of keys directly affects the amount of time required for comparison. A large number of keys scattered in the record should not be used because setup time increases comparison overhead. Arithmetic or numeric comparisons are generally faster than string comparisons. COBOL sorts should use the EBCDIC character set for string comparison whenever possible.

In ALGOL, the compare procedure should be coded to return TRUE when the two arrays are unequal and the first array must precede the second array. A FALSE should be returned when the arrays are equal or when the second array must precede the first array. In ALGOL sort operations, partial word comparisons are normally faster than string comparisons when characters within a word are tested.

SORT attempts to recognize when the input data is less than 40 percent in sequence and switches comparison either from ascending to descending or from descending to ascending. SORT remembers the change of mode and processes the data accordingly. The switch is done as often as necessary in order to produce longer strings. Given a set of input data in exact reverse sequence, SORT produces two strings – rather than the maximum use of strings – and completes the sort much faster.

Number of Tapes

This value specifies the number of tapes to be used in sorting. If the number of tapes is greater than 0, an integrated tape/disk sort or a tape-only sort occurs. You can use between 3 and 8 tapes. With integrated tape/disk or tape-only sorting, using more sort tapes generally increases sorting speed. However, gains made by using more than 5 sort tapes are marginal.

Record Size

This value specifies the length of the records submitted to SORT. Variable-length records should be edited into fixed-length records by an input procedure and edited back to variable-length records by an output procedure.

The record length specifies the length in words or characters – depending on whether the array parameters of the procedure are word or character arrays, respectively – of the largest item to be sorted.

In ALGOL, if the record size is not a positive integer, the largest integer not greater than the absolute value of the expression is used. For example, a record length of 12 would be used if an expression had a value of -12.995. If the value of the record size is 0, the program terminates.

Memory Size

The memory size specifies the number of words in memory to be used when sorting. If memory size is not specified, SORT assumes a default value of 12,000 words. SORT calculates the number of records to be kept simultaneously in memory with a limit of 65535 records. SORT also calculates the number of buffers to be used for each string, with a limit of 256 buffers per string.

The memory size estimate determines stringing and merging vector sizes, which, in turn, control string length and merging. Producing a small number of long strings is desired for sorting because fewer merge passes are required. Increasing the memory available to SORT is the most effective way to increase sorting efficiency. However, if the memory size is increased beyond the optimum, SORT might run slower rather than faster. The optimum memory size varies according to file size, which includes the record size, the blocking factor, and the number of records. Optimum memory size is also affected by the number of other jobs in the mix at the time of the sort operation.

SORT attempts to be processor-bound in both the stringing and merging phase (as opposed to processor-bound during the stringing phase and I/O-bound during the merging phase). Unless the memory size specified is relatively small for a given sort, SORT achieves the goal of being processor-bound. When this condition is obtained, speed improvements can be realized only by methods that reduce processor time. Decreasing the amount of processor time helps system throughput as well as reducing sort timings.

In general, memory allocation proceeds through the following steps:

1. Memory size provided by the program is reduced by 1,500 words. The reduced size is used for all subsequent calculations. The reduction is a generous estimate of the amount of space required for working storage and the space required for various SORT procedures.
2. A buffer size is selected for the internal disk or tape files or both. SORT tries to select buffer sizes so that it does not become I/O-bound. For disk sorting, SORT normally allocates two buffers for each string. For tape sorting with n tapes, SORT allocates $1/n$ th of memory as buffers for each tape.
3. During executions of the stringing phase, two output buffers are normally allocated; thus, the remainder of memory is left for the sort vector. During execution of the merge phase, virtually all available memory is used for buffers.

In general, Unisys recommends a memory estimate of 4,000 or more words. Memory estimates of 20,000 or more are recommended only on large systems. Information on how to determine memory size for different sorting modes is given later in this section.

When the memory allocation is completed, SORT initializes its internal files with the proper computed attributes. SORT takes into account any change made to these file attributes through file equation. However, these file attributes override memory size specification; therefore, actual memory size can be less than or greater than the memory size specified to SORT. Refer to "SORT Files" in this section for more information on file equation of SORT internal files.

Determining Memory Size for Disk Sorting

Use the following to determine memory size for disk sorting:

1. Convert the sort record size to the number of words required to contain a single record. For example, a 1-character record requires 1 word, and fifteen 6-bit characters require 2 words, while fifteen 8-bit characters require 3 words.
2. Add 3 additional words to the record size—to be used only by SORT.
3. Multiply the number obtained in step 2 by the desired number of records according to the following steps:
 - For fast sorting, memory size should provide enough space to contain at least 2,000 records.
 - For reasonably fast sorting, memory size should provide enough space to contain at least 1,200 records.
 - For adequate sorting, memory size should provide enough space for 600 records, as a general rule.
4. After memory has been computed for the number of records times the record size, add 1,500 words to provide for sort working space.

Determining Memory Size for Tape Sorting

Use the following to determine memory size for tape sorting:

1. Convert the record size to words.
2. Add 3 additional words – to be used only by SORT.
3. Multiply the number obtained in step 2 by the number of tapes specified in the SORT statement.
4. Multiply the number obtained in step 3 by one of the following:
 - 300 for fast sorting
 - 200 for reasonably fast sorting
 - 100 for adequate sorting
5. Add 1,500 words to provide for sort working space.

Tape sorts are similar to disk sorts in that providing more memory generally yields faster sorts. However, the point of diminishing returns is more data-dependent for tape sorting. In general, using more sort work tapes rather than providing additional memory is more efficient. Providing more memory and more tapes is ideal when speed is the most important factor.

Determining Memory Size for Memory Sorting

Use the following for memory-only sorting:

1. Convert the record size to words.
2. Add 3 additional words – to be used only by SORT.
3. Multiply the number obtained in step 2 by the number of records to be sorted.

For COBOL sort operations, the record size is determined from the SD description, rounded up to the number of words required to contain the record.

Disk Size

The disk size specifies the amount of disk storage in words to be used during the sort operation. If disk size is not specified, the default disk size is used. The default disk size for ALGOL and PL/I is 600,000 words, and for COBOL 900,000 words.

If disk size is set to 0 and the number of tapes is greater than 0, a tape-only sort occurs.

If disk size is set to 0 and no tapes are specified, SORT operates in memory-only mode.

Normally, SORT allocates 20 disk areas with varying area sizes, depending on your disk estimate.

Disk size significantly affects the speed of disk sorting. If sufficient disk is not made available to contain the output of the merge phase, SORT is unable to merge as many

disk strings. Therefore, SORT merges fewer strings, when possible, and attempts to reduce the risk of terminating due to lack of disk space (SORT ERROR #5).

Disk estimates must be large enough to accommodate your input file. The amount of disk space required for merging depends on several factors. Estimating a precise amount of disk space is difficult because of the gaps created by unfilled buffers at the end of strings. However, a safe disk estimate is two times the input file size. The following method for estimating disk size is suggested:

1. Convert the record size to words. Do not add 3 additional words as described under "Memory Size" in this section.)
2. Multiply the record size (in words) by the number of records to be sorted.
3. Multiply the number obtained by step 2 by one of the following:
 - 1.5 to obtain a near minimum estimate.
 - 2.25 to obtain a safe estimate.
 - 3.5 or more if a restartable sort is to be performed.

SORT Operating Modes

SORT can operate in four different modes: disk-only, tape-only, integrated tape and disk, or memory-only.

The combination of disk size and number of tapes determines the sort mode as shown in Table 12-1.

Table 12-1. Determining SORT Operating Mode

Number of Tapes	Disk Size	Mode
Not = 0	0	Tape-only
Not = 0	Not = 0	ITD
0	Not = 0	Disk-only
0	0	Memory-only

The following text identifies important characteristics of various SORT modes. The SORT modes are then described in detail.

Disk-only mode

The following are characteristics of disk-only sorting mode:

- Disk-only is generally faster than tape-only mode.
- A disk is the most reliable peripheral device.
- Less operator intervention is required than when you use tape-only mode.

- The sort operation is limited by disk resources.

Tape-only mode

The following are characteristics of tape-only sorting mode:

- The input file can be an indefinite length.
- A particular machine configuration is required – that is, you need several tape drives that must be capable of performing backward reads.

ITD mode

The following are characteristics of ITD sorting mode:

- The disk develops longer strings on tape.
- The input file can be an indefinite length.

Memory-only mode

The following are characteristics of memory-only sorting mode:

- Memory-only is generally the fastest mode.
- Memory-only is the mode least likely to encounter I/O problems.
- Memory-only mode is limited to the amount of data it can sort:
 - It can sort a large number of small records
 - It can sort a small number of large records
 - It cannot sort a large number of large records

Disk-Only Mode

If the number of tapes is 0 and the disk size is greater than 0, SORT operates in disk-only mode. In this mode, all sort work files are maintained on disk. Disk-only mode is generally the fastest mode.

Disk-Only Stringing Phase

Strings are written serially to the sort work file, titled DISKF, as they are formed during the stringing process. For each string, a disk control word is retained for use during the merge phase. A disk file, titled DISKC, is allocated for these control word records.

If disk space is exhausted during the stringing phase, the sort operation is terminated.

Disk-Only Merging Phase

The disk merging phase begins after the stringing phase is completed and merges strings into longer strings on disk. As each newly merged string is formed, a new control word is built. When the number of strings remaining to be merged is less than or equal to the number of strings that can be merged at one time, SORT writes the records to the output file or procedure.

During the merging phase, wraparound on the work file might occur. Wraparound occurs when merged records are written at the beginning of the work file. Wraparound is possible because the strings occupying the space at the beginning of the work file have already been handled by the merge operation. This wraparound action means that sorting can be done into the same work file.

Tape-Only Mode

If the number of tapes is greater than 0 (zero), and disk size is specifically set to 0, a tape-only sort is performed. In this mode, all sort work files are maintained on tape. It is advisable to use tapes in good condition when performing tape sorts. You can specify from 3 to 8 tapes for SORT to use as work files. The sorting method used for tape sorting is the polyphase merge/reverse technique.

Unlike a disk-only sort, where strings are written serially on the work file as they are formed, special string distributions and string-sequencing techniques are required for a tape sort. String distributions are based on a generalized Fibonacci number series. The string sequencing (ascending or descending) is specifically designed for reverse tape reads.

Tape-Only Stringing Phase

In the initial stringing phase, one work tape is designated as the first merge output tape and thus is not used during the stringing process. For example, in a 3-tape sort, strings are written to only two tapes. Strings are then dispersed to the stringing tapes in a special pattern until the current level in the distribution is satisfied. The distribution is transferred to the next level, and stringing continues. When the last input record is strung, the stringing phase is complete.

A special-string sequencing pattern is required by SORT because of the reverse tape-read technique used in the merging phase. The pattern is as follows:

1. Strings are written to an individual tape in alternating sequence – ascending, descending, ascending, and so on).
2. In an ascending sort operation, all tapes except the tape with the odd number of strings – that is, the last tape – begins with a descending string. The odd tape begins with an ascending string. In a descending sort, the sequence pattern is reversed.

SORT Utility

The following example depicts the stringing phase of a 5-tape ascending sort:

	TAPE 1	TAPE 2	TAPE 3	TAPE 4
	-----	-----	-----	-----
13 Strings	D	D	D	A
Distribution is: 2,4,4,3	A	A	A	D
		D	D	A
		A	A	

D = DESCENDING STRING. A = ASCENDING STRING.

During the stringing phase, SORT moves cyclically on the tapes—that is, from tape 1 to tape n , where n represents an integer from 2 through 8—looking for a tape to string. If the distribution is satisfied on a given tape, SORT moves to the next tape. When the distribution is satisfied on all tapes, the Fibonacci distribution is transferred to the next level.

At the completion of the stringing phase, if the number of strings distributed is less than the desired Fibonacci distribution level, the tapes are padded with dummy strings to fill out the distribution. The dummy strings are recorded internally but are not physically written on the tapes.

Tape-Only Merging Phase

The merging phase of the tape sort operation uses the polyphase merge/reverse tape read technique. In the polyphase method, strings from working tapes are merged to a designated output tape until one of the tapes contains no more strings. This tape now becomes the output tape and is the end of a merge pass or level. The string totals on the remaining tapes now correspond to the next lower level in the distribution table. The merging operation continues until one final string can be written to the output file or procedure.

ITD Mode

If the number of tapes specified is greater than 0 and the disk size is greater than 0, SORT operates in ITD mode. The ITD or disk/tape mode of sorting uses disk work files with tape backup.

ITD sorting can improve on tape-only sorting by 50 percent or more. The reason for this degree of improvement is that fewer strings are created on tape, which causes tape merging to be completed much sooner. The amount of improvement depends on the inherent sequence of the data and the amount of disk space provided. In most cases, 100,000 words or less of disk space is sufficient to obtain the increased speed from an ITD sort.

SORT begins stringing records on disk; however, if disk space is exhausted during stringing operations, a special merge operation is performed to tape, and the sort

operation is not terminated. This creates strings on tape in the normal tape distribution, but the number of strings written on tape is less than that resulting from a tape-only sort operation. Stringing then resumes normally on disk until disk space is exhausted again. When the stringing phase is complete, a regular tape merge is performed.

During an ITD sort operation, file equations are not used for the internal tape files used during the stringing phase. If SORT requires a scratch tape, either a scratch tape can be mounted or the sorting program can be terminated.

If disk space is exhausted during the merging phase of an ITD sort, the strings are merged to tape, and the remaining merging operations are completed on tape.

One advantage of the ITD sort mode is its ability to circumvent a limited disk resource to sort large files. Another advantage is that it reduces tape merge time because of the use of disk space to consolidate many short strings into a few longer strings.

Memory-Only Mode

If no sort work files are specified on tapes, and the disk size is specifically set to 0, a memory-only sort operation is performed. Failure to set the disk size to 0 results in the default value for disk being used, and a disk-only sort operation occurs.

SORT does not open sort files; it attempts to read your input directly into memory. If sort memory is filled before the last user input record is read, SORT terminates with SORT ERROR #3. If the specified amount of memory can contain all input records, SORT proceeds normally to produce user output.

Memory-only sorting is of particular value when the number of records to be sorted is small. To determine whether or not you have an optimum number of records to sort, use the following formula as a guide:

- Multiply the number of records to be sorted by the size of the records measured in words.
- The product of this result must be less than the memory size specification

SORT Files

SORT uses two disk files when disk or ITD sorts are requested: a control file to store the control records and a work file to store the data or records being sorted. The internal name for the control file is DISK`C`, and the title is SORT/DISK`C`. The internal name for the work file is DISK`F`, and the title is SORT/DISK`F`.

SORT uses between 3 and 8 tape files when ITD or tape-only sorts are requested.

Control Files

The control file is normally a very small disk file whose size is based on the maximum number of strings that can be produced for the sort operation currently being executed.

Control file records are 3 words in length, and blocks are 90 words in length. The maximum number of control file rows is 64, and the number of physical blocks per row is 4—unless the amount of disk space provided is extremely large. The last two rows of the control file contain restart information when restartable sorts are requested. When a restartable sort is desired, file equation should be used to give a unique title to the control file. For example:

```
<i> FILE DISKC (TITLE=JOBNAME/SORTCONTROL)
```

Other attributes in the file equation for the control file can be modified, but the probability of failure is high if attributes such as AREAS, AREASIZE, MAXRECSIZE, and BLOCKSIZE are modified by file equation.

Work Files

The work file size is provided explicitly or implicitly by your program. SORT first determines the desired block size and then computes the number of disk rows provided by you. The maximum number of rows that SORT allocates to the work file is 183; partial rows are rounded up to a full row. Row sizes do not exceed 1,320 segments unless an extremely large amount of disk is provided. When a restartable sort is requested, file equation should be used to give a unique title to the work file. For example:

```
<i> FILE DISKF (TITLE=JOBNAME/SORTWORK)
```

A new BLOCKSIZE and MAXRECSIZE can be provided in the file equation for the work file. Other attributes in the file equation for the work file can be modified, but the probability of failure is high when attributes such as AREAS and AREASIZE are modified by file equation.

The BLOCKSIZE and MAXRECSIZE attributes must be compatible in order to open the file. Modifying the buffer size overrides the normal memory allocation algorithms of SORT. However, care should be exercised when these values are changed. When the buffer size is increased, the number of disk segments per disk row is proportionately increased, and SORT proceeds using the larger disk rows. If the buffer size is decreased, the number of disk segments per disk row is proportionately decreased, which might result in a work file that is not large enough to accomplish the sort. One method of alleviating this condition is to specify a larger quantity of disk space. Modification of the buffer size is usually less effective than providing a different memory size for use by SORT. However, sort operations are always data-dependent, and with some ordering of data, a better sorting can be obtained by judicious selection of buffer size.

Tape Files

SORT uses tape files when ITD or tape-only sort operations are requested. SORT can use between 3 and 8 tapes. To eliminate the necessity for operator intervention to resolve “DUP FILE” messages, the sort tape files are named as follows:

```
SORT0M<task number>  
SORT1M<task number>  
SORT2M<task number>  
SORT3M<task number>  
SORT4M<task number>  
SORT5M<task number>  
SORT6M<task number>  
SORT7M<task number>
```

Tag Sorting

In a normal sort operation, records are continually handled throughout the stringing and merging phases. In other words, the sort algorithms are concerned only with the various keys, but the entire record – including the keys – is carried along throughout the sorting process.

For files that have large records and small keys, it might be more efficient to perform a tag sort. In tag sorting, only the keys and the address of each record are handled throughout the sorting process.

In performing a tag sort, you must provide an input procedure that filters the incoming records, extracting the sort keys and appending the disk address of the location of the entire record. The keys are arranged by the input procedure in a contiguous string. This tag is then submitted to the stringing phase. This structure eliminates the need to accumulate the keys from various fields in the record whenever a comparison is required.

During the final merging phase, the output procedure uses the address portion of the tag to retrieve the records in the correct sequence.

The input file is read twice: once by the input procedure (in a serial fashion) for the stringing phase, and once again by the output procedure (in a random fashion) for the final merge phase. Tag sorting requires less disk and memory space to complete the job. Retrieving the output records is the most time-consuming factor and is highly data-dependent.

If a sort program is heavily used, both entire-record and tag sorts should be tested to determine which sort operation produces the best results.

Figure 12–1 shows the extraction of one or more keys from the incoming record and the building of the tag.

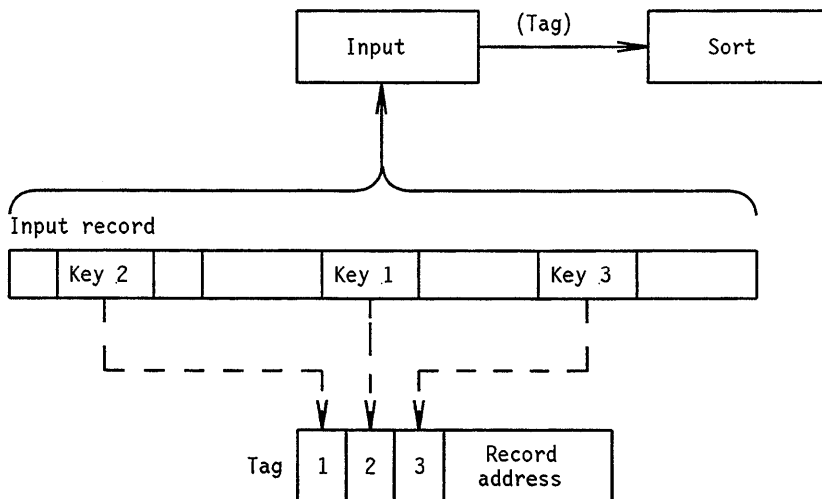


Figure 12-1. Creating a Tag

Figure 12-2 shows a functional diagram of a tag sort that has one or more nondisk input files. The input file must be copied in its entirety to disk.

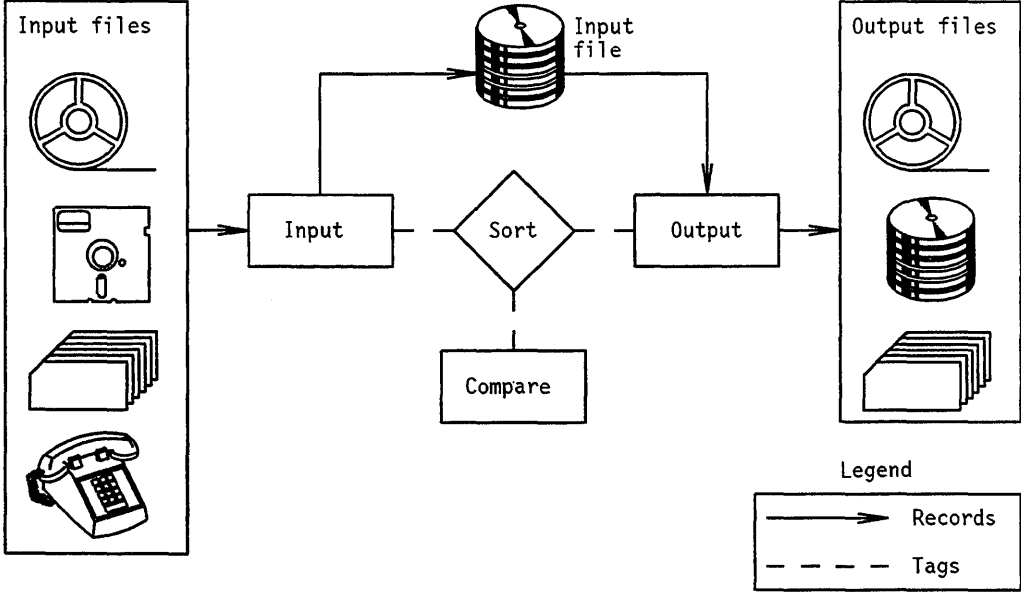


Figure 12-2. Tag Sort, Nondisk Input File

Figure 12-3 shows the functional diagram of a tag sort that has a disk file as its input medium.



Figure 12-3. Tag Sort, Disk Input File

Restart Capability

Restart capability enables SORT to resume processing at the most recent control point following the discontinuation of the program.

Restart capability can be excluded from SORT by setting the RESTART option to FALSE in the SORT portion of the MCP symbolic and then recompiling the MCP. Omitting the code associated with RESTART/ERROR recovery results in sorts that run between 2 and 15 percent faster, with an average improvement of six to ten percent. Disk and ITD sorts are the only sorts measurably improved by this omission.

Operating SORT in restart mode provides the necessary restarting information for the sort but requires certain program inputs that are defined in the following text. The program must provide logic to restore and maintain stack variables, arrays, files, and pointers that are defined for and by the program. In other words, the program must provide the means to restore everything necessary for it to continue from the point of discontinuation. This capability can be simple or complex and is entirely program-dependent. The program cannot use the CHECKPOINT/RERUN function when a sort is executing.

Restart capability is implemented for disk sorts only; however, a partially restartable ITD sort is also possible. When SORT is not in the tape phase of an ITD sort, it functions as a disk sort. After the data is written from disk to tape during an ITD sort, SORT cannot be restarted.

If SORT is started as a disk-only restartable sort with insufficient disk space provided to accomplish the sort, SORT terminates with SORT ERROR #4 or #5. After error termination, SORT can be restarted as an ITD sort by indicating a RESTART and specifying the number of tapes desired. Any other kind of restart is impossible under this condition.

If a restartable sort terminates during the first output of data from disk to tape (possibly as a result of an irrecoverable tape I/O error), SORT can be restarted and the data is written to tape as if no problem had occurred.

When using SORT in restartable mode, file equation should be used to give unique titles to the control file and the work file. Refer to "SORT Files" for more information about these files. Conflicts might arise when two or more sorts use identical file titles for their sort disk files.

When SORT attempts to restart a previously incomplete sort, sort record size and character size of the sort record characters are verified to ensure that continuation is compatible with the previous sort. For ALGOL programs, record size is explicitly specified by the program while character size is 0 (default size is 8). COBOL programs use the SD to determine sort record and character size. When record size or character size does not match the previous sort, error termination occurs.

Other sort parameters — except number of tapes — can be modified. Different values for memory size or disk size are ignored, and the original values are used. However, both values must be valid nonzero values. Different procedures can be specified (for input, output, or comparing) if desired, files can be substituted for input or output procedures, and input or output procedures can be specified for files. The program requesting the restart need not be the originating program. Because SORT can only attempt to meet your request and cannot determine appropriateness of requests, you must make sure that you get the desired results.

RESTART Parameter Values

SORT inspects various bits of the restart parameter to determine the action it is to take. You must supply proper file title attributes for the two disk work files if these files were previously file-equated. Individual bits and combinations of bits can be set by the program to control SORT. The bits and their meanings are as follows:

Bit 0

- ON

The program is restarting a previous sort operation. SORT tries to open its two disk files and obtain restart information. After successfully obtaining this information, SORT continues from the last known restart point.

- OFF

SORT is starting from the beginning. If the sort operation is restartable and previous sort files with identical titles exist, these sort files are removed and replaced by new sort files.

Bit 1

- ON

The program is requesting a restartable sort operation. SORT saves its two internal files and can be restarted on program request. If bit 2 is ON, bit 1 is set by default.

- OFF

A normal sort operation is requested, and no sort files are saved (unless bit 2 is on, which sets bit 1 by default).

Bit 2

- ON

The program is requesting a restartable sort and desires extensive recovery from I/O errors. With this option set, if I/O errors occur while accessing either of the two sort files, SORT attempts to backtrack and remerge strings as necessary. To use this option, the program must provide at least three times as much disk space as required to contain the input data. When less space is provided, SORT emits the message "CHANGE TO RESTARTABLE ONLY MODE" and continues the sort without further capability to backtrack.

- OFF

Recovery from internal errors is not requested.

Bit 3

This bit has meaning only if a restartable sort operation is requested. This option controls SORT during the stringing phase as your input is being read by SORT. Use of this bit determines how SORT restarts—when a restart is requested—if the restart occurs while SORT is in the stringing phase.

- **ON**

The SORT is to restart at the beginning of your input. This restart is the equivalent of starting an entirely new sort. In case the restarted sort operation had passed from the stringing phase into the merge phase, it continues from the merge phase. This bit can be set during a restart even if it was not initially set. Once set, it cannot be reset by subsequent restarts.
- **OFF**

The SORT is to restart at the last restart point that occurred during the stringing phase. If SORT is still in the stringing phase, it skips over the records already processed and continues from the last restart point. Refer to “Restarting During Stringing Phase” for more information about this process. If SORT is in the merging phase, it continues from the last merge phase restart point. Not setting the bit is normally less efficient than setting the bit because more strings are created during the stringing phase.

Bit 4

This bit is reserved for expansion and is not currently used by SORT.

Parameter Values that Combine Bits 0 through 4

When a program is initially starting a sort operation and desires restart ability the restart value should be set as follows:

- Decimal 2 (bit 1 ON) if a restartable sort operation that is capable of restarting at any point during the stringing or merge phase is desired.
- Decimal 10 (bits 1 and 3 ON) if a restartable sort operation that can restart at any point during the merging phase but only at the beginning of the stringing phase is desired.
- Decimal 4 or 6 (bit 2 ON or bits 1 and 2 ON) if a restartable sort operation that can attempt extensive recovery from internal sort I/O errors and can restart at any point during the stringing or merge phase is desired.
- Decimal 12 (bits 2 and 3 ON) or Decimal 14 (bits 1, 2, and 3 ON) if a restartable sort operation that can attempt extensive recovery from internal sort I/O errors and can restart at any point during the merging phase but only at the beginning of the stringing phase is desired.
- Decimal 1, 3, 5, or 7—significant bits are bit 0 ON and bit 3 OFF—if a previously incomplete sort operation that can be restarted is desired. The prior incomplete sort must have been capable of restart, and the two sort disk files must be present. A restart is attempted using the values obtained from the sort files. The previous setting of bit 3 controls the sort if it is restarted during the stringing phase. The previous values of bits 1 and 2 are used.

- Decimal 9, 11, 13, or 15 (significant bits are bits 0 and 3 ON) if a restart of a previously incomplete sort is desired, and if a restart from the beginning of input is desired during the stringing phase. The prior incomplete sort must have been capable of restart, and the two sort disk files must be present. A restart is attempted using the values obtained from the sort files. Bit 3 is set and remains set through all subsequent restarts. Bits 1 and 2 take on their previous values.
- Decimal 0 or 8 – no bits ON or bit 3 ON – causes the sort operation to perform a normal sort with no restart capability.

Restarting during Stringing Phase

Restarting during the stringing phase – while SORT is still reading input records – requires special consideration. If SORT has been passed a file, either a seek is performed or records are read until the desired restart point is reached. However, an input procedure presents a different problem because the program must find the proper restart record. To be able to locate the proper restart record, SORT places values in the first word of the array passed to the input procedure.

The values are negative or positive integers in binary form or 0 to indicate that nothing special is happening. A positive integer is placed in the first word (word 0) to tell the input procedure the relative number of the next record desired by SORT. For example, if SORT has previously processed and saved 99 records, it requests record number 100. A positive nonzero integer occurs in the first word only once, on the first call to the input procedure.

SORT places a negative nonzero integer in the first word to inform the input procedure that SORT has just established a restart point. The absolute value of the number returned represents the number of records saved by SORT. This information can be used by the program to establish its own separate restart points.

Error Recovery

SORT has extensive internal error recovery ability for irrecoverable I/O errors that occur when a sort disk file is accessed. If such an error occurs, SORT issues the following message:

```
SORT ERROR #nn: IRRECOVERABLE I/O ERROR ON <sort file name> FILE
```

The *nn* is a number from 10 through 14, and the sort file name is the name of the internal sort file that is in error. SORT then waits for an OK or DS system command. If the operator requests error recovery by entering *OK*, SORT attempts to recover. If the operator responds by entering *DS*, SORT is terminated.

The sort disk files under discussion are the work files, which contain the data being sorted, and the control file, which contains control information for SORT. These two types of files are described under “Sort Files.”

I/O error recovery depends on the file and the kind of error encountered. The degree of recovery possible is always dependent on the request that SORT carry out error

recovery for the program. Error recovery is never attempted beyond one level of recovery. If recovery is attempted while recovering from a prior error, SORT terminates. The primary areas of error recovery are described in the following paragraphs.

Control File Input Errors

If an error occurs while a record is being read, an attempt is made to obtain the record by rereading the error record several times. If the error record is unreadable and error recovery is not requested, SORT terminates. If error recovery is requested, SORT attempts to read its duplicate copy of the error record.

Control File Output Errors

If an error occurs while a record is being written, an attempt is made to successfully write the error record. If writing is not successful after several retries and error recovery is not requested, SORT terminates. If error recovery is requested, SORT marks as a bad disk the row of the disk containing the error record. SORT retains the other copy of that row for subsequent use. If possible, SORT continues in full error recovery mode; otherwise, SORT displays "SORT ERROR #31" and continues in error recovery mode for the work file. Further error recovery for the control file is no longer possible. In either case, if SORT is unable to write the error record after the bad row has been marked, SORT terminates. Only one disk row is marked as bad disk on an error so that it is not possible to get into a loop and mark large quantities of disk as bad.

User output file error recovery is not dependent on the system option RESTART.

Work File Input Errors

If an error occurs when a record is being read, an attempt is made to obtain the record by rereading the error record several times. If the error record is unreadable and error recovery is not requested, SORT terminates. If error recovery is requested and the data has been duplicated, an attempt is made to read the duplicate copy. If the error record was written by the merge phase and no duplicate copy exists, SORT attempts to re-create the string of information containing the error record. Before backtracking to the previous merge, SORT writes and reads a test record in the error record location. If the test is unsuccessful, SORT marks as bad disk the row of the disk containing the error record location. After testing and possibly marking the affected areas of the disk, SORT backtracks to the desired point for restarting the merge phase. At most, one row of the disk is marked as bad disk for each occurrence of an input error for the work file.

Work File Output Errors

If an error occurs when a record is being written, an attempt is made to successfully write the error record. If writing is not successful after several retries and error recovery is not requested, SORT terminates. If error recovery is requested, SORT marks the row of the disk containing the error record as bad disk. If possible, SORT continues in full error recovery mode; otherwise, SORT displays "SORT ERROR #32"

and continues with error recovery reset. If SORT is in the stringing phase, an attempt is made to write the error record and if the attempt is unsuccessful, SORT terminates. If SORT is in the merge phase, it backtracks to the desired point of restarting the merge phase. At most, one row of disk is marked for each occurrence of an output error for the work file.

User Output File Errors

When the program has given SORT an output file—as opposed to an output procedure—SORT closes and purges the output file and restarts the output from the first output record. If the output file is a disk file and insufficient space was allocated to contain the data, SORT either sets the FLEXIBLE attribute before restarting the output or, if setting the FLEXIBLE attribute is not possible, terminates with SORT ERROR #8. Output error recovery is not dependent on program request for error recovery.

Work File Input Errors during User Output

When your output is a file, the file is closed and purged, and SORT attempts to remerge the desired string. If the output is a procedure and error recovery is specified, SORT repositions itself to remerge the desired string and subsequently terminates with SORT ERROR #19. When SORT is restarted, it remerges the desired string and starts user output with the first output record.

Using SORT in Various Languages

Using SORT in COBOL or COBOL74

In COBOL or COBOL74, the SORT intrinsic is invoked by the SORT statement. The sort work file, input procedure or file, output procedure or file, sort keys, and sorting criteria are specified in the SORT statement. The sort work file must be described in a sort-merge file-description entry in the DATA DIVISION and in a SELECT clause in the FILE-CONTROL paragraph. The SELECT clause determines the sort mode to be used.

Memory size, disk size, and restart information are optional parameters. (For a complete description of the COBOL or COBOL74 SORT statement, refer to the *A Series COBOL ANSI-68 Programming Reference Manual* or the *A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation*.)

SORT Input/Output (I/O) Procedure Logic Flow

The following logic chart shows the interaction of the SORT utility with a COBOL procedural sort operation:

- Sort 1. Begin SORT initialization phase.
- Sort 2. Open sort work files.

SORT Utility

- Sort 3. Go to beginning of your INPUT PROCEDURE.
 - IP1. = Open input file.
 - IP2. = Read input file record (at end, go to IP6).
 - IP3. = If record is to be used, place in record area of sort file; otherwise, go to IP2.
 - IP4. = Release sort file record (transfer to Sort 4).
- Sort 4. Place the released record in sorting process.
- Sort 5. Execute internal sorting, creating strings on sort work files.
- Sort 6. Return to INPUT PROCEDURE at IP5.
 - IP5. = Execute *using* logic; then go to IP2.
 - IP6. = Execute AT END logic, including close of input file (transfer to Sort 7).
- Sort 7. Complete SORT stringing of all input records.
- Sort 8. Begin merge phase of SORT.
- Sort 9. Merge all strings on sort work files until one string remains.
- Sort 10. Go to beginning of OUTPUT PROCEDURE.
 - OP1. = Open output file (transfer to Sort 11).
- Sort 11. Execute final internal merging operation.
- Sort 12. = Pass merged record to OUTPUT PROCEDURE at OP2.
 - OP2. = Return sort file record to user record area (at end, go to OP4).
 - OP3. = Execute user logic (transfer to Sort 11).
 - OP4. = Execute AT END logic, including close of output file (transfer to sort 13).
- Sort 13. = Close all sort work files.
- Sort 14. Exit from SORT.

COBOL SORT Example

The following is an example of a COBOL disk sort, including the actual input and output files:

```
IDENTIFICATION DIVISION.  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER.  B5900.  
OBJECT-COMPUTER.  B5900  
    DISK SIZE 200000 WORDS  
    MEMORY SIZE 30000 WORDS.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT NEWTRANS ASSIGN TO CARD-READER.  
    SELECT CREDITFILE ASSIGN TO 50000 DISK.
```

```
SELECT DEBITFILE ASSIGN TO 5000 DISK.
SELECT SORTFILE ASSIGN TO SORT DISK.
DATA DIVISION.
FILE SECTION.
FD NEWTRANS          VALUE OF ID "TRANSACTIONS".
  Ø1 NEWTR SZ 80.
FD CREDITFILE        VALUE OF ID "NEW"/"CREDITS"
  BLOCK CONTAINS 15 RECORDS.
  Ø1 CR-REC SZ 80.
FD DEBITFILE          VALUE OF ID "NEW"/"DEBITS"
  BLOCK CONTAINS 15 RECORDS.
  Ø1 DR-REC SZ 80.
SD SORTFILE.
  Ø1 SRT.
    Ø3 CODE-KEY PIC 99.
    Ø3 ACCOUNT-KEY PIC 9(10).
    Ø3 DATE-KEY PIC 9(6).
    Ø3 FILLER PIC X(62).
PROCEDURE DIVISION.
SORTIT SECTION.
SRTRN.
  SORT SORTFILE ON DESCENDING KEY CODE-KEY
  ASCENDING KEY ACCOUNT-KEY DATE-KEY
  USING NEWTRANS
  OUTPUT PROCEDURE RECORDS-OUT.
ENDIT. STOP RUN.
RECORDS-OUT SECTION.
CREDITS-OUT.
  OPEN OUTPUT CREDITFILE DEBITFILE.
LOOP-CR.
  RETURN SORTFILE AT END GO TO XIT.
  IF CODE-KEY > 49
    WRITE CR-REC FROM SRT INVALID KEY GO TO IVK
    ELSE GO TO LOOP-CR.
LOOP-DR.
  WRITE DR-REC FROM SRT INVALID KEY GO TO IVK.
  RETURN SORTFILE AT END GO TO XIT
  ELSE GO TO LOOP-DR.
XIT.
  CLOSE CREDITFILE LOCK DEBITFILE LOCK.
  GO TO ENDIT.
IVK.
  DISPLAY "ERROR TERMINATION".
  DISPLAY CODE-KEY ACCOUNT-KEY.
ENDIT. EXIT.
```

Input File to be Sorted:

121000000233040770
121000000233040970
121000000233041270
031200000042041270
031200000042041370
031200000042041670
031200000042040970
55100000012050170
55100000012052370
55100000012051470
55100000012050570
47100000012050570
47100000012052270
720900000243060270
720900000243061970
720900000243062170
710900000243062170
710900000243062670
12400000035062670
90000000017070370

Sorted Output Files:

NEW/CREDITS output file

90000000017070370
720900000243060270
720900000243061970
720900000243062170
710900000243062170
710900000243062670
55100000012050170
55100000012050570
55100000012051470
55100000012052370

NEW/DEBITS output file

47100000012050570
47100000012052270
121000000233040770
121000000233040970
121000000233041270
12400000035062670
031200000042040970
031200000042041270
031200000042041370
031200000042041670

Using SORT in ALGOL

In ALGOL, the SORT intrinsic is invoked by the SORT statement. The input option procedure or file, output option procedure or file, number of tapes, compare procedure, and record length are required parameters in the ALGOL syntax. The sort keys and sorting criteria are determined solely by the compare procedure.

Memory size and disk or pack size are optional parameters. If restart specifications are used, they appear outside the parameter list, in brackets. For a complete description of the ALGOL SORT statement syntax, refer to the *A Series ALGOL Programming Reference Manual, Volume 1: Basic Implementation*.

ALGOL SORT Example

```

BEGIN
COMMENT
THIS IS AN EXAMPLE OF A SIMPLE, ASCENDING SORT. THE PROGRAM
SORTS A FILE WITH NAMES IN THE FIRST 12 COLUMNS AND WRITES
THE SORTED FILE TO A PRINTER FILE. IF COLUMN 13 CONTAINS AN
ASTERISK ("*"), THE RECORD IS DISCARDED AND IS NOT SORTED.
THIS PROGRAM USES A PROCEDURE AS THE INPUT OPTION AND A FILE
AS THE OUTPUT OPTION. (FOR SIMPLICITY, ANY EXCEPTION
OCCURRING ON THE INPUT FILE IS ASSUMED TO BE END-OF-FILE.) ;

FILE      INFILE(KIND=DISK,DEPENDENTSPECS),
          OUTFILE(KIND=PRINTER, MAXRECSIZE=72);

DEFINE    SKIPCODE = "*"#,
          SKIPFIELD = POINTER(R[2],8)#;

BOOLEAN PROCEDURE INPROC(R);
ARRAY R[0];
BEGIN
  BOOLEAN RESULT;
  DO
    RESULT := READ(INFILE,12,R[*])
  UNTIL RESULT OR SKIPFIELD NEQ SKIPCODE;
  INPROC := RESULT;
END INPROC;

BOOLEAN PROCEDURE COMPARE(R1,R2);
ARRAY R1, R2 [0];
BEGIN
  COMPARE := POINTER(R1,8) LSS POINTER(R2,8) FOR 12;
END COMPARE;

SORT(OUTFILE,INPROC,0,COMPARE,12);
END.

```

ALGOL Example Files

Input File to be Sorted

```
GLENN
JACK
EARL
ROLLIE
STEVE
DAVE
JOEL
HARRY
HANSEL      *
SHERRY
RICHARD
DICK
BILL
GRETEL      *
DON
JIM
CAROLYN
```

Sorted Output File

```
BILL
CAROLYN
DAVE
DICK
DON
EARL
GLENN
HARRY
JACK
JIM
JOEL
RICHARD
ROLLIE
SHERRY
STEVE
```

Using SORT in PL/I

In PL/I, the SORT intrinsic is invoked by the SORT statement. The sort work file, keys, and sorting criteria are specified in the statement.

The input option procedure or file, output option procedure or file, number of tapes, coresize, disk size, and pack size are optional parameters and can be specified in the statement. For a complete description of the PL/I SORT statement, refer to the *A Series PL/I Reference Manual*.

PL/I SORT Example

```
SORTEXAMPLE: PROC; /* OPTIONS (MAIN) */

DCL AA FILE INPUT RECORD ENV (KIND='READER', MAXRECSIZE=80);

DCL BB FILE OUTPUT RECORD ENV (KIND='PRINTER',MAXRECSIZE=132);

DCL 1 COURSES,
      2 DEPT                CHAR (5),
      2 COURSENAME          CHAR (20),
      2 INSTRUCTOR          CHAR (10),
      2 REQUIRED              CHAR (1);

SORTIN: PROC(A) RETURNS (BIT (1)) OPTIONS (SORTINPUT);
      DCL A CHAR (*);
      ON ENDFILE (AA) GO TO EOF;

LOOP: READ FILE (AA) INTO (COURSES);
      IF REQUIRED = '*' THEN RETURN ('0'B);
      ELSE GO TO LOOP;

EOF: RETURN ('1'B);
      END SORTIN;

SORT COURSES ON
      ASCENDING KEY (COURSES.DEPT, COURSES.COURSENAME,
      COURSES.INSTRUCTOR)
      INPUT (SORTIN)
      GIVING FILE (BB);

END SORTEXAMPLE;
```


SORT Utility

PL/I Example Files

Input File to be Sorted

MATH	TRIGONOMETRY	LAMBERT	*
HIST	AMERICAN HISTORY	JERONIMO	*
MUS	MIXED CHORUS	PAINTER	
ENG	ELECTRONICS	SHERIDAN	*
MATH	CALCULUS I	GUILFORD	*
ENGL	READING COMP	WOODS	*
MATH	CALCULUS II	GUILFORD	*
MATH	STATISTICS	GALLOP	
MATH	MATRIX THEORY	WAIHAU	*
HIST	WESTERN CIV	GREENLEAF	
MUS	MUSIC APPREC	PAINTER	*
ENG	SURVEYING	SHERIDAN	*
ACCT	ACCOUNTING I	BLOCK	*
BUS	BUSINESS LAW	BAILEY	*
MATH	ALGEBRA	MULBERRY	*
MUS	INSTRUMENTAL MUSIC	BLAIR	*
ACCT	ACCOUNTING II	BLOCK	*
MATH	LINEAR ALGEBRA	GUILFORD	
MATH	CALCULUS I	AMSDALE	*
ACCT	COST ACCOUNTING	BLOCK	*
ACCT	ACCOUNTING I	ANOLA	*
HIST	AMERICAN HISTORY	LIVERMORE	*

Sorted Output File

ACCT	ACCOUNTING I	ANOLA	*
ACCT	ACCOUNTING I	BLOCK	*
ACCT	ACCOUNTING II	BLOCK	*
ACCT	COST ACCOUNTING	BLOCK	*
BUS	BUSINESS LAW	BAILEY	*
ENG	ELECTRONICS	SHERIDAN	*
ENG	SURVEYING	SHERIDAN	*
ENGL	READING COMP	WOODS	*
HIST	AMERICAN HISTORY	JERONIMO	*
HIST	AMERICAN HISTORY	LIVERMORE	*
MATH	ALGEBRA	MULBERRY	*
MATH	CALCULUS I	AMSDALE	*
MATH	CALCULUS I	GUILFORD	*
MATH	CALCULUS II	GUILFORD	*
MATH	MATRIX THEORY	WAIHAU	*
MATH	TRIGONOMETRY	LAMBERT	*
MUS	INSTRUMENTAL MUSIC	BLAIR	*
MUS	MUSIC APPREC.	PAINTER	*

Using a Procedural Interface for SORT

The SORT/MERGE procedural interface provides a means for those languages that do not support a SORT or MERGE statement to invoke the SORT utility in the MCP. This is done through library calls to procedures made available in the system library GENERAL SUPPORT. For a program to invoke these library entrypoints, the program must be in a language that supports the type of parameters described for these entrypoints.

SORTFILES

This procedure sorts an input file into a specified output file. The declaration of this procedure in GENERAL SUPPORT is as follows:

```
PROCEDURE SORTFILES (INFO_STRING,
                    TRANS_TABLE,
                    COMPARE_PROC,
                    OUTPUT_FILE,
                    INPUT_FILE);
EBCDIC ARRAY INFO_STRING [*];
ARRAY TRANS_TABLE [*];
BOOLEAN PROCEDURE COMPARE_PROC (A, B);
    ARRAY A, B[*]; FORMAL;
FILE INPUT_FILE, OUTPUT_FILE;
```

The following text describes the input attributes for the procedure:

Attribute	Explanation
INFO_STRING	A string in which the user can describe some additional parameters for the sort. The format of this string is described following this table.
TRANS_TABLE	An optional array containing a translate table that identifies the alternate collating sequence to use for the sort. Translation of keys using the collating sequence is performed only on alphanumeric key types. This parameter is used only if the COLLSEQ clause is present in the INFO_STRING parameter and the user compare procedure is not to be used. The TTABLE routine should have been used to build the translate table prior to passing this array for use by sort. The TTABLE routine is described under "TTABLE" later in this section. If TRANS_TABLE is used, the size of this array should be at least 64 words in length.
COMPARE_PROC	A reference to the user's compare procedure. The compare procedure must be coded as a Boolean function with two array parameters. The function is called every time two records are to be compared. This procedure is called by the sort only if an indication to use it is present in the info string (COMP). The alternative to using a compare procedure is to include the key description in the KEY clause of the INFO_STRING parameter.
OUTPUT_FILE	The file to which the output from the sort is written. If the file is a disk file for which the file attribute SAVEFACTOR has a nonzero value, it is closed and locked after the sort. The output file must not be open when it is passed to SORTFILES by the program.

continued

SORT Utility

continued

Attribute	Explanation
INPUT_FILE	The input file to be used during the sort. The input file must not be open when it is passed to SORTFILES by the program.

The **INFO_STRING** parameter provides the means for the user to specify additional parameters for the sort or merge. This parameter is specified in the form of specific clauses. The clauses must be separated by blanks or commas, and the entire string must be terminated by a period. The only required clauses are **RSZ**, and **KEY** or **COMP**. The following are the clauses that can be specified in the info string:

RSZ=record length	Record length in bytes.
KEY=key-desc	Describes a single key.
KEY=(d,...,d)	<p>Describes several keys. The first key listed is the primary key. Subsequent keys are used to further sort or merge any records that were identical in terms of the previous key. Each key description takes the form: position/length/sequence/type.</p> <ul style="list-style-type: none">● Position – Position beginning at 1, in bytes, unless hex type. If type is hex, position must be specified in hex digits● Length – In bytes, unless hex type. If type is hex, length must be specified in hex digits. (default 1)● Sequence – A (ascending) or D (descending). Default is A.● Type – Character specifying format of the key. Each format is byte-or hex-oriented. This orientation affects position and length. <p>The types are:</p> <ul style="list-style-type: none">– N - display numeric (default, byte-oriented)– S - alphabetic/alphanumeric (byte-oriented)– B - integer binary (byte-oriented)– E - real, binary with exponent (byte-oriented)– X - double, extended (byte-oriented)– V - left overpunched sign (byte-oriented)– Q - right overpunched sign (byte-oriented)– R - left separate sign (byte-oriented)– T - right separate sign (byte-oriented)– U - unsigned (hex-oriented)– L - left signed (hex-oriented)– P - right signed (hex-oriented)
COMP	Appears only if a comparison routine is used. Either COMP or KEY must appear, but not both, meaning that if there is no comparison routine, then KEY must be used.
COLLSEQ	Indicates an alternate collating sequence is to be used.

continued

continued

DUPL	Indicates that duplicate keys should be ordered based on order of appearance (for SORT only).
CORE=size	Core size, in words (default 12000).
WORKFILE=type	Describes type of a single workfile. WORKFILE can be abbreviated to WORK.
WORKFILE=(t,t)	<p>Describes disk and tape workfiles. The workfile type can be:</p> <ul style="list-style-type: none"> ● DISK/size. Indicates disk workfile. Size is number of words Default size is 600,000 words. ● TAPE/#tapes. Indicates tape workfiles. #tapes indicates number of tapes between 3 and 8. Default number is 3. <p>If no workfiles are specified, a memory only sort is performed.</p>
CCSVERSION=name	Specifies coded character set version
CCSVERSION	Uses default version if no name is specified.
RESTART=r-val	<p>Describes restart functions. The default is 0 (zero).</p> <p>The allowable values for r-val are:</p> <ul style="list-style-type: none"> ● 0 - No restart capability. ● 1 - Restart previous sort. The prior uncompleted sort must have been capable of restart. ● 2 - Allow restartable sort. ● 4 or 6 - Allow restartable sort and enable extensive error recovery from I/O errors. ● 9 - Restart previous sort if all input has been received. The prior uncompleted sort must have been capable of restart. ● 10 - Allow restartable sort after all input received. ● 12 or 14 - Options 4 and 10.

SORTPROCS

This procedure performs the sort on a set of records where input is provided from a procedure in the user's program and output is sent to a procedure in the user's program. SORTPROCS should be used when there are files to be sorted which are variable in length. The declaration of this procedure in GENERALSUPPORT is as follows:

```

PROCEDURE SORTPROCS (INFO_STRING,
                    TRANS_TABLE,
                    COMPARE_PROC,
                    OUTPUT_PROC,
                    INPUT_PROC);
EBCDIC ARRAY INFO_STRING [*];
ARRAY TRANS_TABLE [*];
BOOLEAN PROCEDURE COMPARE_PROC (A, B);
    ARRAY A, B[*]; FORMAL;
PROCEDURE OUTPUT_PROC (B, A);
    VALUE B;
    BOOLEAN B;
    ARRAY A[*]; FORMAL;
BOOLEAN PROCEDURE INPUT_PROC (A);
    ARRAY A[*]; FORMAL;

```

The following text describes the input attributes for the procedure:

Attributes	Description
INFO_STRING	A string in which the user can describe some additional parameters for the sort. (Refer to the description of SORTFILES for a complete description of this string).
TRANS_TABLE	An optional array containing a translate table that identifies the alternate collating sequence to use for the sort. Translation of keys using the collating sequence is performed only on alphanumeric key types. This parameter is used only if the COLLSEQ clause is present in the INFO_STRING parameter and the user compare procedure is not to be used. The TTABLE routine should have been used to build the translate table prior to passing this array for use by the sort. The TTABLE routine is described under "TTABLE" later in this section. If TRANS_TABLE is used, the size of this array should be at least 64 words in length.
COMPARE_PROC	A reference to the user's compare procedure. The compare procedure must be coded as a Boolean function with two array parameters. The function is called every time two records are to be compared. This procedure is called by the sort only if an indication to use it is present in the info string. The alternative to using a compare procedure is to include the key description in the KEY clause of the INFO_STRING parameter.

continued

continued

Attributes	Description
OUTPUT_PROC	The procedure in the user's program that receives each output record from the sort. This procedure takes two parameters. The first parameter is a Boolean parameter, and the second parameter is an array parameter which contains the output record. The array parameter can be any type which is compatible with an ALGOL Real array. The Boolean parameter contains FALSE as long as the second parameter contains a valid sorted record. When all the records have been returned, the first parameter is TRUE and the second parameter must not be accessed.
INPUT_PROC	The function which provides the input records to be used during the sort. This is a Boolean function with one array parameter in which the record to be used during the sort is provided. The function is to return a TRUE result to indicate the end of input data.

MERGEFILES

This procedure merges from two to eight files into one file. The declaration of this procedure in GENERALSUPPORT is as follows:

```

PROCEDURE MERGEFILES (INFO_STRING,
                      NUM_FILES,
                      TRANS_TABLE,
                      COMPARE_PROC,
                      OUTPUT_FILE,
                      INPUT_FILE1,
                      INPUT_FILE2,
                      INPUT_FILE3,
                      INPUT_FILE4,
                      INPUT_FILE5,
                      INPUT_FILE6,
                      INPUT_FILE7,
                      INPUT_FILE8);
EBCDIC ARRAY INFO_STRING [*];
ARRAY TRANS_TABLE [*];
BOOLEAN PROCEDURE COMPARE_PROC (A, B);
  ARRAY A, B[*]; FORMAL;
FILE OUTPUT_FILE,
  INPUT_FILE1, INPUT_FILE2, INPUT_FILE3, INPUT_FILE4,
  INPUT_FILE5, INPUT_FILE6, INPUT_FILE7, INPUT_FILE8;

```

The following text describes the input attributes for the procedure:

Attributes	Description
INFO_STRING	A string in which the user would describe some additional parameters for the merge (refer to "SORTFILES" earlier in this section for a complete description of this string).
NUM_FILES	Indicates how many of the input files are valid and are to be used during the merge.

continued

SORT Utility

continued

Attributes	Description
TRANS_TABLE	An optional array containing a translate table which identifies the alternate collating sequence to use for the merge. Translation of keys using the collating sequence is only performed on alphanumeric key types. This is used only if an indication to use this is present in the Info string (COLLSEQ) and if the user compare procedure is not to be used. The TTABLE routine (described later in this section) should have been used to build the translate table prior to passing this array for use by the merge. If used, the size of this array should be at least 64 words in length.
COMPARE_PROC	A reference to the user's compare procedure. The compare procedure must be coded as a Boolean function with two array parameters. The function is called every time two records are to be compared. This procedure is only called by the merge if an indication to use it is present in the info string. The alternative to using a compare procedure is to include the key description in the info string.
OUTPUT_FILE	The file to which the output from the merge is written. If the file is a disk file for which the file attribute SAVEFACTOR has a nonzero value, it is closed and locked after the MERGE. The output file must not be open when it is passed to MERGEFILES by the program.
INPUT_FILE1	An input file to be used during the merge. The input files (INPUT_FILE1 - INPUT_FILE8) must not be open when they are passed to MERGEFILES by the program.
INPUT_FILE2	An input file to be used during the merge.
INPUT_FILE3	An input file to be used during the merge. This may be a dummy file if only 2 files are to be merged.
INPUT_FILE4	An input file to be used during the merge. This may be a dummy file if only 3 or less files are to be merged.
INPUT_FILE5	An input file to be used during the merge. This may be a dummy file if only 4 or less files are to be merged.
INPUT_FILE6	An input file to be used during the merge. This may be a dummy file if only 5 or less files are to be merged.
INPUT_FILE7	An input file to be used during the merge. This may be a dummy file if only 6 or less files are to be merged.
INPUT_FILE8	An input file to be used during the merge. This may be a dummy file if only 7 or less files are to be merged.

Note: *If the user wishes to merge only two files, the number two would be entered for the NUM_FILES parameter, the first two input file parameters would be the files that would be merged, and dummy files would be entered for the remaining input files. These files are ignored by the library procedure.*

MERGEPROCS

This procedure uses from two to eight input procedures to perform the merge, giving the results to the output procedure. MERGEPROCS should be used when there are files to be merged which are variable in length. The declaration of this procedure in GENERALSUPPORT is as follows:

```
PROCEDURE MERGEPROCS (INFO_STRING,  
                      NUM_PROCS,  
                      TRANS_TABLE,  
                      COMPARE_PROC,  
                      OUTPUT_PROC,  
                      INPUT_PROC1,  
                      INPUT_PROC2,  
                      INPUT_PROC3,  
                      INPUT_PROC4,  
                      INPUT_PROC5,  
                      INPUT_PROC6,  
                      INPUT_PROC7,  
                      INPUT_PROC8);  
EBCDIC ARRAY INFO_STRING [*];  
ARRAY TRANS_TABLE [*];  
BOOLEAN PROCEDURE COMPARE_PROC (A, B);  
  ARRAY A, B[*]; FORMAL;  
PROCEDURE OUTPUT_PROC (B, A);  
  VALUE B;  
  BOOLEAN B;  
  ARRAY A[*]; FORMAL;  
BOOLEAN PROCEDURE INPUT_PROC1 (A);  
  ARRAY A[*]; FORMAL;  
BOOLEAN PROCEDURE INPUT_PROC2 (A);  
  ARRAY A[*]; FORMAL;  
BOOLEAN PROCEDURE INPUT_PROC3 (A);  
  ARRAY A[*]; FORMAL;  
BOOLEAN PROCEDURE INPUT_PROC4 (A);  
  ARRAY A[*]; FORMAL;  
BOOLEAN PROCEDURE INPUT_PROC5 (A);  
  ARRAY A[*]; FORMAL;  
BOOLEAN PROCEDURE INPUT_PROC6 (A);  
  ARRAY A[*]; FORMAL;  
BOOLEAN PROCEDURE INPUT_PROC7 (A);  
  ARRAY A[*]; FORMAL;  
BOOLEAN PROCEDURE INPUT_PROC8 (A);  
  ARRAY A[*]; FORMAL;
```


The following text describes the input attributes for the procedure:

Attributes	Description
INFO_STRING	A string in which the user can describe some additional parameters for the merge. Refer to "SORTFILES" earlier in this section for a complete description of this string.
NUM_PROCS	Indicates how many of the input procedures are valid and are to be used during the merge.
TRANS_TABLE	An optional array containing a translate table which identifies the alternate collating sequence to use for the merge. Translation of keys using the collating sequence is performed only on alphanumeric key types. This parameter is used only if the COLLSEQ clause is present in the INFO_STRING parameter and the user compare procedure is not to be used. The TTABLE routine should have been used to build the translate table prior to passing this array for use by the merge. the TTABLE routine is described under "TTABLE" later in this section. If TRANS_TABLE is used, the size of this array should be at least 64 words in length.
COMPARE_PROC	A reference to the user's compare procedure. The compare procedure must be coded as a Boolean function with two array parameters. The function is called every time two records are to be compared. This procedure is called by the merge only if an indication to use it is present in the info string. The alternative to using a compare procedure is to include the key description in the KEY clause of the INFO_STRING parameter.
OUTPUT_PROC	The procedure in the user's program that receives each output record from the merge. This procedure takes two parameters. The first parameter is a Boolean parameter, and the second parameter is an array parameter which contains the output record. The Boolean parameter contains FALSE as long as the second parameter contains a valid sorted record. When all the records have been returned, the first parameter is TRUE and the second parameter must not be accessed.
INPUT_PROC1 through INPUT_PROC8	The functions which provide the input records to be used during the merge. These are Boolean functions each with one array parameter in which the record to be used during the merge is provided. The functions are to return a TRUE result to indicate the end of input data. Only INPUT_PROC1 and INPUT_PROC2 are required, the rest of the functions can be dummy functions which are never accessed during the merge.

Note: *If the user wishes to use only two procedures for the merge, the number 2 would be entered for the NUM_PROCS parameter, the first two input procedure parameters would be the procedures that would be used for the merge, and dummy procedures would be entered for the remaining input procedures. Dummy procedures are ignored by the library procedure.*

TTABLE

This routine provides the means for the user to specify an alternate collating sequence to be used for the sort or merge procedural interface. This routine creates a translate table based on the user's specification of a collating sequence. The format for this library procedure declaration is as follows:

```
PROCEDURE TTABLE (IN_ARRAY,  
                  TRANS_TABLE);  
  EBCDIC ARRAY IN_ARRAY [*];  
  ARRAY TRANS_TABLE [*];
```

The following text describes the input attributes for the procedure:

Attributes	Description
IN_ARRAY	<p>The array that contains the user-specified collating sequence. The contents must be in the following form and must contain a terminating period to denote the end of the string:</p> <ul style="list-style-type: none"> ● If specifying a user defined collating sequence then: <pre style="margin-left: 40px;"> [- literal-2] literal-1 [& literal-3 [& literal-4] ...] [[- literal-6]] [literal-5 [& literal-7 [& literal-8] ...]] ... </pre> <ul style="list-style-type: none"> - Where literals may either be character-strings delimited by quotation marks (") or numeric literals in the range of 0 through 255. Numeric literals denote the ordinal number of a character within the native character set. The literals following and preceding the ampersand (&) and hyphen (-) characters must be one character in length. <p style="margin-left: 40px;">Example: "WGF" "R" & 232 90 - 100</p> <ul style="list-style-type: none"> - The order in which the literals appear specifies, in ascending sequence, the ordinal number of the character within the collating sequence being specified. - Any characters within the native collating sequence that are not explicitly specified in the literal phrase assume a position greater than any of the explicitly specified characters in the collating sequence being specified. The relative order within the set of these unspecified characters is unchanged from the native collating sequence. - If the hyphen (-) character is used, the set of contiguous characters in the native character set beginning with the character specified by the value of literal-1 and ending with the character specified by the value of literal-2 is assigned a successive ascending position in the collating sequence being specified. A given hyphen (-) specification can specify characters of the native character set in either ascending or descending sequence. - If the ampersand (&) character is specified, the characters of the native character set specified by the value of literal-1, literal-3, literal-4, and so on are assigned to the same position in the collating sequence being specified. ● If specifying the ASCII collating sequence then the IN_ARRAY parameter must contain only the following: <pre style="margin-left: 40px;">ASCII.</pre> <p style="margin-left: 40px;">ASCII is not delimited by double quotes, must be in uppercase, and must end in a period (.).</p>
TRANS_TABLE	<p>The array that receives the output from the routine and contains the resulting translate table. This array must be at least 64 words in length.</p>

The translate table resulting from this routine can be passed in the SORTPROCS, SORTFILES, MERGEPROCS, or MERGEFILES routines with a COLLSEQ clause in the INFO_STRING parameter specifying that this table is to be used.

Sample Programs

SortProcs Example Program

```

PROGRAM P;
TYPE
  Info_string_type = PACKED ARRAY [1..500] OF CHAR;
  File_Buff        = PACKED ARRAY [1..35] OF CHAR;
  Array_type       = RECORD
    CASE BOOLEAN OF
      TRUE : (F_Arry : File_Buff);
      FALSE: (R_Arry : PACKED ARRAY [1..6] OF REAL;
    END;
  Trans_table_type = ARRAY [1..64] OF REAL;

LIBRARY Gensupport (LIBACCESS = BYFUNCTION,
                   FUNCTIONNAME = 'GENERSUPPORT');

PROCEDURE SortProcs (VAR Info_string : Info_String_type;
                    VAR Trans_table : Trans_table_type;
                    FUNCTION Compare_proc
                      (VAR Ary1 : Array_type;
                     VAR Ary2 : Array_type):
                      BOOLEAN;

                    PROCEDURE Output_proc
                      (Done      : BOOLEAN;
                     VAR Ary   : Array_type);
                    FUNCTION Input_proc
                      (VAR Ary : Array_type):
                      BOOLEAN);

  Gensupport;

VAR
  If_string      : Info_string_type;
  T_table        : Trans_table_type;
  In_file        : FILE OF File_buff;
  Out_file       : FILE OF File_buff;
  I_Status       : INTEGER;

FUNCTION Dummy_Comp_proc (VAR Ary1 : Array_type;
                        VAR Ary2 : Array_type) : BOOLEAN;
BEGIN
  If Ary1.R.Arry[1] > Ary2.R.Arry[1] then
    Dummy_Comp_Proc := True
  else
    Dummy_Comp_proc := False;
END;

```

SORT Utility

```
PROCEDURE Output_proc (Done : BOOLEAN;
                      VAR Ary : Array_type);
BEGIN
  IF Done THEN
    CLOSE (Out_file, SAVE)
  ELSE
    BEGIN
      Out_file@ := Ary.F_array;
      PUT (Out_file);
    END;
  END;

FUNCTION Input_Proc (VAR Ary : Array_type): BOOLEAN;
BEGIN
  IF I_Status = IORES(EOF) THEN
    Input_proc := TRUE
  ELSE
    BEGIN
      I_Status := GET (In_file);
      Ary.F_array := In_file@;
      Input_proc := FALSE;
    END;
  END;

BEGIN
  :
  (* set file attributes for input and output file *)
  :
  If_string := 'RSZ = 35, KEY = 10/3/A/S.';
  OPEN (In_file);
  OPEN (Out_file, NEW);
  SortProcs (If_string, T_table, Dummy_comp_proc, Output_proc,
            Input_proc);
  :
END.
```

MergeFiles Example Program

```
PROGRAM P;
TYPE
  Info_string_type = PACKED ARRAY [1..500] OF CHAR;
  File_Buff        = PACKED ARRAY [1..35] OF CHAR;
  Array_type       = RECORD
    CASE BOOLEAN OF
      TRUE : (F_Array : File_Buff);
      FALSE: (R_Array : PACKED ARRAY [1..6] OF REAL;
    END;
  Trans_table_type = ARRAY [0..64] OF REAL;
  File_Type        = SYSTEMFILE (CHAR);
```

```

LIBRARY Gensupport (LIBACCESS = BYFUNCTION,
                  FUNCTIONNAME = 'GENERSUPPORT');

PROCEDURE MergeFiles (VAR Info_string : Info_String_type;
                    Num_files      : INTEGER;
                    VAR Trans_table : Trans_table_type;
                    FUNCTION Compare_proc
                        (VAR Ary1 : Array_type;
                         VAR Ary2 : Array_type) :
                        BOOLEAN;
                    VAR Output_file : File_type;
                    VAR In_file1   : File_type;
                    VAR In_file2   : File_type;
                    VAR In_file3   : File_type;
                    VAR In_file4   : File_type;
                    VAR In_file5   : File_type;
                    VAR In_file6   : File_type;
                    VAR In_file7   : File_type;
                    VAR In_file8   : File_type);
    Gensupport;

VAR
    If_string      : Info_string_type;
    T_table        : Trans_table_type;
    In_file1       : File_type;
    In_file2       : File_type;
    Out_file       : File_type;
    Dummy_file     : File_type;

FUNCTION Comp_proc (VAR Ary1 : Array_type;
                  VAR Ary2 : Array_type) : BOOLEAN;
    VAR
        S1 : STRING(35);
        S2 : STRING(35);
    BEGIN
        S1 := Ary1.F_array;
        S2 := Ary2.F_array;
        IF S1 <= S2 THEN
            Comp_proc := TRUE
        ELSE
            Comp_proc := FALSE;
        END;

    BEGIN
        :
        (* set file attributes for in_file1, in_file2, and out_file *)
        :
        If_string := 'RSZ = 35, COMP.';
        MergeFiles (If_string, 2, T_table, Comp_proc, Out_file,
                   In_file1, In_file2,
                   Dummy_file, Dummy_file, Dummy_file,
                   Dummy_file, Dummy_file, Dummy_file);
        :
    END.

```

SORT Error Messages

All error messages are displayed in the following form:

<job number> SORT ERROR #<error number>

Table 12-2 contains the numbers for fatal error messages and an explanation of the possible errors.

Refer to Table 12-3 for nonfatal error messages.

Table 12-2. Fatal Error Messages

Error Number	Message, Cause, and Solution
1	RECORD SIZE ZERO. The record size specified was either equal to 0 or equal to or greater than 65.536. Specify the correct record size.
2	INPUT AND OUTPUT RECORD COUNTS DIFFER. The counts of sort input records and sort output records do not agree. Contact your Unisys field engineer.
4	SORT DISKEXHAUSTED DURING STRINGING. The sort disk was exhausted during the stringing phase, and no tapes were specified. Rerun the sort and specify more disk space.
6	INPUT FILE PASSED ALREADY OPEN. Input file passed to SORT was already open. Change the program so close the file before calling the sort routine.
7	UNMATCHED BLOCK NUMBER FROM TAPE INPUT. During a tape or ITD sort, the block number of the last record read from a sort work tape did not match the expected block number. Look for tape I/O error messages and rerun the sort routine. If the problem exists, contact your Unisys field engineer.
8	OUTPUT FILE PASSED NOT LARGE ENOUGH. The output file passed to SORT was not large enough to contain the output, and SORT was unable to expand the output file. Increase the size of the output file or decrease the number of records to be sorted.
9	OUTPUT FILE PASSED ALREADY OPEN. The output file passed to SORT was already open. Change the program so that it closes the file before calling the sort routine.
10	IRRECOVERABLE I/O ERROR ON WORKFILE READ. An irrecoverable I/O error occurred while the system was reading a sort work tape or work disk file. Look for I/O error messages, use a different disk or tape unit, and rerun the sort routine.
11	IRRECOVERABLE I/O ERROR ON WORKFILE WRITE. An irrecoverable I/O error occurred while the system was writing a sort work tape or work disk file. Look for I/O error messages, use a different disk or tape unit, and rerun the sort routine.
12	IRRECOVERABLE I/O ERROR ON CONTROL FILE. An irrecoverable I/O error occurred while the system was reading or writing control records in the sort control file. Look for I/O error messages, use a different disk or tape unit, and rerun the sort routine.

continued

Table 12–2. Fatal Error Messages (cont.)

Error Number	Message, Cause, and Solution
13	IRRECOVERABLE I/O ERROR ON USER OUTPUT FILE. An irrecoverable I/O error occurred while the system was writing your output file. Look for I/O error messages, use a different disk or tape unit, and rerun the sort routine.
14	IRRECOVERABLE I/O ERROR ON USER INPUT FILE. An irrecoverable I/O error occurred while reading your input file. Look for I/O error messages, use a different disk or tape unit, and rerun the sort routine.
15	RESTART RECORD SIZE DIFFERS FROM ORIGINAL. A restart was attempted, but the record size (or character size of the record) did not match the originating sort. When a restartable sort is not able to continue, it saves restart information so that this error occurs during subsequent restart attempts. You can either rerun the restart with the correct record or character size or rerun the original sort routine without the restart option.
16	USER INPUT FILE DIFFERENT AT RESTART TIME. The system attempted to restart, and your input file reached end-of-file (EOF) the system read the restart record. Your input file is shorter at restart time than the original file. Get the correct input file and run the restart again or use the program with a new input file and without the restart option.
17	UNABLE TO READ RESTART INFORMATION. A restart was attempted, but SORT was unable to obtain the necessary restart information from the control file. Check for I/O errors.
18	INPUT FILE HAS AN INDEXED FILE ORGANIZATION. CANNOT BE SORTED. The input file type is invalid. Files using the FILEORGANIZATION attribute with a value of INDEXED or PLIISAM cannot be sorted by the SORT utility.
19	IRRECOVERABLE ERROR DURING RESTART. An irrecoverable error occurred while the system was reading the sort work file; the system had already passed some output records to your output procedure. This error occurs only for restartable sorts with output procedures. Because the sort is restartable, you should specify a restart so that the SORT utility can perform the necessary recovery.
20	DIRECT FILE USED AS INPUT. A direct file was used as an input file. Change the program so that it does not use a direct file declaration.
21	DIRECT FILE USED AS OUTPUT. A direct file was used as an output file. Change the program so that it does not use a direct file declaration.
22	INPUT FILE HAS VARIABLE LENGTH RECORDS. A variable record length file was used as an input file. The SORT utility cannot handle variable-length records.
23	OUTPUT FILE HAS VARIABLE LENGTH RECORDS. A variable record length file was used as an output file. The SORT utility cannot handle variable-length records.

continued

Table 12-2. Fatal Error Messages (cont.)

Error Number	Message, Cause, and Solution
24	SPECIFIED CORESIZE TOO LARGE. The memory size parameter specified for the sort exceeds the number of words of memory physically present on the system. Reduce the value of the memory size parameter in the program.
25	SPECIFIED DISK SIZE EXCEEDS MAXIMUM DISK FILE SIZE. The disk size that you specified is larger than what is allowed. Reduce the specified disk size to a value less than 258,435,456 words.
27	SORT KEY GREATER THAN RECORD SIZE. The sort key that you specified is larger than the record size you specified in your program. Either change the specification for the sort key location and length to be within the record size or correct the record size.
76	INVALID UNIT TYPE. When SORT requested that an additional tape be loaded, a file on some medium other than tape was specified using the FA (File Attribute) system command. Rerun the program again and avoid using the FA command.
84	CONTROL FILE TOO SMALL. The control file is not large enough to contain all control records. This error is an internal sort error and can be circumvented by specifying more disk or a different memory size, or both.

The errors in Table 12-3 appear on the display like other sort error messages, but the SORT does not terminate as a result of these errors.

Table 12-3. Nonfatal Error Messages

Error Number	Message, Cause, and Solution
3	INSUFFICIENT MEMORY FOR MEMORY SORT. Insufficient memory was specified for a memory-only sort. If the SORTLIMITS option of the OPTION task attribute was set, SORT waits on an RSVP after issuing the message ENTER OK TO ALLOW SORT TO INCREASE MEMORY FROM xx TO yy where xx is the original (insufficient) core size and yy is xx plus the default core size. A response of OK causes SORT to expand the file and continue processing. A response of NOTOK causes SORT to terminate and display an error message. If the task attribute option SORTLIMITS was set to FALSE, normal error handling results in SORT being terminated.
5	SORT DISK EXHAUSTED DURING MERGING. Sort disk was exhausted during the merge phase, and no tapes were specified. If the SORTLIMITS option of the OPTION task attribute was set, SORT waits on an RSVP after issuing the message SORT FILE FILL = OK TO EXPAND SORT FILE BY xx SEGMENTS ON <pack name> where xx is the default core size and <pack name> is the name of the disk pack on which the sort file is located. A response of OK causes SORT to expand the file and continue processing. A response of NOTOK causes SORT to terminate and display an error message. If the task attribute option SORTLIMITS was set to FALSE, normal error handling results in SORT being terminated.
26	TOO MANY RECORDS FOR MEMORY SORT. DISK SORT WILL BE USED. Not enough memory is available for a memory sort. The system performs a disk sort instead.
30	CONTROL FILE NOT LARGE ENOUGH. The control file is not large enough to contain two copies of the control records. Two copies are maintained for sorts with error recovery. This error is an internal sort problem and can be circumvented by specifying more disk or a different memory size. SORT continues when this error occurs; however, the error recovery function is disabled.
31	I/O ERROR ON CONTROL FILE WRITE. An irrecoverable I/O error occurred while writing the control file for an error recovery sort. One copy of the control records is discarded, and SORT continues using one copy of the control records.
32	I/O ERROR ON WORK FILE WRITE. An irrecoverable I/O error occurred while writing the sort work file. Error recovery mode is abandoned, and SORT continues as a normal restartable sort.
33	WORKFILE TOO SMALL FOR ERR/REC. Insufficient disk space was provided for the work file to contain three copies of the data. This situation occurs only when error recovery mode is requested. SORT continues as a normal restartable sort with error recovery reset.

continued

Table 12-3. Nonfatal Error Messages (cont.)

Error Number	Message, Cause, and Solution
Unnumbered	RESTART ALLOWED ON DISK SORT ONLY--SORT IGNORED. A restart is not allowed on a disk sort. The sort procedure will proceed to do the entire sort from the beginning of the routine.
Unnumbered	INVALID CORESIZE GIVEN--WILL USE A DEFAULT OF 12000. You specified an invalid coresize. The program will use a default value.
Unnumbered	INVALID CORESIZE GIVEN--WILL USE A DEFAULT OF 75000. You specified an invalid coresize. The program will use a default value.

SORT Statistical Array

If the MCP is compiled with the compiler control option SORTSTAT set to TRUE, a sort statistical array that contains data collected while SORT was executing is created. This option must be set to TRUE before the first reference to the option.

A copy of the sort statistical array is written into a file titled SORT/STATISTICX and is passed to the output procedure of your program if such a procedure exists. A program SYMBOL/SORTSTAT (source language) or SYSTEM/SORTSTAT (object code) is provided on the system release tapes. This program reads the SORT/STATISTICX file and produces a report from the sort statistical information. The sole purpose of this program is to guide you in using the sort statistical information. Any other use of this program or the sort statistical information is left entirely to you. When the contents of the sort statistical array are changed, those changes are reflected in SYMBOL/SORTSTAT and SYSTEM/SORTSTAT.

The following describes the contents of the sort statistical array. Unless otherwise specified, units are words.

Word	Field	Value	Contents
0			Control word from compiler.
	[46:01]		
		1	Merge.
		0	Sort.
	[43:02]		Compiler identification.
		1	COBOL.
		Any other value.	ALGOL.
	[13:04]		Number of merge inputs.

continued

continued

Word	Field	Value	Contents
	[09:01]		Output destination of SORT.
		1	Procedure.
		0	File.
	[08:01]		Input source to SORT.
		1	Procedure.
		0	File.
	[07:08]		This word contains eight fields of 1 bit each. The bit number equals the source number; that is, bit 0 = source 0. Source is input to merge.
		1	Merge input procedure.
		0	Merge file.
1			Memory utilization information.
	[47:12]		Merge order (records).
	[35:16]		Stringing vector size (records).
	[19:20]		Specified memory size (words).
2			Work file information.
	[47:06]		Disk work file unit type.
	[41:10]		Number of tapes specified.
	[31:16]		Sort disk block (words).
	[15:16]		Sort tape block (words).
3			Processor time of stringing phase (units of 2.4 microseconds).
4			Processor time of merging phase (units of 2.4 microseconds).
5			Disk input. This word contains six fields of eight bits each.

continued

SORT Utility

continued

Word	Field	Value	Contents
			Each field reflects the number of rows for a particular disk type.
	[47:08]		IIB-6
	[39:08]		IIB-2
	[31:08]		IC-4
	[23:08]		5N
	[15:08]		IC-3
	[07:08]		PACK
6			Disk output. This word contains six fields of eight bits each. each field reflects the number of rows for a particular disk type.
	[47:08]		IIB-6
	[39:08]		IIB-2
	[31:08]		IC-4
	[23:08]		5N
	[15:08]		IC-3
	[07:08]		PACK
7			Run date (TIME(15)).
8			Disk size specified (words).
9			MCP level.
	[47:16]		Mark level.
	[31:16]		Release level.
	[15:16]		Patch level.
10			Time(11) at initialization.
11			Time(11) at end of stringing phase.
12			Time(11) at end of merging phase.

continued

continued

Word	Field	Value	Contents
13			IO time during string phase (units of 2.4 microseconds).
14			IO time during merging phase (units of 2.4 microseconds).
15			Number of strings created when stringing to disk (DISK and ITD sorts).
16			Number of strings created when stringing to tape (TAPE sort) or merging disk to tape (ITD sort).
17			Sort record information.
	[47:16]		Sort record size. Size is in characters for COBOL; otherwise, size is in words.
	[31:32]		Number of input records.
18			Number of calls on compare procedure during merging phase.
19			Number of calls on compare procedure during stringing phase.
20			Unit type information. This word contains eight fields of 6 bits each. Each field contains the unit type of one of the eight possible work tape or merge inputs.
	[47:06]		FILE7.
	[41:06]		FILE6.
	[35:06]		FILE5.
	[29:06]		FILE4.
	[23:06]		FILE3.
	[17:06]		FILE2.
	[11:06]		FILE1.

continued

SORT Utility

continued

Word	Field	Value	Contents
	[05:06]		FILE0.
21			Input and output file types and file density information for the eight possible work tapes or merge inputs, and input and output files.
	[47:06]		Input file unit type.
	[41:06]		Output file unit type.
	[29:03]		File7 density code.
	[26:03]		FILE6 density code.
	[23:03]		FILE5 density code.
	[20:03]		FILE4 density code.
	[17:03]		FILE3 density code.
	[14:03]		FILE2 density code.
	[11:03]		FILE1 density code.
	[08:03]		FILE0 density code.
	[05:03]		Output file density code.
	[02:03]		Input file density code.
22			Number of rows for various types of work disk files. This word contains six fields of 8 bits each. Each field reflects the number of rows for a particular disk type.
	[47:08]		IIB-6
	[39:08]		IIB-2
	[31:08]		IC-4
	[23:08]		5N
	[15:08]		IC-3
	[07:08]		PACK
23			Reserved for expansion.
24-29			Job name in standard form representation.

Table 12-4 shows the collating sequence used by the SORT utility.

Table 12-4. SORT Collating Sequence

Binary	Hexadecimal	EBCDIC
00000000	00	NUL
00000001	01	SOH
00000010	02	STX
00000011	03	ETX
00000100	04	
00000101	05	HT
00000110	06	
00000111	07	DEL
00001000	08	
00001001	09	
00001010	0A	
00001011	0B	VT
00001100	0C	FF
00001101	0D	CR
00001110	0E	S0
00001111	0F	S1
00010000	10	DLE
00010001	11	DC1
00010010	12	DC2
00010011	13	DC3
00010100	14	
00010101	15	NL
00010110	16	BS
00010111	17	
00011000	18	CAN
00011001	19	EM
00011010	1A	
00011011	1B	
00011100	1C	FS
00011101	1D	GS
00011110	1E	RS

continued

Table 12-4. SORT Collating Sequence (cont.)

Binary	Hexadecimal	EBCDIC
00011111	1F	US
00100000	20	
00100001	21	
00100010	22	
00100011	23	
00100100	24	
00100101	25	LF
00100110	26	ETB
00100111	27	ESC
00101000	28	
00101001	29	
00101010	2A	
00101011	2B	
00101100	2C	
00101101	2D	ENQ
00101110	2E	ACK
00101111	2F	BEL
00110000	30	
00110001	31	
00110010	32	SYN
00110011	33	
00110100	34	
00110101	35	
00110110	36	
00110111	37	EOT
00111000	38	
00111001	39	
00111010	3A	
00111011	3B	
00111100	3C	DC4
00111101	3D	NAK

continued

Table 12-4. SORT Collating Sequence (cont.)

Binary	Hexadecimal	EBCDIC
00111110	3E	
00111111	3F	SUB
01000000	40	SP (blank)
01000001	41	
01000010	42	
01000011	43	
01000100	44	
01000101	45	
01000110	46	
01000111	47	
01001000	48	
01001001	49	
01001010	4A	[
01001011	4B	.
01001100	4C	<
01001101	4D	(
01001110	4E	+
01001111	4F	!
01010000	50	&
01010001	51	
01010010	52	
01010011	53	
01010100	54	
01010101	55	
01010110	56	
01010111	57	
01011000	58	
01011001	59	
01011010	5A]
01011011	5B	\$
01011100	5C	*

continued

Table 12-4. SORT Collating Sequence (cont.)

Binary	Hexadecimal	EBCDIC
01011101	5D)
01011110	5E	;
01011111	5F	^
01100000	60	-
01100001	61	/
01100010	62	
01100011	63	
01100100	64	
01100101	65	
01100110	66	
01100111	67	
01101000	68	
01101001	69	
01101010	6A	
01101011	6B	,
01101100	6C	%
01101101	6D	_
01101110	6E	>
01101111	6F	?
01110000	70	
01110001	71	
01110010	72	
01110011	73	
01110100	74	
01110101	75	
01110110	76	
01110111	77	
01111000	78	
01111001	79	\
01111010	7A	:

continued

Table 12-4. SORT Collating Sequence (cont.)

Binary	Hexadecimal	EBCDIC
01111011	7B	;
01111100	7C	@
01111101	7D	'
01111110	7E	=
01111111	7F	"
10000000	80	
10000001	81	a
10000010	82	b
10000011	83	c
10000100	84	d
10000101	85	e
10000110	86	f
10000111	87	g
10001000	88	h
10001001	89	i
10001010	8A	
10001011	8B	
10001100	8C	
10001101	8D	
10001110	8E	
10001111	8F	
10010000	90	
10010001	91	j
10010010	92	k
10010011	93	l
10010100	94	m
10010101	95	n
10010110	96	o
10010111	97	p
10011000	98	q

continued

Table 12-4. SORT Collating Sequence (cont.)

Binary	Hexadecimal	EBCDIC
10011001	99	r
10011010	9A	
10011011	9B	
10011100	9C	
10011101	9D	
10011110	9E	
10011111	9F	
10100000	A0	
10100001	A1	" "
10100010	A2	s
10100011	A3	t
10100100	A4	u
10100101	A5	v
10100110	A6	w
10100111	A7	x
10101000	A8	y
10101001	A9	z
10101010	AA	
10101011	AB	
10101100	AC	
10101101	AD	
10101110	AE	
10101111	AF	
10110000	B0	
10110001	B1	
10110010	B2	
10110011	B3	
10110100	B4	
10110101	B5	
10110110	B6	
10110111	B7	

continued

Table 12-4. SORT Collating Sequence (cont.)

Binary	Hexadecimal	EBCDIC
10111000	B8	
10111001	B9	
10111010	BA	
10111011	BB	
10111100	BC	
10111101	BD	
10111110	BE	
10111111	BF	
11000000	C0	{
11000001	C1	A
11000010	C2	B
11000011	C3	C
11000100	C4	D
11000101	C5	E
11000110	C6	F
11000111	C7	G
11001000	C8	H
11001001	C9	I
11001010	CA	
11001011	CB	
11001100	CC	
11001101	CD	
11001110	CE	
11001111	CF	
11010000	D0	}
11010001	D1	J
11010010	D2	K
11010011	D3	L
11010100	D4	M
11010101	D5	N
11010110	D6	O

continued

Table 12-4. SORT Collating Sequence (cont.)

Binary	Hexadecimal	EBCDIC
11010111	D7	P
11011000	D8	Q
11011001	D9	R
11011010	DA	
11011011	DB	
11011100	DC	
11011101	DD	
11011110	DE	
11011111	DF	
11100000	E0	\
11100001	E1	
11100010	E2	S
11100011	E3	T
11100100	E4	U
11100101	E5	V
11100110	E6	W
11100111	E7	X
11101000	E8	Y
11101001	E9	Z
11101010	EA	
11101011	EB	
11101100	EC	
11101101	ED	
11101110	EE	
11101111	EF	
11110000	F0	0
11110001	F1	1
11110010	F2	2
11110011	F3	3
11110100	F4	4
11110101	F5	5

continued

Table 12-4. SORT Collating Sequence (cont.)

Binary	Hexadecimal	EBCDIC
11110110	F6	6
11110111	F7	7
11111000	F8	8
11111001	F9	9
11111010	FA	
11111011	FB	
11111100	FC	
11111101	FD	
11111110	FE	
11111111	FF	

Section 13

XREFANALYZER Utility

The SYSTEM/XREFANALYZER utility is a system software utility that constructs detailed information about all identifiers declared in a program.

The cross-reference files (XREF files) generated by XREFANALYZER contain entries for each identifier in a program. Each separate entry is referred to as the *header line information*. The header line contains the following information about the identifier:

- The alphanumeric name
- The environment
- The declared type
- The stack location
- The sequence number of the declaration

This information is helpful in developing, debugging, and updating programs.

Cross-reference information is generated by a compiler during the compilation of a particular version of a program. Thus, the XREF files do not reflect changes that are made to a file after the compilation was performed. Also, if any syntax errors are encountered during the compilation of a program, XREFANALYZER is not run.

The language compilers that have the capability to execute XREFANALYZER are ALGOL, DCALGOL, FORTRAN, Pascal, and COBOL74. FORTRAN does not keep track of environment information; therefore, no environment information is included in the header line.

When XREFANALYZER is executed, it makes two passes through its input file. The first pass builds the REFERENCES file. The second pass alphabetically sorts the identifiers and writes the actual cross-reference listing.

XREFANALYZER Files

The two files that can be created by XREFANALYZER are titled *XREFFILES/<code file name>/DECS* and *XREFFILES/<code file name>/REFS*. The code file name is the title of the code file that was generated by the compiler when the XREF file was created. If compiled through the Command and Edit (CANDE) message control system (MCS), the code file name normally has the prefix *OBJECT/*.

The DECS file contains information about all declarations in a file. The REFS file contains information about the references to all identifiers in a program.

These files are used by the A Series Editor and by SYSTEM/INTERACTIVEXREF to interactively access the cross-reference information for a program. XREFANALYZER

includes version information in the XREF files, so that when INTERACTIVEXREF is executed, if it finds that the version information is not compatible, an error message is displayed indicating that the XREFFILES were created by an incompatible XREFANALYZER.

Invoking XREFANALYZER

Implicit Execution

Two compiler control options are used to produce a printed output of the cross-reference information for a program, to create the files necessary for the Editor and INTERACTIVEXREF, or both.

When the compiler control option XREF is TRUE in a program, the language compiler creates and saves a file titled *XREF/<code file name>*, which contains raw cross-reference information. XREFANALYZER is then run to organize and print the data contained in the XREF file. XREFANALYZER then purges the XREF file.

When the compiler control option XREFFILES is TRUE in a program, the language compiler creates and saves the XREF file, XREFANALYZER is run to create and save the DECS and REFS files, and finally the XREF file is purged.

When both XREF and XREFFILES are TRUE, the printed output and the DECS and REFS files are produced.

Both of these options are available in all language compilers that are capable of running XREFANALYZER. Refer to individual language manuals for any additional compiler options that are available for use with XREFANALYZER.

Explicit Execution

You can also obtain cross-reference information by running XREFANALYZER explicitly. The input file *TITLE=XREF/<code file name>* must be specified during initialization, and you should be aware of the following:

Files

XREFFILE

This is the input file and it should be label-equated to a valid XREF file produced by a compiler. This file is purged when XREFANALYZER is done with it unless the compile-time option DEBUG is TRUE. The title of this file should normally begin with *XREF/*. If an invalid XREF file is used, the following error message is printed and XREFANALYZER is terminated.

```
;;;;;;;;;; XREF ERROR: 1 ;;;;;;;;;;
```

LINE

This is the output file.

DECLARATIONS

This file contains information about all declarations and can optionally be saved for later use. Its title is constructed from the XREF file by replacing *XREF/* with *XREFFILES/* and appending */DECS* to the end of the title.

REFERENCES

This file contains information about all references and can optionally be saved for later use. Its title is constructed from the XREF file by replacing *XREF/* with *XREFFILES/* and appending */REFS* to the end of the title.

Parameters

LINewidth

XREFANALYZER is a procedure that has one REAL parameter: the line width of the output file LINE. The value of this parameter can be set between 72 and 132 characters. If it is set at less than 72, then the value is adjusted to 72. If it is set at more than 132, then the value is adjusted to 132. If it is set at 0, then printed output is not produced.

TASKVALUE

If the task value for XREFANALYZER is set to a negative number, then cross-reference information is saved in the DECLARATIONS and REFERENCES files.

Compile Time Options

The compile-time options that are available for XREFANALYZER are the following:

- FAST
- DEBUG

When FAST is set to TRUE, blank lines are not inserted between items in the XREF output, and the following character substitutions are made to increase printing speed. The default is FALSE.

- - FOR :
- E FOR =
- AT FOR @

XREFANALYZER Utility

When `DEBUG` is set to `TRUE`, it aids in debugging `XREFANALYZER`. `DEBUG` adds extra checks to the code and prevents the `XREF` file from being purged. The default is `FALSE`.

Examples

The following is an `ALGOL` program with its corresponding cross-reference listing.

```

00000100 $SET XREF
00000200 BEGIN
00000300 REAL B, C, MEAN;
00000400 INTEGER I;
00000500 ARRAY R[1:3];
00000600 DEFINE ONEE= 1;;
00000700          TWOO= 2;;
00000800          THREEE= 3;;
00000900
00001000 PROCEDURE ONE;
00001100     BEGIN
00001200     MEAN:= 2;
00001300     FOR I:= 0 STEP 1 UNTIL 5 DO
00001400         MEAN:= MEAN * B;
00001500     C:= C * MEAN;
00001600     END ONE;
00001700
00001800
00001900 R[ONEE]:= 1;
00002000 R[TWOO]:= 2;
00002100 R[THREEE]:= R[ONEE] + R[TWOO];
00002200 B:= 3;
00002300 C:= 25;
00002400 ONE;
00002500 B:= 5;
00002600 ONE;
00002700 END.

B---REAL AT (2,2)---DECLARED AT 00000300
      00001400 *00002200 *00002500
B.0000 - PROCEDURE AT (1,2) - DECLARED AT 0000200 ENDS AT 00002700
C---REAL AT (2,3)---DECLARED AT 00000300
      *00001500 *00002300
I---INTEGER AT (2,5)---DECLARED AT 00000400
      *00001300
MEAN---REAL AT (2,4)---DECLARED AT 00000300
      *00001200 *00001400 00001500
ONE - PROCEDURE AT (2,7) - DECLARED AT 00001000 ENDS AT 00001600
      00002400 00002600
ONEE---DEFINE---DECLARED AT 00000600
      00001900 00002100
R---REAL ARRAY AT (2,6)---DECLARED AT 00000500
      *00001900 *00002000 *00002100
THREEE---DEFINE---DECLARED AT 00000800
      00002100
TWOO---DEFINE---DECLARED AT 00000700
      00002000 00002100

```

Each reference consists of an 8 digit sequence number, preceded by either an E if the reference appears in an address equation or an asterisk (*) if the value might be changed by the statement.

XREFANALYZER Utility

The sequence number is followed by a number sign (#) if the reference occurs as part of an expanded define.

The following is a FORTRAN program with its corresponding cross-reference listing.

```
00000100 $SET XREF
00000200     REAL B, C, MEAN
00000300     INTEGER I
00000400     DIMENSION R(3)
00000500     R(1) = 1
00000600     R(2) = 2
00000700     R(3) = R(1) + R(2)
00000800     B = 3
00000900     C = 25
00001000     CALL ONE
00001100     B = 5
00001200     CALL ONE
00001300     END
00001400     SUBROUTINE ONE
00001500         MEAN = 2
00001600         DO 10 I=0,5,1
00001700             MEAN = MEAN * B
00001800             C = C * MEAN
00001900 10     CONTINUE
00002000     RETURN
00002100     END

B    ---REAL---DECLARED AT 00000200
    *00000800 *00001100
B    ---REAL---DECLARED AT 00001700
C    ---REAL---DECLARED AT 00000200
    *00000900
C    ---REAL---DECLARED AT 00001800
    *00001800
I    ---INTEGER---DECLARED AT 00000300
I    ---INTEGER---DECLARED AT 00001600
    *00001600
MEAN ---REAL---DECLARED AT 00000200
MEAN ---INTEGER---DECLARED AT 00001500
    *00001500 *00001700 00001800
ONE  ---SUBROUTINE---DECLARED AT 00001000
    00001200 00001400
R    ---REAL ARRAY---DECLARED AT 00000400
    *00000500 *00000600 *00000700
10   ---LABEL---FORWARD AT 00001600     OCCURS AT 00001900
```

Each reference consists of an 8 digit sequence number, preceded by an asterisk (*) if the value might be changed by the statement.

The following job gives an example of running XREFANALYZER explicitly to generate the files that are necessary for Editor and INTERACTIVEXREF to perform cross-referencing commands.

```
10000 ?BEGIN JOB WBM;
20000 INTEGER TASKVAL;
30000 TASK T;
40000 TASKVAL:= -1;
50000 COMPILE OBJ/WBM/1 ALGOL [T] LIBRARY;
60000 DATA
70000 $SET NOXREFLIST XREF
80000 BEGIN
90000 INTEGER I;
100000 I:= 0;
120000 END.
130000 ?%END OF ALGOL DATA
140000 IF T IS COMPILEDOK THEN
150000     BEGIN
160000         RUN SYSTEM/XREFANALYZER (0);
170000         VALUE=TASKVAL;
180000         FILE XREFFILE (TITLE=XREF/OBJ/WBM/1);
190000         FILE REFERENCES (SECURITYTYPE=PUBLIC,SECURITYUSE=IO);
200000         FILE DECLARATIONS (SECURITYTYPE=PUBLIC,SECURITYUSE=IO);
210000         END;
220000 ?END JOB
```

The title of the ALGOL program is WBM/1. When compiled, it generates cross-reference information containing a list of all identifiers that appear in the program and saves it as the XREF file. Since NOXREFLIST is TRUE, XREFANALYZER is not initiated by the ALGOL compiler and consequently no cross-reference listing is printed. When XREFANALYZER is run explicitly, the TASKVALUE equal to -1 causes the DECS and REFS files to be created and saved. Since the LINEWIDTH parameter is equal to 0, no printed output is produced.

Appendix A

Understanding Railroad Diagrams

What Are Railroad Diagrams?

Railroad diagrams are diagrams that show you the rules for putting words and symbols together into commands and statements that the computer can understand. These diagrams consist of a series of paths that show the allowable structure, constants, and variables for a command or a statement. Paths show the order in which the command or statement is constructed. Paths are represented by horizontal and vertical lines. Many railroad diagrams have a number of different paths you can take to get to the end of the diagram. For example:



If you follow this railroad diagram from left to right, you will discover three acceptable commands. These commands are

- REMOVE
- REMOVE SOURCE
- REMOVE OBJECT

If all railroad diagrams were this simple, this explanation could end here. However, because the allowed ways of communicating with the computer can be complex, railroad diagrams sometimes must also be complex.

Regardless of the level of complexity, all railroad diagrams are visual representations of commands and statements. Railroad diagrams are intended to

- Show the mandatory items.
- Show the user-selected items.
- Present the order in which the items must appear.
- Show the number of times an item can be repeated.
- Show the necessary punctuation.

To familiarize you with railroad diagrams, this explanation describes the elements of the diagrams and provides examples.

Some of the actual railroad diagrams you will encounter might be more complex. However, all railroad diagrams, simple or complex, follow the same basic rules. They

Understanding Railroad Diagrams

all consist of paths that represent the allowable structure, constants, and variables for commands and statements.

By following railroad diagrams, you can easily understand the correct syntax for commands and statements. Once you become proficient in the use of railroad notation, the diagrams serve as quick references to the commands and statements.

Constants and Variables

A constant is an item that cannot be altered. You must enter the constant as it appears in the diagram, either in full or as an allowable abbreviation. If a constant is partially underlined, you can abbreviate the constant by entering only the underlined letters. In addition to the underlined letters, any of the remaining letters can be entered. If no part of the constant is underlined, the constant cannot be abbreviated. Constants can be recognized by the fact that they are never enclosed in angle brackets (< >) and are in uppercase letters.

A variable is an item that represents data. You can replace the variable with data that meets the requirements of the particular command or statement. When replacing a variable with data, you must follow the rules defined for the particular command or statement. Variables appear in railroad diagrams enclosed in angle brackets.

In the following example, BEGIN and END are constants while <statement list> is a variable. The constant BEGIN can be abbreviated since it is partially underlined. Valid abbreviations for BEGIN are BE, BEG, and BEGI.

— BEGIN —<statement list>— END —————|

Constraints

Constraints are used in a railroad diagram to control progression through the diagram. Constraints consist of symbols and unique railroad diagram line paths. They include

- Vertical bars
- Percent signs
- Right arrows
- Required items
- User-selected items
- Loops
- Bridges

A description of each item follows.

Vertical Bar

The vertical bar symbol (|) represents the end of a railroad diagram and indicates the command or statement can be followed by another command or statement.

— SECONDWORD — (—<arithmetic expression>—) —————|

Percent Sign

The percent sign (%) represents the end of a railroad diagram line and indicates the command or statement must be on a line by itself.

— STOP —————| %

Right Arrow

The right arrow symbol (>) is used when the railroad diagram is too long to fit on one line and must continue on the next. A right arrow appears at the end of the first line, and another right arrow appears at the beginning of the next line.

— SCALERIGHT — (—<arithmetic expression>— , —————→
→<arithmetic expression>—) —————|

Required Items

A required item can be either a constant, a variable, or punctuation. A required item appears as a single entry, by itself or with other items, on a horizontal line. Required items can also exist on horizontal lines within alternate paths or nested (lower-level) diagrams. If the path you are following contains a required item, you must enter the item in the command or statement; the required item cannot be omitted.

In the following example, the word EVENT is a required constant and <identifier> is a required variable:

— EVENT —<identifier>—————|

User-Selected Items

User-selected items appear one below the other in a vertical list. You can choose any one of the items from the list. If the list also contains an empty path (solid line), none of the choices are required. A user-selected item can be either a constant, a variable, or punctuation. In the following railroad diagram, either the plus sign (+) or the minus sign (-) can be entered before the required variable <arithmetic expression>, or the symbols can be disregarded because the diagram also contains an empty path.

+
-

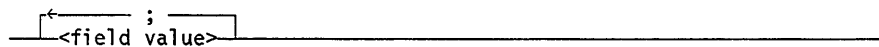
 <arithmetic expression>—————|

Understanding Railroad Diagrams

Loop

A loop represents an item or group of items that you can repeat. A loop can span all or part of a railroad diagram. It always consists of at least two horizontal lines, one below the other, connected on both sides by vertical lines. The top line is a right-to-left path that contains information about repeating the loop.

Some loops include a return character. A return character is a character – often a comma (,) or semicolon (;) – required before each repetition of a loop. If there is no return character, the items must be separated by one or more blank spaces.

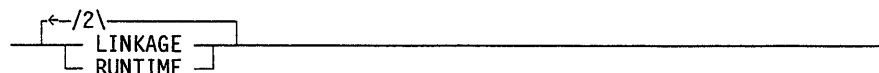
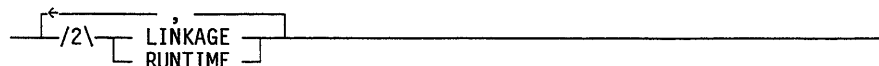


Bridge

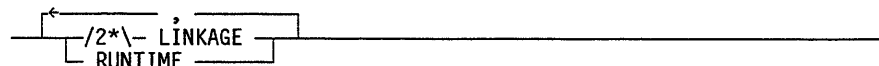
Sometimes a loop also includes a bridge, which is used to show the maximum number of times the loop can be repeated. The bridge can precede the contents of the loop, or it can precede the return character (if any) on the upper line of the loop.

The bridge determines the number of times you can cross that point in the diagram. The bridge is an integer enclosed in sloping lines (/ \). Not all loops have bridges. Those that do not can be repeated any number of times until all valid entries have been used.

In the first bridge example, you can enter LINKAGE or RUNTIME no more than two times. In the second bridge example, you can enter LINKAGE or RUNTIME no more than three times.



In some bridges an asterisk (*) follows the number. The asterisk means that you must cross that point in the diagram at least once. The maximum number of times that you can cross that point is indicated by the number in the bridge.



In the previous bridge example, you must enter LINKAGE at least once but no more than twice, and you can enter RUNTIME any number of times.

The following figure shows the types of constraints used in railroad diagrams.



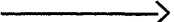

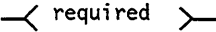
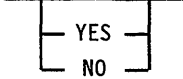
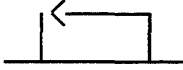
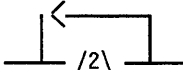
SYMBOL/PATH	EXPLANATION
	Vertical bar. Indicates that the command or statement can be followed by another command or statement.
	Percent sign. Indicates that the command or statement must be on a line by itself.
 	Right arrow. Indicates that the diagram occupies more than one line.
	Required items. Indicates the constants, variables, and punctuation that must be entered in a command or statement.
	User-selected items. Indicates the items that appear one below the other in a vertical list. You select which item or items to include.
	A loop. Indicates an item or group of items that can be repeated.
	A bridge. Indicates the maximum number of times a loop can be repeated.

Figure A-1. Railroad Constraints

Following the Paths of a Railroad Diagram

The paths of a railroad diagram lead you through the command or statement from beginning to end. Some railroad diagrams have only one path, while others have several alternate paths. The following railroad diagram indicates there is only one path that requires the constant LINKAGE and the variable <linkage mnemonic>:

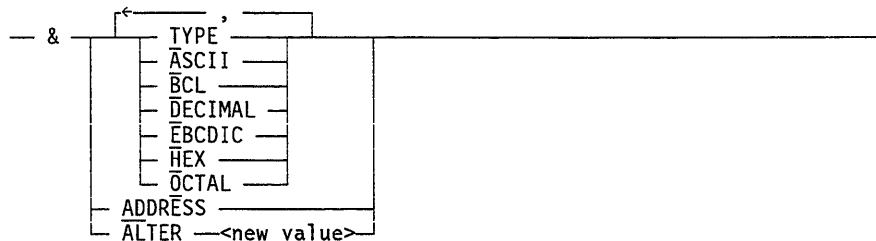
— LINKAGE —<linkage mnemonic>—————|

Alternate paths provide choices in the construction of commands and statements. Alternate paths are provided by loops, user-selected items, or a combination of both. More complex railroad diagrams can consist of many alternate paths, or nested (lower-level) diagrams, that show a further level of detail.

For example, the following railroad diagram consists of a top path and two alternate paths. The top path includes an ampersand (&) and the constants (that are

Understanding Railroad Diagrams

user-selected items) in the vertical list. These constants are within a loop that can be repeated any number of times until all options have been selected. The first alternate path requires the ampersand and the required constant ADDRESS. The second alternate path requires the ampersand followed by the required constant ALTER and the required variable <new value>.



Railroad Diagram Examples with Sample Input

The following examples show five railroad diagrams and possible command and statement constructions based on the paths of these diagrams.

Example 1

<lock statement>



Sample Input

LOCK (FILE4)

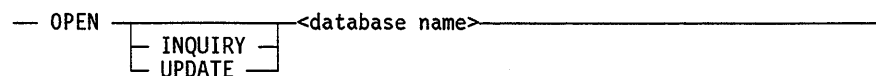
Explanation

LOCK is a constant and cannot be altered. Because no part of the word is underlined, the entire word must be entered.

The parentheses are required punctuation, and FILE4 is a sample file identifier.

Example 2

<open statement>



Sample Input

OPEN DATABASE1

Explanation

The constant OPEN is followed by the variable DATABASE1, which is a database name.

The railroad diagram shows two user-selected items, INQUIRY and UPDATE. However, because there is an empty path (solid line), these entries are not required.

continued

continued

Sample Input

OPEN INQUIRY DATABASE1

OPEN UPDATE DATABASE1

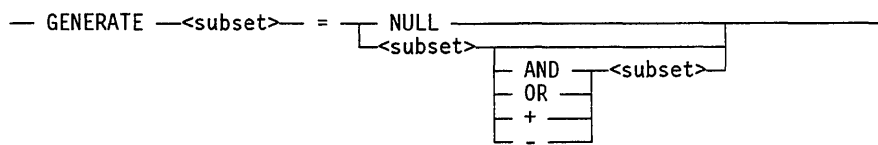
Explanation

The constant OPEN is followed by the user-selected constant INQUIRY and the variable DATABASE1.

The constant OPEN is followed by the user-selected constant UPDATE and the variable DATABASE1.

Example 3

<generate statement>



Sample Input

GENERATE Z = NULL

GENERATE Z = X

GENERATE Z = X AND B

GENERATE Z = X + B

Explanation

The GENERATE constant is followed by the variable Z, an equal sign (=), and the user-selected constant NULL.

The GENERATE constant is followed by the variable Z, an equal sign, and the user-selected variable X.

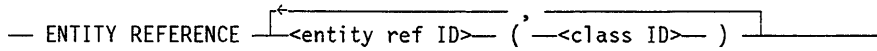
The GENERATE constant is followed by the variable Z, an equal sign, the user-selected variable X, the AND command (from the list of user-selected items in the nested path), and a third variable, B.

The GENERATE constant is followed by the variable Z, an equal sign, the user-selected variable X, the plus sign (from the list of user-selected items in the nested path), and a third variable, B.

Understanding Railroad Diagrams

Example 4

<entity reference declaration>



Sample Input

ENTITY REFERENCE ADVISOR1 (INSTRUCTOR)

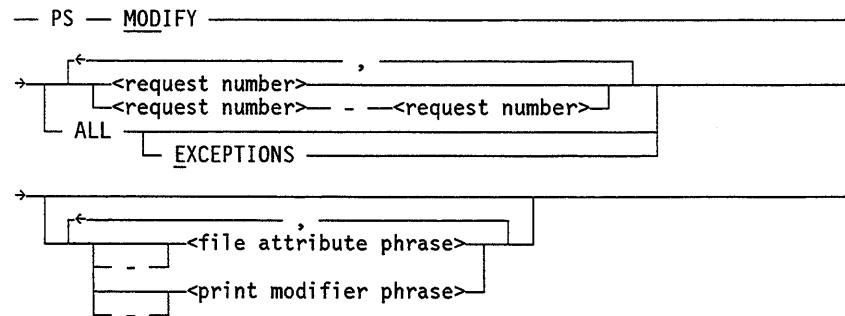
Explanation

The required item ENTITY REFERENCE is followed by the variable ADVISOR1 and the variable INSTRUCTOR. The parentheses are required.

ENTITY REFERENCE ADVISOR1 (INSTRUCTOR),
ADVISOR2 (ASST_INSTRUCTOR)

Because the diagram contains a loop, the pair of variables can be repeated any number of times.

Example 5



Sample Input

PS MODIFY 11159

Explanation

The constants PS and MODIFY are followed by the variable 11159, which is a request number.

PS MODIFY 11159,11160,11163

Because the diagram contains a loop, the variable 11159 can be followed by a comma, the variable 11160, another comma, and the final variable 11163.

PS MOD 11159-11161 DESTINATION =
"LP7"

The constants PS and MODIFY are followed by the user-selected variables 11159-11161, which are request numbers, and the user-selected variable DESTINATION = "LP7", which is a file attribute phrase. Note that the constant MODIFY has been abbreviated to its minimum allowable form.

PS MOD ALL EXCEPTIONS

The constants PS and MODIFY are followed by the user-selected constants ALL and EXCEPTIONS.

Glossary

A

actual segment descriptor (ASD)

Pointer to the location of a data or code item in memory or on disk.

ALGOL

Algorithmic language. A structured, high-level programming language that provides the basis for the stack architecture of the Unisys A Series systems. ALGOL was the first block-structured language developed in the 1960s and served as a basis for such languages as Pascal and Ada. It is still used extensively on A Series systems, primarily for systems programming.

ASD

See actual segment descriptor.

B

block

(1) A group of physically adjacent records that can be transferred to or from a physical device as a group. (2) A program, or a part of a program, that is treated by the processor as a discrete unit. Examples are a procedure in ALGOL, a procedure or function in Pascal, a subroutine or function in FORTRAN, or a complete COBOL program.

BNA

The network architecture used on A Series and V Series systems, as well as CP9500 and CP 2000 communications processors to connect multiple, independent, compatible computer systems into a network for distributed processing and resource sharing.

C

CANDE

See Command and Edit.

CD-ROM

Compact disk read-only memory. A high-density read-only storage medium. The data is stored on a removable polycarbonate disk, and is read by a laser beam.

COBOL

Common Business-Oriented Language. A widely used, procedure oriented language intended for use in solving problems in business data processing. The characteristics are the easy readability of programs and a considerable degree of machine independence. COBOL is the most widely used procedure-oriented language.

Glossary

COBOL74

A version of the COBOL language that is compatible with the American National Standard X3.23-1974.

COBOL85

The latest version of the COBOL language. This version is compatible with the American National Standard X3.23-1985.

Command and Edit (CANDE)

A time-sharing message control system (MCS) that enables a user to create and edit files, and to develop, test, and execute programs, interactively.

Communications Management System (COMS)

A general message control system (MCS) that controls online environments on the A Series systems. COMS can support the processing of multiprogram transactions, single-station remote files, and multistation remote files.

COMS

See Communications Management System.

D

Data Management System II (DMSII)

A specialized system software package used to describe a database and maintain the relationships among the data elements in the database.

Data Communications ALGOL (DCALGOL)

A Unisys language based on ALGOL that contains extensions for writing message control system (MCS) programs and other specialized system programs.

DCALGOL

See Data Communications ALGOL.

directory name

A name used to refer to a group of files whose file names are identical to the directory name, except that the file names have at least one additional node following the directory name.

distributed systems service (DSS)

One of a collection of services provided on Unisys hosts to support communications across multihost networks. DSSs can be services such as file handling, station transfer, and mail transfer.

DMSII

See Data Management System II.

DSS

See distributed systems service.

E**EMS**

See Entry and Medium Systems.

Entry and Medium Systems (EMS)

A designation referring to the Micro A and A 1 through A 10 systems.

F**family name**

The name, consisting of up to 17 alphanumeric characters, assigned by an installation to identify a family of disks.

file attribute

An element that describes a characteristic of a file and provides information the system needs to handle the file. Examples of file attributes are the file title, record size, number of areas, and date of creation. For disk files, permanent file attribute values are stored in the disk file header.

file equation

A mechanism for specifying the values of file attributes when a program is compiled or executed. A file equation implicitly assigns a value to the FILECARDS task attribute.

file name

(1) A name or word that designates a set of data items. (2) A unique identifier for a file, consisting of name constants separated by slashes. Each name constant consists of letters, digits, and selected special characters. A file name can be optionally preceded by an asterisk (*) or usercode, and optionally followed by ON and a family name. (3) In RPG, a name that designates a set of data items. (4) In COBOL, a user-defined word that names a file described in a file description entry or a sort-merge file description entry within the FILE SECTION of the DATA DIVISION.

file title

The complete identifier for a file that consists of the file name, the word ON, and the family name.

file transfer

The communication of files between host systems, workstations, or terminals. The category of distributed systems service (DSS) that enables a user to transfer files between host systems, workstations, or terminals. *See also* library maintenance.

File Transfer, Access, and Management (FTAM)

The standard developed by the International Standards Organization (ISO) for file exchange and management across an Open Systems Interconnection (OSI) network. FTAM systems can access file attributes (for example, password information) and the contents of files (including individual records, as well as entire files).

FORTRAN

Formula Translation. A high-level, structured programming language intended primarily for scientific use.

Glossary

FORTRAN77

A version of the FORTRAN language that is compatible with the ANSI X3.9-1978 standard.

FTAM

See File Transfer, Access, and Management.

G

GEMCOS

See Generalized Message Control System.

Generalized Message Control System (GEMCOS)

A message control system (MCS) developed for online systems. GEMCOS is transaction oriented.

H

HDP

See host dependent port.

HDU

See host data unit.

host data unit (HDU)

The A 12 and A 15 system host interface to the I/O subsystem. An HDU is configured with up to three host dependent ports (HDPs), each of which supports two message level interface (MLI) cables.

host dependent port (HDP)

A hardware module capable of interfacing a processor to the message level interface (MLI).

I

I/O

See input/output.

I/O processor (IOP)

A specialized processor for moving data between system memory and the I/O subsystem.

indexed sequential-access method (ISAM)

A method that provides efficient, flexible random access to records identified by keys stored in an index.

initialization, verification, and relocation (IVR)

A maintenance procedure used to write sector boundaries and a blank label on a disk. You can use the IVR procedure to make a new disk pack usable by the system or a

damaged disk reusable by eliminating defective sectors. The end product of an IVR is a master available table (MAT) of available disk segments.

input/output (I/O)

An operation in which the system reads data from or writes data to a peripheral device such as a disk.

IOP

See I/O processor.

ISAM

See indexed sequential-access method.

IVR

See initialization, verification, and relocation.

J**job**

(1) A group of one or more tasks under the control of a single Work Flow Language (WFL) program. The system assigns each job a mix number and treats each job as a discrete unit of work. (2) *See* WFL job.

job log

A log that is stored in a job file and contains log entries for a particular job and its descendant tasks. When the job terminates, the job log is processed to produce the job summary.

job queue

A structure in the system software that stores a list of jobs that have been compiled and are waiting to be initiated.

job summary

A file, produced after a job completes execution, that lists information such as the tasks initiated by the job, the beginning and ending times for each task, and the termination information for each task.

L**label**

(1) The first 28 sectors on a disk, in which information about the disk is stored. This information includes the family name and serial number, the master available table (MAT), the family index number, information about the family's base pack, and a pointer to the flat directory if the disk contains a directory. (2) An area on a magnetic tape (MT) that contains permanent attributes associated with the tape volume or with individual files on the volume, such as the volume serial number and the file name. (3) In some programming languages, a name that identifies either a point in the Calculation Specifications where a GOTO operation branches or the beginning of a subroutine.

labeled tape

A tape that has label records. The label records contain information needed to locate a specific file on a tape. Each file on a labeled tape is preceded and followed by a set of label records. A tape mark is used to separate the label records from the records of a file on the tape.

large systems

Refers to systems that interface with a host data unit (HDU) or a resource management module (RMM).

library

(1) A collection of one or more named routines or library objects that are stored in a file and can be called by other programs. (2) A program that exports objects for use by user programs.

library maintenance tape

A tape created by library maintenance that contains backup copies of disk files.

logging

The process of recording events, and, often, their times of occurrence.

LS

See large systems.

M

magnetic tape (MT)

A tape with a surface layer that can be magnetized, on which data can be stored by magnetic recording, and from which data can be retrieved.

MARC

See Menu-Assisted Resource Control.

master available table (MAT)

A table stored on each disk that lists the valid sectors on the disk that were successfully processed by the initialization, verification, and relocation (IVR) procedure. Pointers to defective sectors are deleted from the MAT so that these sectors will not be accessed. Normally, the MAT shows the entire disk as being available, minus any defective sectors.

master control program (MCP)

The central program of the A Series operating system. The term applies to any master control program that Unisys may release for A Series systems.

MAT

See master available table.

MCP

See master control program.

MCS

See message control system.

Menu-Assisted Resource Control (MARC)

A menu-driven interface to A Series systems that also enables direct entry of commands.

message control system (MCS)

A program that controls the flow of messages between terminals, application programs, and the operating system. MCS functions can include message routing, access control, audit and recovery, system management, and message formatting.

message level interface (MLI)

The interface between the host system, the I/O subsystem, and the data communications subsystem.

message level interface processor (MLIP)

See I/O Processor (IOP) and Entry and Medium Systems (EMS).

MLI

See message level interface.

MLIP

See message level interface processor.

MT

See magnetic tape.

multivolume

A tape file or a set of tape files that have been written to successive tape volumes. When the end of a tape volume is reached while the system is reading or writing a file, the system automatically switches to the next tape volume for that file.

N**NDLII**

See Network Definition Language II.

Network Definition Language II (NDLII)

The Unisys language used to describe the physical, logical, and functional characteristics of the data communications subsystem to network support processors (NSPs), line support processors (LSPs), and data communications data link processors (DCDLPs).

network support processor (NSP)

A data communications subsystem processor that controls the interface between a host system and the data communications peripherals. The NSP executes the code generated by the Network Definition Language II (NDLII) compiler for line control and editor procedures. An NSP can also control line support processors (LSPs).

nonstandard labeled tape

A tape created by systems other than A series systems. Such a tape might have a label that is not fully compatible with the A Series system.

NSP

See network support processor.

O

ODT

See operator display terminal.

operating system

The set of programs that control the operational environment of a computer system by activities such as managing processors, memory, and peripherals, logging system activities, enforcing security, and executing system commands. On A Series systems, the operating system consists of a master control program (MCP) and system libraries such as CENTRALSUPPORT, GENERALSUPPORT, JOBFORMATTER, and PRINTSUPPORT.

operator display terminal (ODT)

(1) A terminal or other device that is connected to the system in such a way that it can communicate directly with the operating system. The ODT allows operations personnel to accomplish system operations functions through either of two operating modes: system command mode or data comm mode. (2) The name given to the system control terminal (SCT) when it is used as an ODT.

P

Pascal

A high-level programming language developed by Niklaus Wirth, based on the block structuring nature of ALGOL 60 and the data structuring innovations of C.A.R. Hoare. Pascal is a general purpose language.

peripheral device

A hardware device used for input, output, or file storage. Examples are magnetic tape drives, printers, and disk drives.

R

record

(1) A group of logically related items of data in a file that are treated as a unit. (2) The data read from or written to a file in one execution of a read or write statement in a program.

remote job entry (RJE)

A Unisys message control system (MCS) that permits jobs, data, and control commands to be sent to a central system from a remote card reader; RJE also permits output of data from the central system to be sent to remote peripherals.

Report Program Generator (RPG)

See RPG.

RJE

See remote job entry.

RPG

Report Program Generator. A high-level, commercially oriented programming language used most frequently to produce reports based on information derived from data files.

T**tape mark**

A special physical record that a system or program writes on a magnetic tape volume to delimit logical entities, such as files, from one another.

task

(1) A dependent process. (2) Any process, whether dependent or independent. *See also* process.

transliteration

The mapping of characters in one representation to characters in another representation.

U**unit number**

A number assigned by an installation to a peripheral device (such as a disk drive) and used to identify the device. The unit number is commonly used in conjunction with an acronym indicating the type of unit to provide a unique identifier for a particular peripheral.

unlabeled tape

A tape without any identifying data at the beginning of the tape. An unlabeled tape can be read only as data and cannot be used as input to library maintenance or the print system.

usercode

An identification code used to establish user identity, control security, and provide for segregation of files. Usercodes can be applied to every task, job, session, and file on the system. A valid usercode is identified by an entry in the USERDATAFILE.

V**volume**

The medium of a mass storage device such as a disk, disk pack, or tape reel. The term *volume* is not restricted to the volume library on a cataloging system or the volume

directory on a system with tape volume security. For example, on the BTOS™ family of workstations, the hard disk is a volume, and each floppy disk is a volume. When a volume is initialized, it is assigned a volume name and an optional password.

W

WFL

See Work Flow Language.

WFL job

(1) A Work Flow Language (WFL) program, or the execution of such a program. (2) A collection of Work Flow Language (WFL) statements that enable the user to run programs or tasks.

Work Flow Language (WFL)

A Unisys language used for constructing jobs that compile or run programs on A Series systems. WFL includes variables, expressions, and flow-of-control statements that offer the programmer a wide range of capabilities with regard to task control.

Bibliography

- A Series ALGOL Programming Reference Manual, Volume 1: Basic Implementation* (form 8600 0098). Unisys Corporation.
- A Series ALGOL Programming Reference Manual, Volume 2: Product Interfaces* (form 8600 0734). Unisys Corporation.
- A Series BNA Version 1 Operations Guide* (form 8600 0783). Unisys Corporation.
- A Series C Programming Reference Manual* (form 3950 8775). Unisys Corporation.
- A Series CANDE Operations Reference Manual* (form 8600 1500). Unisys Corporation.
- A Series COBOL ANSI-68 Programming Reference Manual* (form 8600 0320). Unisys Corporation.
- A Series COBOL ANSI-74 Programming Reference Manual, Volume 1: Basic Implementation* (form 8600 0296). Unisys Corporation.
- A Series COBOL ANSI-74 Programming Reference Manual, Volume 2: Product Interfaces* (form 8600 0130). Unisys Corporation.
- A Series COBOL ANSI-85 Programming Reference Manual, Volume 1: Basic Implementation* (form 8600 1518). Unisys Corporation.
- A Series COBOL ANSI-85 Programming Reference Manual, Volume 2: Product Interfaces* (form 8600 1526). Unisys Corporation.
- A Series DCALGOL Programming Reference Manual* (form 8600 0841). Unisys Corporation.
- A Series File Attributes Programming Reference Manual* (form 8600 0064). Unisys Corporation.
- A Series FORTRAN Programming Reference Manual* (form 1222691). Unisys Corporation.
- A Series FORTRAN77 Programming Reference Manual* (form 3950 8759). Unisys Corporation.
- A Series GETSTATUS/SETSTATUS Programming Reference Manual* (form 8600 0346). Unisys Corporation.
- A Series I/O Subsystem Programming Guide* (form 8600 0056). Unisys Corporation.
- A Series Interactive Sort (ISORT) Operations Guide* (form 8600 1583). Unisys Corporation.

Bibliography

- A Series KEYEDIOII Programming Reference Manual* (form 8600 0684). Unisys Corporation.
- A Series LINC II Installation & Configuration Guide* (form 3943 4469). Unisys Corporation.
- A Series Mark 4.0 Software Release Capabilities Overview* (form 8600 0015). Unisys Corporation.
- A Series Menu-Assisted Resource Control (MARC) Operations Guide* (form 8600 0403). Unisys Corporation.
- A Series Message Translation Utility (MSGTRANS) Operations Guide* (form 8600 0106). Unisys Corporation. Formerly *A Series Message Translation Utility Operations Guide*.
- A Series MultiLingual System (MLS) Administration, Operations, and Programming Guide* (form 8600 0288). Unisys Corporation.
- A Series NEWP Programming Reference Manual* (form 5044233). Unisys Corporation.
- A Series Operating System Installation Guide* (form 8600 1021). Unisys Corporation.
- A Series Pascal Programming Reference Manual, Volume 1: Basic Implementation* (form 8600 0080). Unisys Corporation.
- A Series Pascal Programming Reference Manual, Volume 2: Product Interfaces* (form 8600 1294). Unisys Corporation.
- A Series PL/I Reference Manual* (form 1169620). Unisys Corporation.
- A Series SORT Language Programming Reference Manual* (form 1169794). Unisys Corporation.
- A Series System Administration Guide* (form 8600 0437). Unisys Corporation.
- A Series System Commands Operations Reference Manual* (form 8600 0395). Unisys Corporation.
- A Series System Configuration Guide* (form 8600 0445). Unisys Corporation.
- A Series System Software Support Reference Manual* (form 8600 0478). Unisys Corporation.
- A Series Systems Functional Overview* (form 8600 0353). Unisys Corporation.
- A Series SYSTEMSTATUS Programming Reference Manual* (form 8600 0452). Unisys Corporation.
- A Series Task Attributes Programming Reference Manual* (form 8600 0502). Unisys Corporation.

A Series Task Management Programming Guide (form 8600 0494). Unisys Corporation.

A Series Work Flow Language (WFL) Programming Reference Manual (form 8600 1047). Unisys Corporation.

Index

A

<A Series system>, COMPATIBILITY request, in FILEDATA, 5-28
<A Series system>, INCOMPATIBILITY request, in FILEDATA, 5-37
ABBREVIATED modifier, in FILEDATA, 5-44
ACCESSMODE attribute, in KEYEDIO, 7-13
ADDED/ALLFILES request, in FILECOPY, 4-6
AGAIN command, in DUMPALL, 3-56
ALGAMA mathematical function, 8-1
ALGOL, use of SORT, 12-3
ALL modifier, in FILEDATA, 5-44
ALOG mathematical function, 8-2
ALOG10 mathematical function, 8-2
<alpha identifier>, DECLARATIONS command, in INTERACTIVEXREF, 6-11
<alphanumeric character>
 basic constructs, in DUMPALL, 3-70
 basic constructs, in FILECOPY, 4-3
 basic constructs, in FILEDATA, 5-1
 tape serial number, in FILEDATA, 5-3
 usercode, in FILEDATA, 5-3
 volume name, in FILEDATA, 5-3
ALTERDATE modifier, in FILEDATA, 5-44
<any character string>, in comment, in PATCH, 9-19
ARCHIVE modifier, in FILEDATA, 5-44
ARCHIVEBACKUP modifier, in FILEDATA, 5-45
ARCHIVEINFO request, in FILEDATA, 5-11
ARCOS mathematical function, 8-2
AREALENGTH modifier, in FILEDATA, 5-45
AREAS attribute
 in FILEDATA, 5-45
 in ISAM, 10-6
AREASIZE attribute

 in FILEDATA, 5-45
 in ISAM, 10-6
AREASUMMARY request, in FILEDATA, 5-13
ARSIN mathematical function, 8-2
ATAN mathematical function, 8-3
ATAN2 mathematical function, 8-3
<attribute field>, RECORD command, 11-7
ATTRIBUTES
 command, in DUMPALL, 3-12, 3-13, 3-56
 request, in FILEDATA, 5-14

B

BACKUP request, in FILEDATA, 5-17
BACKUPSN modifier, in FILEDATA, 5-45
<base>
 \$.INSERT option, in PATCH, 9-19
 \$.MOVE option, in PATCH, 9-25
<baseinc>
 \$.INSERT option, in PATCH, 9-19
 \$.MOVE option, in PATCH, 9-25
binary data in CARDLINE printing, 1-1
block size, in DUMPALL, 3-6
<block size>, old specs, in DUMPALL, 3-78
BLOCKSIZE attribute
 in DUMPALL
 effects on MAXRECSIZE, 3-4
 in FILEDATA, 5-46
 in ISAM, 10-7
BLOCKSIZE attribute, in DUMPALL, 3-6
BLOCKSTRUCTURE
 EXTERNAL, 3-8
 LINKED, 3-9
 VARIABLE, 3-8
 VARIABLEOFFSET, 3-8
 VARIABLE2, 3-8

BLOCKSTRUCTURE modifier, in
 FILEDATA, 5-46
 <Boolean option name>, in PATCH, 9-8
 <Boolean option>, in PATCH, 9-8
 <Boolean-valued attribute>, in DUMPALL,
 3-70
 BRIEF option, in PATCH, 9-8
 <bug option>, in PATCH, 9-29

C

CABS mathematical function, 8-12
 <CANDE option>, in PATCH, 9-30
 CARDLINE utility, 1-1
 printing, 1-1
 CAT command, in DUMPALL, 3-12, 3-14
 CATALOGINFO request, in FILEDATA,
 5-20
 CATALOGUE modifier, in FILEDATA, 5-46
 CCOS mathematical function, 8-12
 CCSVERSION
 file attribute, copying in DUMPALL, 3-11
 modifier, in FILEDATA, 5-46
 CEXP mathematical function, 8-13
 character set translation, in DUMPALL, 3-9
 CHECKERBOARD request, in FILEDATA,
 5-22
 CLOG mathematical function, 8-13
 CM (Change MCP) system command, using
 with RLTABLEGEN, 11-1
 coarse tables
 in ISAM files, 10-4
 size of, 10-5
 in KEYEDIO files, 7-1
 COBOL
 option, in PATCH, 9-8
 use of SORT, 12-3
 COBOL74 option, in PATCH, 9-8, 9-11
 <code file name>, in XREFANALYZER,
 13-1
 coded character set transliteration, in
 DUMPALL, 3-9
 CODEFILEINFO request, in FILEDATA,
 5-24
 CODEVERSION modifier, in FILEDATA,
 5-47
 <CODEVERSION modifier>
 ARCHIVEINFO request, in FILEDATA,
 5-11
 CATALOGINFO request, in FILEDATA,
 5-20

COPYDECK request, in FILEDATA, 5-30
 FILENAMES request, in FILEDATA,
 5-32
 COINCIDENCE command, in
 INTERACTIVEXREF, 6-24
 collating sequences, in SORT, 12-51
 <column>
 RECOGNITION command, in
 RLTABLEGEN, 11-5
 RECORD command, in RLTABLEGEN,
 11-6
 commands
 DUMPALL utility, 3-12
 ATTRIBUTES, 3-12, 3-13
 CAT, 3-12, 3-14
 COPY, 3-12, 3-21
 DEFINE, 3-12
 DUMPMT, 3-12
 FILE, 3-12, 3-13
 HEXDSK, 3-12
 LIBMT, 3-12
 LIST, 3-12, 3-57
 list of, 3-12
 MODE, 3-59
 OPEN, 3-60
 PREVIOUS, 3-61
 PRINT, 3-61
 QUIT, 3-62
 RECORD, 3-62
 SKIP, 3-63
 TEST, 3-12, 3-52
 INTERACTIVEXREF utility
 COINCIDENCE, 6-24
 DECLARATIONS, 6-10
 EXPAND, 6-17
 HELP, 6-20
 LOAD, 6-22
 LOCATE, 6-23
 MERGE, 6-24
 QUALIFY, 6-28
 RANGE, 6-28
 REFERENCE, 6-29
 RESET, 6-32
 SET, 6-32
 STOP, 6-33
 SUMMARY, 6-33
 SYMBOL, 6-34
 TERMINAL, 6-35
 WHAT, 6-36
 WHATFILES, 6-36
 RLTABLEGEN utility
 FIELD, 11-6

ID, 11-3
 RECOGNITION, 11-5
 RECORD, 11-6
 commands, DUMPALL utility
 NEXT, 3-60
 <comment>
 FIELD command, in RLTABLEGEN, 11-6
 ID command, in RLTABLEGEN, 11-3
 patch comment record, in PATCH, 9-6
 \$.GUARD, in PATCH, 9-19
 common constants in mathematical
 functions, 8-15
 COMPARE
 option, in PATCH, 9-8
 utility, 2-1
 running, 2-2
 <compare input>, 2-2
 <compatibility specification>, in
 FILEDATA, 5-28
 COMPATIBILITY request, in FILEDATA,
 5-28
 COMPILE option, in PATCH, 9-8
 <compiler control record>, in PATCH, 9-5
 Complex exponentiation, 8-15
 CONFLICT option, in PATCH, 9-8
 CONTINUE command, in DUMPALL, 3-56
 <control information>, patch delimiter
 record, in PATCH, 9-6
 COPY command
 copying files from a remote host in
 DUMPALL, 3-17, 3-24
 determining KIND attribute for
 DUMPALL, 3-17, 3-24
 in DUMPALL, 3-12, 3-21
 copy from tape, examples in DUMPALL
 labeled tapes, 3-28
 nonstandard labeled tapes, 3-28
 unlabeled tapes, 3-28
 COPYDECK request, in FILEDATA, 5-30
 COS mathematical function, 8-3
 COSH mathematical function, 8-4
 COTAN mathematical function, 8-4
 <count option>, in value option, in PATCH,
 9-9
 <count>
 record range list, in DUMPALL, 3-80
 skip specification, in DUMPALL, 3-82
 CREATED/ACCESSED/UPDATED request,
 in FILECOPY, 4-5
 CREATIONDATE modifier, in FILEDATA,
 5-48
 CRUNCHED modifier, in FILEDATA, 5-48

CSIN mathematical function, 8-14
 CSQRT mathematical function, 8-14
 CYCLE modifier, in FILEDATA, 5-48
 <cycle number>
 \$.CYCLE, in PATCH, 9-28
 \$.FLAG, in PATCH, 9-18
 <cycle option>, in value option in PATCH,
 9-9

D

DARCOS mathematical function, 8-7
 DARSIN mathematical function, 8-8
 <data deck>
 CARDLINE printing, 1-1
 CARDLINE punching, 1-2
 data translation, in DUMPALL, 3-9
 DATABASE modifier, in FILEDATA, 5-48
 databases, in FILEDATA
 creating a reusable database, 5-52
 generation and use of, 5-7
 including raw disk file headers, 5-53
 using database from previous run, 5-48
 DATAN mathematical function, 8-8
 DATAN2 mathematical function, 8-8
 <date>, basic constructs, in FILECOPY, 4-4
 DCOS mathematical function, 8-8
 DCOSH mathematical function, 8-8
 <debug immediate option>, in PATCH,
 9-29
 <debug value option>, in PATCH, 9-29
 <declaration specification>
 DECLARATIONS command, in
 INTERACTIVEXREF, 6-10
 DECLARATIONS command, in
 INTERACTIVEXREF, 6-10
 DEFINE command, in DUMPALL, 3-12,
 3-38
 DEFINEOUTPUT request, in FILEDATA,
 5-31
 DELETE option, in PATCH, 9-8
 DELIMOPT option, in PATCH, 9-8
 <density value> modifier, in FILECOPY,
 4-10
 DEPENDENTSPECS attribute, effects on
 file attributes, 3-4
 DERF mathematical function, 8-9
 DERFC mathematical function, 8-9
 <dest>, COPY command, in DUMPALL,
 3-21

<destination> modifier, in FILECOPY, 4-9, 4-11

DEXP mathematical function, 8-9

DGAMMA mathematical function, 8-9

<digit>

- basic constructs, in DUMPALL, 3-70
- basic constructs, in FILEDATA, 5-2
- identifier specification, in INTERACTIVEXREF, 6-3
- release level, in FILEDATA, 5-3
- serial number, in DUMPALL, 3-73

DIR command, in FILEDATA, 5-58

DIRECTORY modifier, in FILEDATA, 5-48

<directory name>

- basic constructs, in DUMPALL, 3-70
- source from group, in FILECOPY, 4-9

<directory title>

- ATTRIBUTES command, in DUMPALL, 3-13
- basic constructs, in DUMPALL, 3-70
- FILE command, in DUMPALL, 3-13

<discard option>, in PATCH, 9-29

<DISK \$ option>, immediate option, in PATCH, 9-8

<DISK option>, immediate option, in PATCH, 9-8

DLGAMMA mathematical function, 8-9

DLOG mathematical function, 8-10

DLOG10 mathematical function, 8-10

DOCUMENTTYPE modifier, in FILEDATA, 5-49

double-precision

- exponentiation, 8-11
- mathematical functions, 8-7

DSIN mathematical function, 8-10

DSINH mathematical function, 8-10

DSQRT mathematical function, 8-10

DTAN mathematical function, 8-10

DTANH mathematical function, 8-11

DUMP option, in PATCH, 9-8

DUMPALL utility, 3-1

- commands, 3-12
- ATTRIBUTES, 3-13
- CAT, 3-14
- COPY, 3-21
- DEFINE, 3-38
- DMPMT, 3-39
- FILE, 3-13
- HEXDSK, 3-43
- LIBMT, 3-45
- LIST, 3-46
- list of, 3-12

- TEST, 3-52
- error detection, 3-64
- I/O exceptions, 3-67
- interactive commands, 3-55
- AGAIN, 3-56
- ATTRIBUTE, 3-56
- CONTINUE, 3-56
- FILE, 3-56
- LIST, 3-57
- MODE, 3-59
- NEXT, 3-60
- OPEN, 3-60
- PREVIOUS, 3-61
- PRINT, 3-61
- QUIT, 3-62
- RECORD, 3-62
- SKIP, 3-63
- modes

 - card, 3-65
 - interactive, 3-66
 - parameter, 3-64
 - TASKVALUE attribute, 3-64

DUMPMT command, in DUMPALL, 3-12, 3-39

E

<EBCDIC string character>

- basic constructs, in FILECOPY, 4-3
- basic constructs, in FILEDATA, 5-2
- in DUMPALL, 3-70
- string, in FILECOPY, 4-9

<end option>, in PATCH, 9-29

<EOF option>, immediate option, in PATCH, 9-8

<equate option>, in PATCH, 9-29

ERF mathematical function, 8-4

ERFC mathematical function, 8-5

ERRLIST option, in PATCH, 9-8

error detection, for DUMPALL tasks, 3-64

EXECUTE option, in PATCH, 9-8

EXP mathematical function, 8-5

EXPAND command, in INTERACTIVEXREF, 6-17

<expand>, DECLARATIONS command, in INTERACTIVEXREF, 6-11

EXPIRED request, in FILECOPY, 4-7

EXTMODE

- attribute, effects on translation in DUMPALL, 3-11
- modifier, in FILEDATA, 5-49

F

- <family name>
 - ARCHIVEINFO request, in FILEDATA, 5-11
 - AREASUMMARY request, in FILEDATA, 5-13
 - ATTRIBUTES request, in FILEDATA, 5-14
 - BACKUP request, in FILEDATA, 5-17
 - basic constructs, in DUMPALL, 3-71
 - basic constructs, in FILECOPY, 4-4
 - basic constructs, in FILEDATA, 5-2
 - CATALOGINFO request, in FILEDATA, 5-20
 - CHECKERBOARD request, in FILEDATA, 5-22
 - CODEFILEINFO request, in FILEDATA, 5-24
 - COMPATIBILITY request, in FILEDATA, 5-28
 - COPYDECK request, in FILEDATA, 5-30
 - FILENAMES request, in FILEDATA, 5-32
 - HEADERCONTENTS request, in FILEDATA, 5-35
 - INCOMPATIBILITY request, in FILEDATA, 5-37
 - NOREPORTS request, in FILEDATA, 5-39
 - old packdir syntax, in FILEDATA, 5-56
 - STRUCTUREMAP request, in FILEDATA, 5-41
- <family option>, CHECKERBOARD request, in FILEDATA, 5-22
- FAMILYNAME modifier, in FILEDATA, 5-49
- FIELD command, in RLTABLEGEN, 11-6
- <field definition>
 - basic constructs, in DUMPALL, 3-73
 - DEFINE command, in DUMPALL, 3-38
 - format definition, in DUMPALL, 3-77
- <field length>
 - field definition, in DUMPALL, 3-73
 - in COMPARE, 2-2
- <field mnemonic>, format definition, in DUMPALL, 3-77
- <field offset>, field definition, in DUMPALL, 3-73
- <field type>, field definition, in DUMPALL, 3-73
- <file attribute assignment>, in DUMPALL, 3-71
- <file attribute>
 - DMPMT command, in DUMPALL, 3-39
 - LIST command, in DUMPALL, 3-46
 - OPEN command, in DUMPALL, 3-60
 - TEST command, in DUMPALL, 3-52
- file attributes
 - BLOCKSIZE, in DUMPALL, 3-6
 - CCSVERSION, copying in DUMPALL, 3-11
 - determining, in DUMPALL/inx>, 3-4
 - MAXRECSIZE, in DUMPALL, 3-6
 - specifying values, in DUMPALL, 3-5
 - structural, in DUMPALL, 3-5
 - using with DUMPALL, 3-3
- <file attributes>
 - ATTRIBUTES request, in FILEDATA, 5-14
 - CAT command, in DUMPALL, 3-14
 - CODEFILEINFO request, in FILEDATA, 5-24
 - COPY command, in DUMPALL, 3-21
 - definition, in FILEDATA, 5-15
 - FILE command, in DUMPALL, 3-12, 3-13, 3-56
- <file equation>
 - CANDE command, in PATCH, 9-2
 - for WFL job, in PATCH, 9-1
- <file ID>,\$.INSERT option, in PATCH, 9-19
- file modifiers
 - ABBREVIATED, 5-44
 - ALL, 5-44
 - ALTERDATE, 5-44
 - ARCHIVE, 5-44
 - ARCHIVEBACKUP, 5-45
 - AREALENGTH<inx>, 5-45
 - AREAS, 5-45
 - AREASIZE, 5-45
 - BACKUPSN, 5-45
 - BLOCKSIZE, 5-46
 - BLOCKSTRUCTURE, 5-46
 - CATALOGUE, 5-46
 - CCSVERSION, 5-46
 - CODEVERSION, 5-47
 - CREATIONDATE, 5-48
 - CRUNCHED, 5-48
 - CYCLE, 5-48
 - DATABASE, 5-48
 - DIRECTORY, 5-48
 - DOCUMENTTYPE, 5-49

- EXTMODE, 5-49
- FAMILYNAME, 5-49
- FILEKIND, 5-49
- FILELENGTH, 5-49
- FILEORGANIZATION, 5-49
- FILESTRUCTURE, 5-49
- FILETYPE, 5-50
- FRAMESIZE, in FILEDATA, 5-50
- GUARDFILE, 5-50
- IDENTITY, 5-50
- INTMODE, 5-50
- LASTACCESSDATE, 4-7, 4-16, 5-51
- LASTRECORD, 5-51
- LEVEL, 5-51
- LICENSEKEY, 5-51
- LINEWIDTH, 5-51
- LOCKEDFILE, 5-51
- MAXRECSIZE, 5-52
- MINRECSIZE, 5-52
- NAMESONLY, 5-52
- NEWDATABASE, 5-52
- NONRESIDENTONLY, 5-52
- NOTE, 5-52
- PACKNAME, 5-52
- PAGESIZE modifier, 5-53
- PERMITTEDACTIONS, 5-53
- RAWHEADERS, 5-53
- RELEASEID, 5-53
- RESIDENTONLY, 5-53
- SAVEFACTOR, 5-53
- SECURITY, 5-53
- SUMMARY, 5-22
- TAPE, 5-32
- TIMESTAMP, 5-54
- TITLE, 5-54
- TOTALSECTORS, 5-54
- UNITS, 5-54
- USERINFO, 5-54
- VERSION, 5-54
- WARNINGS, 5-54
- <file name 1>, in COMPARE, 2-2
- <file name 2>, in COMPARE, 2-2
- <file name>
 - basic constructs, in FILECOPYY, 4-3
 - basic constructs, in FILEDATA, 5-2
 - COINCIDENCE command, in INTERACTIVEXREF, 6-24
 - DECLARATIONS command, in INTERACTIVEXREF, 6-11
 - HEXDSK command, in DUMPALL, 3-43
 - LIST command
 - in DUMPALL, 3-46
 - LIST command, in INTERACTIVEXREF, 6-21
 - LOAD command, in INTERACTIVEXREF, 6-22
 - MERGE command, in INTERACTIVEXREF, 6-24
 - old packdir syntax, in FILEDATA, 5-56
 - output option, in FILEDATA, 5-5
 - REFERENCE command, in INTERACTIVEXREF, 6-29
 - source file name, in FILECOPYY, 4-9
 - source from group, in FILECOPYY, 4-9
 - SYMBOL command, in INTERACTIVEXREF, 6-34
 - \$.EQUATE, in PATCH, 9-32
 - <FILE option>, immediate option, in PATCH, 9-8
 - <file specification>
 - CAT command, in DUMPALL, 3-14
 - in FILECOPYY, 4-8
 - modifier, in FILECOPYY, 4-8, 4-10
 - <file title>
 - ADDED/ALLFILES request, in FILECOPYY, 4-6
 - ARCHIVEINFO request, in FILEDATA, 5-11
 - ATTRIBUTES command, in DUMPALL, 3-13
 - ATTRIBUTES request, in FILEDATA, 5-14
 - BACKUP request, in FILEDATA, 5-17
 - basic constructs
 - in DUMPALL, 3-71
 - in FILECOPYY, 4-4
 - in FILEDATA, 5-2
 - CAT command, in DUMPALL, 3-14
 - CATALOGINFO request, in FILEDATA, 5-20
 - CHECKERBOARD request, in FILEDATA, 5-22
 - CODEFILEINFO request, in FILEDATA, 5-24
 - COMPATIBILITY request, in FILEDATA, 5-28
 - COPY command, in DUMPALL, 3-21
 - COPYDECK request, in FILEDATA, 5-30
 - FILE command, in DUMPALL, 3-13
 - FILENAMES request, in FILEDATA, 5-32
 - HEADERCONTENTS request, in FILEDATA, 5-35

- INCOMPATIBILITY request, in
 - FILEDATA, 5-37
- NOREPORTS request, in FILEDATA, 5-39
- OPEN command, in DUMPALL, 3-60
- output option, in FILEDATA, 5-5
- STRUCTUREMAP request, in
 - FILEDATA, 5-41
- TEST command, in DUMPALL, 3-52
- \$.FILE, in PATCH, 9-18
- <filecopy modifier>
 - ADDED/ALLFILES request, in
 - FILECOPY, 4-6
 - CREATED/ACCESSED/UPDATED request, in FILECOPY, 4-5
 - description of, in FILECOPY, 4-8
 - EXPIRED request, in FILECOPY, 4-7
 - input to FILECOPY, 4-3
- <filecopy task request>, input to
 - FILECOPY, 4-3
- FILECOPY utility
 - description of, 4-1
 - index files, 4-18
 - input to, 4-2
 - options, 4-12
 - task requests, 4-5, 4-8
 - ADDED/ALLFILES, 4-6
 - CREATED/ACCESSED/UPDATED, 4-5
 - EXPIRED, 4-7
- FILEDATA utility
 - databases
 - creating a reusable database, 5-52
 - generation and use of, 5-7
 - including raw disk file headers, 5-53
 - using database from previous run, 5-48
 - description of, 5-1
 - error reporting, 5-6
 - input to, 5-5
 - methods of running, 5-5
 - modifier meanings, 5-44
 - numeric report requests, 5-55
 - old packdir syntax, 5-56
 - reports, COMPATIBILITY request, 5-28
 - reports, INCOMPATIBILITY request, 5-37
 - system commands
 - DIR command, 5-58
 - TDIR command, 5-59
 - task requests, 5-10
 - ARCHIVEINFO, 5-11
 - AREASUMMARY, 5-13
 - ATTRIBUTES, 5-14
 - BACKUP, 5-17
 - CATALOGINFO, 5-20
 - CHECKERBOARD, 5-22
 - CODEFILEINFO, 5-24
 - COPYDECK, 5-30
 - DEFINEOUTPUT, 5-31
 - FILENAMES, 5-32
 - HEADERCONTENTS, 5-35
 - NOREPORTS, 5-39
 - STRUCTUREMAP, 5-41
 - TAPEDIR, 5-43
- FILEKIND modifier, in FILEDATA, 5-49
- <filekind>, option, in FILECOPY, 4-12
- FILELENGTH modifier, in FILEDATA, 5-49
- FILENAMES request, in FILEDATA, 5-32
- FILEORGANIZATION
 - attributes
 - INDEXED, 7-1
 - INDEXEDNOTRESTRICTED, 7-1
 - KEYEDIOII, 7-1
 - KEYEDIOISET, 7-1
 - PLIISAM, 10-1
 - modifier, in FILEDATA, 5-49
- FILESTRUCTURE
 - modifier, in FILEDATA, 5-49
 - value used, in DUMPALL, 3-5
- FILETYPE modifier, in FILEDATA, 5-50
- fine tables
 - in ISAM files, 10-3
 - size of, 10-5
 - in KEYEDIO files, 7-2
- <first>
 - \$.INSERT option, in PATCH, 9-19
 - \$.MOVE option, in PATCH, 9-25
- <flag option>, in value option, in PATCH, 9-9
- <format definition>
 - basic constructs, in DUMPALL, 3-77
 - DEFINE command, in DUMPALL, 3-38
 - DMPMT command, in DUMPALL, 3-39
 - LIST command, in DUMPALL, 3-46, 3-57
 - MODE command, 3-59
 - PRINT command, 3-61
 - TEST command, in DUMPALL, 3-52
- <format mnemonic>, format definition, in
 - DUMPALL, 3-77
- format of tapes in DUMPALL
 - labeled, 3-85
 - unlabeled, 3-84

FORTRAN, using with
 INTERACTIVEXREF, 6-38

FORTRAN77, using with
 INTERACTIVEXREF, 6-38

FRAMESIZE
 file attribute, in DUMPALL, 3-6
 modifier, in FILEDATA, 5-50

<from clause> modifier, in FILECOPY, 4-9,
 4-11

G

GAMMA mathematical function, 8-5

<GUARD option>, patch control record, in
 PATCH, 9-8

GUARDFILE DOES NOT EXIST message,
 in FILEDATA, 5-50

GUARDFILE IS NOT VISIBLE message, in
 FILEDATA, 5-50

GUARDFILE modifier, in FILEDATA, 5-50

H

HEADERCONTENTS request, in
 FILEDATA, 5-35

HELP command, in INTERACTIVEXREF,
 6-20

<hex digit>, basic constructs, in
 FILEDATA, 5-2

<hex int>, DECLARATIONS command, in
 INTERACTIVEXREF, 6-11

<hex string>, basic constructs, in
 FILEDATA, 5-2

HEXDSK command, in DUMPALL, 3-12,
 3-43

<host name>, basic constructs, in
 DUMPALL, 3-71

HOSTNAME attribute
 COPY command, in DUMPALL, 3-17,
 3-24
 LIST command, in DUMPALL, 3-47, 3-53

<hyphen>
 basic constructs, in DUMPALL, 3-71
 option, in FILECOPY, 4-12

I

I/O exceptions, controlling, in DUMPALL,
 3-67

ID command, in RLTABLEGEN, 11-3

<identifier qualification>
 basic constructs, in INTERACTIVEXREF,
 6-3

QUALIFY command, in
 INTERACTIVEXREF, 6-28

<identifier specification>
 basic constructs, in INTERACTIVEXREF,
 6-3

COINCIDENCE command, in
 INTERACTIVEXREF, 6-24

EXPAND command, in
 INTERACTIVEXREF, 6-17

LOCATE command, in
 INTERACTIVEXREF, 6-23

MERGE command, in
 INTERACTIVEXREF, 6-24

REFERENCE command, in
 INTERACTIVEXREF, 6-29

SUMMARY command, in
 INTERACTIVEXREF, 6-33

<identifier>
 basic constructs, in DUMPALL, 3-72
 basic constructs, in FILECOPY, 4-3
 basic constructs, in FILEDATA, 5-2
 basic constructs, in INTERACTIVEXREF,
 6-3

DECLARATIONS command, in
 INTERACTIVEXREF, 6-10

from clause, in FILECOPY, 4-9

IDENTITY modifier, in FILEDATA, 5-50

<immediate option>, in PATCH, 9-8

INCOMPATIBILITY request, in FILEDATA,
 5-37

<incompatibility specification>, in
 FILEDATA, 5-37

<increment>
 \$.INSERT option, in PATCH, 9-19
 \$.MOVE option, in PATCH, 9-25

index files, in FILECOPY, 4-18

indexed sequential-access method, 10-1
 data record links, 10-4
 files, copying, 3-37
 I/O result information, 10-17
 using primitive ISAM, 10-18

invocation of
 primitive method, 10-1
 standard method, 10-2

- limitations on use, 10-8
- management of overflow areas, 10-4
- planning files, 10-5, 10-6, 10-7
 - BLOCKSIZE attribute, 10-7
 - coarse table size, 10-5
 - EXCLUSIVE attribute, 10-7
 - fine table ratio, 10-7
 - fine table size, 10-5
 - INFO record size, 10-6
 - key length, 10-8
 - key offset, 10-8
 - maximum number of records, 10-5
 - practical considerations, 10-8
- procedures, 10-9
 - ISCLOSE, 10-12
 - ISDELETE, 10-17
 - ISKEYWRITE, 10-16
 - ISOPEN, 10-9
 - ISREAD, 10-13
 - ISREADNEXT, 10-15
 - ISREWRITE, 10-15
 - ISWRITE, 10-14
- structure of files
 - data overflow area, 10-3
 - prime data area, 10-3
 - tables for locating data
 - coarse tables, 10-3
 - fine tables, 10-3
- <input record>, for WFL job, in PATCH, 9-1
- <input to filecopy>, in FILECOPY WFL job, 4-1
- <INSERT option>, immediate option, in PATCH, 9-8
- <integer-valued attribute>
 - basic constructs, in DUMPALL, 3-72
- <integer>
 - ARCHIVEINFO request, in FILEDATA, 5-11
 - AREASUMMARY request, in FILEDATA, 5-13
 - ATTRIBUTES request, in FILEDATA, 5-14
 - BACKUP request, in FILEDATA, 5-17
 - basic constructs, in DUMPALL, 3-72
 - basic constructs, in FILECOPY, 4-4
 - basic constructs, in FILEDATA, 5-2
 - CARDLINE printing, 1-1
 - CAT command, in DUMPALL, 3-14
 - CATALOGINFO request, in FILEDATA, 5-20
 - CHECKERBOARD request, in FILEDATA, 5-22
 - COINCIDENCE command, in INTERACTIVEXREF, 6-24
 - COMPATIBILITY request, in FILEDATA, 5-28
 - COPY command, in DUMPALL, 3-21
 - COPYDECK request, in FILEDATA, 5-30
 - DECLARATIONS command, in INTERACTIVEXREF, 6-11
 - DEFINEOUTPUT request, in FILEDATA, 5-31
 - destination, in FILECOPY, 4-9
 - DMPMT command, in DUMPALL, 3-39
 - EXPAND command, in INTERACTIVEXREF, 6-11
 - EXPIRED request, in FILECOPY, 4-7
 - FILENAMES request, in FILEDATA, 5-32
 - HEADERCONTENTS request, in FILEDATA, 5-35
 - INCOMPATIBILITY request, in FILEDATA, 5-37
 - LIST command, in DUMPALL, 3-46, 3-57
 - LIST command, in INTERACTIVEXREF, 6-21
 - MERGE command, in INTERACTIVEXREF, 6-24
 - numeric report requests, in FILEDATA, 5-55
 - REFERENCE command, in INTERACTIVEXREF, 6-29
 - serial number list, in FILECOPY, 4-9
 - STRUCTUREMAP request, in FILEDATA, 5-41
 - TERMINAL command, in INTERACTIVEXREF, 6-35
 - TEST command, in DUMPALL, 3-52
 - unit number, in FILEDATA, 5-3
 - \$.BUG, in PATCH, 9-30
 - \$.GUARD, in PATCH, 9-19
 - \$.LISTN, in PATCH, 9-23
- INTERACTIVEXREF utility commands
 - COINCIDENCE, 6-24
 - DECLARATIONS, 6-10
 - EXPAND, 6-17
 - HELP, 6-20
 - LIST, 6-21
 - list of, 6-9
 - LOAD, 6-22
 - LOCATE, 6-23

- MERGE, 6-24
 - QUALIFY, 6-28
 - RANGE, 6-28
 - REFERENCE, 6-29
 - RESET, 6-32
 - SET, 6-32
 - STOP, 6-33
 - SUMMARY, 6-33
 - SYMBOL, 6-34
 - TERMINAL, 6-35
 - WHAT, 6-36
 - WHATFILES, 6-36
 - description of, 6-1
 - identifier specification, 6-3
 - producing XREFFILES, 6-1
 - range specification, 6-7
 - using with FORTRAN, 6-38
 - using with FORTRAN77, 6-38
 - using with improperly sequenced source, 6-37
 - using with Pascal, 6-38
 - <interface identifier>
 - LIST command, in INTERACTIVEXREF, 6-21
 - Interprogram Communication, in ISAM, 10-9
 - INTMODE
 - attribute
 - effects on translation in DUMPALL, 3-10
 - modifier, in FILEDATA, 5-50
 - <intname>
 - \$.DISK, in PATCH, 9-18
 - \$.INSERT option, in PATCH, 9-19
 - \$.PATCHDECK, in PATCH, 9-18
 - ISAM, (See indexed sequential-access method)
 - ISOPEN procedure, 10-9
 - types of files, 10-2
 - ISAMKEYS attribute, in KEYEDIO, 7-12
 - ISCLOSE procedure, in ISAM, 10-12
 - ISDELETE procedure, in ISAM, 10-17
 - ISKEYWRITE procedure, in ISAM, 10-16
 - ISMCLOSE procedure, in KEYEDIO, 7-16
 - ISMDELETE procedure, in KEYEDIO, 7-22
 - ISMGETKEYSTRUCTURE procedure, in KEYEDIO, 7-14
 - ISMOPEN procedure, in KEYEDIO, 7-15
 - ISMRANDOMREAD procedure, in KEYEDIO, 7-20
 - ISMRANDOMWRITE procedure, in KEYEDIO, 7-20
 - ISMREWRITE procedure, in KEYEDIO, 7-21
 - ISMSEQUENTIALREAD procedure, in KEYEDIO, 7-19
 - ISMSEQUENTIALWRITE procedure, in KEYEDIO, 7-18
 - ISMSETUPLIMIT procedure, in KEYEDIO, 7-23
 - ISMSTART procedure, in KEYEDIO, 7-17
 - ISREAD procedure, in ISAM, 10-13
 - ISREADNEXT procedure, in ISAM, 10-15
 - ISREWRITE procedure, in ISAM, 10-15
 - ISWRITE procedure, in ISAM, 10-14
- ## K
- KEYEDIO, 7-1
 - file structure, 7-24
 - block information layout, 7-28
 - course table layout, 7-29
 - fine table layout, 7-29
 - inserting keys, 7-32
 - key info table layout, 7-30
 - logical layout, 7-30
 - segment 0 (zero), 7-24
 - files
 - management, 7-2
 - GENERALSUPPORT attributes, 7-10
 - indexed attributes, 7-4
 - BLOCKSIZE, 7-7
 - BUFFERS, 7-5
 - FILEORGANIZATION, 7-5
 - internal attributes
 - ACCESSMODE, 7-13
 - ISAMKEYS, 7-12
 - library management, 7-2
 - installing, 7-3
 - removing, 7-3
 - physical file structure, 7-1
 - coarse tables, 7-1
 - data area, 7-2
 - fine tables, 7-2
 - locating data, 7-2
 - procedures
 - ISMCLOSE, 7-16
 - ISMDELETE, 7-22
 - ISMGETKEYSTRUCTURE, 7-14
 - ISMOPEN, 7-15
 - ISMRANDOMREAD, 7-20
 - ISMRANDOMWRITE, 7-20
 - ISMREWRITE, 7-21

ISMSEQUENTIALREAD, 7-19
 ISMSEQUENTIALWRITE, 7-18
 ISMSETUPLIMIT, 7-23
 ISMSTART, 7-17
 program interface, 7-3
 recovery, 7-35
 messages, 7-35
 warnings, 7-35
 KEYEDIOII files, copying, 3-37
 KIND attribute, determining for COPY
 command in DUMPALL, 3-17, 3-24

L

LABEL option, in PATCH, 9-8
 <last>
 \$.INSERT option, in PATCH, 9-19
 \$.MOVE option, in PATCH, 9-25
 LASTACCESSDATE modifier, 4-7, 4-16
 LASTACCESSDATE modifier, in
 FILEDATA, 5-51
 LASTRECORD IS UNKNOWN message
 FILEDATA, 5-51
 LASTRECORD modifier, in FILEDATA,
 5-51
 <length>, RECORD command, in
 RLTABLEGEN, 11-6
 <letter>
 basic constructs, in INTERACTIVEXREF,
 6-3
 option, in FILECOPY, 4-12
 serial number, in DUMPALL, 3-73
 LEVEL modifier, in FILEDATA, 5-51
 LIBMT command, in DUMPALL, 3-12, 3-45
 library maintenance tape restrictions
 copying, 3-45
 listing, 3-45
 LICENSEKEY modifier, in FILEDATA, 5-51
 LINEWIDTH modifier, in FILEDATA, 5-51
 LIST command
 in INTERACTIVEXREF, 6-21
 listing files from a remote host in
 DUMPALL, 3-47, 3-53
 LIST command, in DUMPALL, 3-12, 3-46,
 3-57
 LISTD option, in PATCH, 9-8
 LISTI option, in PATCH, 9-8
 listing files from a remote host in
 DUMPALL, 3-47, 3-53
 LISTN option, in PATCH, 9-8
 LISTP option, in PATCH, 9-8

<literal compiler record>, in PATCH, 9-5
 LOAD command, in INTERACTIVEXREF,
 6-22
 LOCATE command, in
 INTERACTIVEXREF, 6-23
 LOCKEDFILE modifier, in FILEDATA, 5-51

M

MARK option, in PATCH, 9-8
 MARKBLANK option, in PATCH, 9-8
 master control program
 in SORT, 12-1
 LABEL recognition routine, 11-1
 mathematical functions
 ALGAMA, 8-1
 ALOG, 8-2
 ALOG10, 8-2
 ARCOS, 8-2
 ARSIN, 8-2
 ATAN, 8-3
 ATAN2, 8-3
 CABS, 8-12
 CCOS, 8-12
 CEXP, 8-13
 CLOG, 8-13
 common constants, 8-15
 complex functions, 8-11
 COS, 8-3
 COSH, 8-4
 COTAN, 8-4
 CSIN, 8-14
 CSQRT, 8-14
 DARCOS, 8-7
 DARSIN, 8-8
 DATAN, 8-8
 DATAN2, 8-8
 DCOS, 8-8
 DCOSH, 8-8
 DERF, 8-9
 DERFC, 8-9
 description of, 8-1
 DEXP, 8-9
 DGAMMA, 8-9
 DLGAMMA, 8-9
 DLOG, 8-10
 DLOG10, 8-10
 DSIN, 8-10
 DSINH, 8-10
 DSQRT, 8-10
 DTAN, 8-10

DTANH, 8-11
 ERF, 8-4
 ERF, 8-5
 EXP, 8-5
 function names equation table, 8-17
 GAMMA, 8-5
 permissible argument range, 8-16
 RANDOM, 8-5
 SIN, 8-6
 single-precision functions, 8-1
 SINH, 8-6
 SQRT, 8-7
 TAN, 8-7
 TANH, 8-7
 <maximum errors>, in COMPARE, 2-2
 MAXRECSIZE
 attribute
 in DUMPALL, 3-4, 3-6, 3-78
 in ISAM, 10-7
 modifier, in FILEDATA, 5-52
 <maxrecsize>, old specs, in DUMPALL,
 3-78
 MCP, (See master control program)
 MERGE command, in INTERACTIVEXREF,
 6-24
 MINRECSIZE
 attribute, in ISAM, 10-7
 modifier, in FILEDATA, 5-52
 <mnemonic>
 field definition, in DUMPALL, 3-73
 format definition, in DUMPALL, 3-77
 MODE command, in DUMPALL, 3-59
 <mode>, RECORD command, 11-7
 modes, DUMPALL
 card, 3-65
 interactive, 3-65
 parameter, 3-64
 <MOVE option>, immediate option, in
 PATCH, 9-8

N

<name>
 basic constructs, in DUMPALL, 3-72
 directory name, in DUMPALL, 3-70
 file title, in DUMPALL, 3-71
 host name, in DUMPALL, 3-71
 usercode, in DUMPALL, 3-73
 NAMESONLY modifier, in FILEDATA, 5-52
 NDII option, in PATCH, 9-8
 NEW option, in PATCH, 9-8

NEWDATABASE modifier, in FILEDATA,
 5-52
 NEXT command, in DUMPALL, 3-60
 <nonquote EBCDIC character>, 5-3
 NONRESIDENTONLY modifier, in
 FILEDATA, 5-52
 NOREPORTS request, in FILEDATA, 5-39
 NOTE modifier, in FILEDATA, 5-52
 <number>
 COINCIDENCE command, in
 INTERACTIVEXREF, 6-24
 DECLARATIONS command, in
 INTERACTIVEXREF, 6-11
 field definition, in DUMPALL, 3-73
 MERGE command, in
 INTERACTIVEXREF, 6-24
 RECORD command, in DUMPALL, 3-62
 REFERENCE command, in
 INTERACTIVEXREF, 6-29
 SKIP command, 3-63
 \$.LABEL, in PATCH, 9-21
 \$.TOTAL, in PATCH, 9-28
 <numeric report request>
 definition, in FILEDATA, 5-55
 DIR command, in FILEDATA, 5-58
 parameter list, in FILEDATA, 5-5

O

<old packdir syntax>
 DIR command, in FILEDATA, 5-58
 in FILEDATA, 5-56
 parameter list, in FILEDATA, 5-5
 <old specs>
 basic constructs, in DUMPALL, 3-78
 CAT command, in DUMPALL, 3-14
 COPY command, in DUMPALL, 3-21
 LIST command, in DUMPALL, 3-46
 OPEN command, in DUMPALL, 3-60
 TEST command, in DUMPALL, 3-52
 OPEN command, in DUMPALL, 3-60
 options in FILECOPY, 4-12
 OUT option, in PATCH, 9-8
 <output medium> modifier, in FILECOPY,
 4-9
 <output option>
 ARCHIVEINFO request, in FILEDATA,
 5-11
 AREASUMMARY request, in FILEDATA,
 5-13

- ATTRIBUTES request, in FILEDATA, 5-14
 - CATALOGINFO request, in FILEDATA, 5-20
 - CODEFILEINFO request, in FILEDATA, 5-24
 - COMPATIBILITY request, in FILEDATA, 5-28
 - COPYDECK request, in FILEDATA, 5-30
 - FILENAMES request, in FILEDATA, 5-32
 - HEADERCONTENTS request, in FILEDATA, 5-35
 - INCOMPATIBILITY request, in FILEDATA, 5-37
 - STRUCTUREMAP request, in FILEDATA, 5-41
- P**
- PACKNAME modifier, in FILEDATA, 5-52
 - PAGESIZE modifier, in FILEDATA, 5-53
 - <parameter list>
 - DIR command, in FILEDATA, 5-58
 - in FILEDATA, 5-5
 - PASCAL option, in PATCH, 9-8
 - Pascal, using with INTERACTIVEXREF, 6-38
 - <patch comment record>, in PATCH, 9-6
 - <patch control record>, in PATCH, 9-8
 - <patch delimiter record>, in PATCH, 9-6
 - <patch patch record>, in PATCH, 9-6
 - <patch records to the moved material>
 - \$.MOVE option, in PATCH, 9-25
 - PATCH utility
 - compiler control options, 9-4
 - control record categories, 9-3
 - control records, 9-4
 - \$. RECORDS, 9-8
 - \$\$ RECORDS, 9-7
 - \$ RECORDS, 9-6
 - \$. RECORDS, 9-6
 - \$\$& RECORDS, 9-5
 - \$\$# RECORDS, 9-6
 - debug options, 9-29
 - \$.BUG, 9-30
 - \$.CANDE, 9-30
 - \$.DISCARD, 9-31
 - \$.END, 9-31
 - \$.EQUATE, 9-32
 - \$.PDUMP, 9-32
 - examples of input, 9-33
 - explanation of, 9-1
 - files used by, 9-3
 - required conditions, 9-3
 - running, 9-1
 - \$. OPTIONS, 9-10
 - \$.BRIEF, 9-10
 - \$.COBOL, 9-11
 - \$.COBOL74, 9-11
 - \$.COMPARE, 9-11
 - \$.COMPILE, 9-12
 - \$.CONFLICT, 9-12
 - \$.COUNT, 9-13
 - \$.CYCLE, 9-13, 9-28
 - \$.DELETE, 9-13
 - \$.DELIMOPT, 9-14
 - \$.DISK, 9-15
 - \$.DISK \$, 9-18
 - \$.DUMP, 9-16
 - \$.EOF, 9-17
 - \$.ERRLIST, 9-17
 - \$.EXECUTE, 9-17
 - \$.FILE, 9-18
 - \$.FLAG, 9-18
 - \$.GUARD, 9-19
 - \$.INSERT, 9-19
 - \$.LABEL, 9-21
 - \$.LIST, 9-22
 - \$.LISTD, 9-22
 - \$.LISTI, 9-22
 - \$.LISTN, 9-23
 - \$.LISTP, 9-23
 - \$.MARK, 9-23
 - \$.MARKBLANK, 9-24
 - \$.MOVE, 9-25
 - \$.NDLII, 9-26
 - \$.NEW, 9-26
 - \$.OUT, 9-26
 - \$.PASCAL, 9-27
 - \$.PATCHDECK, 9-18
 - \$.RPG, 9-27
 - \$.SINGLE, 9-27
 - \$.SQUASH, 9-28
 - \$.TOTAL, 9-28
 - \$.VERSION, 9-28
 - <patch WFL record>, in PATCH, 9-7
 - <patch>, 9-6
 - <PATCHDECK option>, immediate option, in PATCH, 9-8
 - <pdump option>, in PATCH, 9-30
 - permissible argument range, mathematical functions, 8-16

Index

- PERMITTEDACTIONS modifier, in FILEDATA, 5-53
 - PLIISAM files
 - copying, 3-37
 - <pointer-valued attribute>
 - basic constructs, in DUMPALL, 3-72
 - PREVIOUS command, in DUMPALL, 3-61
 - PRINT command, in DUMPALL, 3-61
 - <print option>
 - basic construct, in DUMPALL, 3-79
 - DMPMT command, in DUMPALL, 3-39
 - LIST command, in DUMPALL, 3-46, 3-57
 - TEST command, in DUMPALL, 3-52
 - print options, in DUMPALL, 3-79
 - <procedure identifier>
 - LIST command, in INTERACTIVEXREF, 6-21
 - procedure specification, in INTERACTIVEXREF, 6-4
 - <procedure specification>
 - basic constructs, in INTERACTIVEXREF, 6-4
 - <procedure specification>, in INTERACTIVEXREF, 6-7
- Q**
- QUALIFY command, in INTERACTIVEXREF, 6-28
 - QUIT command, in DUMPALL, 3-62
- R**
- railroad diagrams, explanation of, A-1
 - RANDOM mathematical function, 8-5
 - RANGE command, in INTERACTIVEXREF, 6-28
 - <range specification>
 - basic constructs, in INTERACTIVEXREF, 6-7
 - COINCIDENCE command, in INTERACTIVEXREF, 6-24
 - DECLARATIONS command, in INTERACTIVEXREF, 6-11
 - MERGE command, in INTERACTIVEXREF, 6-24
 - RANGE command, in INTERACTIVEXREF, 6-28
 - REFERENCE command, in INTERACTIVEXREF, 6-29
 - <range>
 - CHECKERBOARD request, in FILEDATA, 5-22
 - RAWHEADERS modifier, in FILEDATA, 5-53
 - RECOGNITION command, in RLTABLEGEN utility, 11-5
 - RECORD command, in DUMPALL, 3-62
 - RECORD command, in RLTABLEGEN utility, 11-6
 - <record number>
 - RECORD command, in RLTABLEGEN, 11-6
 - record range list, in DUMPALL, 3-80
 - skip specification, in DUMPALL, 3-82
 - <record range list>
 - basic constructs, in DUMPALL, 3-80
 - CAT command, in DUMPALL, 3-14
 - COPY command, in DUMPALL, 3-21
 - DMPMT command, in DUMPALL, 3-39
 - HEXDSK command, in DUMPALL, 3-43
 - LIST command, in DUMPALL, 3-46, 3-57
 - TEST command, in DUMPALL, 3-52
 - record size, in DUMPALL, 3-6
 - REFERENCE command, in INTERACTIVEXREF, 6-29
 - <reference>, DECLARATIONS command, in INTERACTIVEXREF, 6-11
 - <relation>
 - basic constructs, in FILEDATA, 5-3
 - CODEFILEINFO request, in FILEDATA, 5-24
 - <release level>
 - basic constructs, in FILEDATA, 5-3
 - CODEFILEINFO request, in FILEDATA, 5-24
 - RELEASEID modifier, in FILEDATA, 5-53
 - reports, in FILEDATA, 5-1
 - REQUESTED FILE OR DIRECTORY NOT FOUND message, 5-6
 - RESET command, in INTERACTIVEXREF, 6-32
 - RESIDENTONLY modifier, in FILEDATA, 5-53
 - RLTABLEGEN utility
 - commands
 - FIELD, 11-6
 - ID, 11-3
 - RECOGNITION, 11-5
 - RECORD, 11-6

- definition, 11-1
 - procedures, 11-1
 - running, 11-2
 - RPG option, in PATCH, 9-8
 - RTOR intrinsic, 8-5
- S**
- SAVEFACTOR modifier, in FILEDATA, 5-53
 - SECURITY modifier, in FILEDATA, 5-53
 - <seq number>
 - LIST command, in INTERACTIVEXREF, 6-21
 - <sequence number column>, in COMPARE utility, 2-2
 - <sequence number>
 - EXPAND command, in INTERACTIVEXREF, 6-17
 - procedure specification, in INTERACTIVEXREF, 6-4
 - range specification, in INTERACTIVEXREF, 6-7
 - <sequence range>, LIST command, in INTERACTIVEXREF, 6-21
 - <serial number list>
 - destination, in FILECOPY, 4-9
 - modifier, in FILECOPY, 4-9
 - <serial number>
 - basic constructs, in DUMPALL, 3-73
 - file attribute assignment, in DUMPALL, 3-71
 - SET command, in INTERACTIVEXREF, 6-32
 - SIN mathematical function, 8-6
 - SINGLE option, in PATCH, 9-8, 9-27
 - single-precision functions, 8-1
 - exponentiation, 8-5
 - SINH mathematical function, 8-6
 - SKIP command, in DUMPALL, 3-63
 - <skip specification>
 - basic constructs, in DUMPALL, 3-82
 - CAT command, in DUMPALL, 3-14
 - COPY command, in DUMPALL, 3-21
 - DMPMT command, in DUMPALL, 3-39
 - HEXDSK command, in DUMPALL, 3-43
 - LIST command, in DUMPALL, 3-46, 3-57
 - TEST command, in DUMPALL, 3-52
 - SL (Support Library) command
 - in ISAM, 10-1
 - in KEYEDIO, 7-3
 - SORT utility
 - characteristics of sort modes, 12-7
 - collating sequences, 12-51
 - determining sort mode, 12-7
 - explanation of, 12-1
 - input file, 12-2
 - input procedure, 12-2
 - output file, 12-3
 - output procedure, 12-3
 - parameters
 - compare procedure, 12-3
 - disk size, 12-6
 - input, 12-2
 - memory size, 12-4
 - number of tapes, 12-4
 - output, 12-3
 - record size, 12-4
 - <source file name> modifier, in FILECOPY, 4-9
 - <source file name> modifier, in FILEDATA, 4-11
 - <source from group> file modifier, in FILECOPY, 4-9
 - <source from group> modifier, in FILEDATA, 4-10
 - <source>
 - file specification, in FILECOPY, 4-8
 - modifier, in FILECOPY, 4-9
 - SQRT mathematical function, 8-7
 - SQUASH option, in PATCH, 9-8
 - STOP command, in INTERACTIVEXREF, 6-33
 - <string> modifier, in FILECOPY, 4-9
 - structural file attributes, in DUMPALL, 3-5
 - STRUCTUREMAP request, in FILEDATA, 5-41
 - <subrange specification>, in INTERACTIVEXREF, 6-7
 - SUMMARY
 - command, in INTERACTIVEXREF, 6-33
 - modifier in FILEDATA, 5-22
 - SYMBOL command, in INTERACTIVEXREF, 6-34
- T**
- TAN mathematical function, 8-7
 - TANH mathematical function, 8-7
 - tape labels
 - installation-defined
 - restrictions, 11-1
 - Tape labels

Index

- installation-defined
 - LABEL description format, 11-2
 - RLTABLEGEN commands, 11-3
- TAPE modifier, 5-32
- <tape name>
 - basic constructs, in FILEDATA, 5-3
 - FILENAMES request, in FILEDATA, 5-32
 - NOREPORTS request, in FILEDATA, 5-39
- <tape serial number>
 - ARCHIVEINFO request, in FILEDATA, 5-11
 - ATTRIBUTES request, in FILEDATA, 5-14
 - BACKUP request, in FILEDATA, 5-17
 - basic constructs, in FILEDATA, 5-3
 - CATALOGINFO request, in FILEDATA, 5-20
 - FILENAMES request, in FILEDATA, 5-32
- TAPEDIR request, in FILEDATA, 5-43
- tapes
 - format of, in DUMPALL
 - labeled, 3-85
 - unlabeled, 3-84
 - labels, restrictions on installation-defined, 11-1
 - library maintenance, in DUMPALL
 - copying, 3-45
 - listing, 3-45
 - nonstandard names, copying, 3-25
 - unloading, suppressing in DUMPALL, 3-19, 3-26
- <task request>, in FILEDATA, 5-10
- TDIR command, in FILEDATA, 5-59
- TERMINAL command, in
 - INTERACTIVEXREF, 6-35
- TEST command, in DUMPALL, 3-12, 3-52
- TIMESTAMP modifier, in FILEDATA, 5-54
- <timestamp>
 - basic constructs, in FILECOPY, 4-4
 - CREATED/ACCESSED/UPDATED request, 4-5
 - EXPIRED request, in FILECOPY, 4-7
- TITLE modifier, in FILEDATA, 5-54
- <title>, \$.INSERT option, in PATCH, 9-19
- <total option>, in value option, in PATCH, 9-9
- TOTALSECTORS modifier, in FILEDATA, 5-54
- <train ID>

- DMPMT command, in DUMPALL, 3-39
- LIST command, in DUMPALL, 3-46, 3-57
- TEST command, in DUMPALL, 3-52
- translation, data or character set, in DUMPALL, 3-9

U

- <underscore>, basic constructs, in DUMPALL, 3-73
- <unit number>
 - basic constructs, in FILEDATA, 5-3
 - FILENAMES request, in FILEDATA, 5-32
 - NOREPORTS request, in FILEDATA, 5-39
 - TAPEDIR request, in FILEDATA, 5-43
 - TDIR command, in FILEDATA, 5-59
- UNITS modifier, in FILEDATA, 5-54
- unloading tapes, suppression of, 3-19, 3-26
- <usercode>
 - basic constructs, in DUMPALL, 3-73
 - basic constructs, in FILEDATA, 5-3
 - file name, in FILECOPY, 4-3
- USERINFO modifier, in FILEDATA, 5-54

V

- <value option>, patch control record, in PATCH, 9-8
- variable length records, 3-7
- VERSION modifier, in FILEDATA, 5-54
- <version number>
 - \$.FLAG, in PATCH, 9-18
 - \$.VERSION, in PATCH, 9-28
- <version option>, in value option, in PATCH, 9-9
- <volume name>
 - basic constructs, in FILEDATA, 5-3
 - TAPEDIR request, in FILEDATA, 5-43
 - TDIR command, in FILEDATA, 5-59

W

- WARNINGS modifier, in FILEDATA, 5-54
- <WFL statement>, patch WFL record, in PATCH, 9-7

WHAT command, in INTERACTIVEXREF,
6-36

WHATFILES command, in
INTERACTIVEXREF, 6-36

X

XREFANALYZER

explanation of, 13-1

use by INTERACTIVEXREF, 6-1

XREFFILES

producing for INTERACTIVEXREF, 6-1

\$.NDLII option, in PATCH, 9-26

\$.NEW option, in PATCH, 9-26

\$.OUT option, in PATCH, 9-26

\$.PASCAL option, in PATCH, 9-27

\$.PATCHDECK options, in PATCH, 9-18

\$.PDUMP option, in PATCH, 9-32

\$.RPG option, in PATCH, 9-27

\$.SQUASH option, in PATCH, 9-28

\$.TOTAL option, in PATCH, 9-28

\$.VERSION option, in PATCH, 9-28

\$. FLAG option, in PATCH, 9-18

\$. OPTIONS, (See PATCH utility)

\$.BRIEF option, in PATCH, 9-10

\$.BUG option, in PATCH, 9-30

\$.CANDE option, in PATCH, 9-30

\$.COBOL option, in PATCH, 9-11

\$.COMPARE option, in PATCH, 9-11

\$.COMPILE option, in PATCH, 9-12

\$.CONFLICT option, in PATCH, 9-12

\$.COUNT option, in PATCH, 9-13

\$.CYCLE option, in PATCH, 9-13, 9-28

\$.DELETE option, in PATCH, 9-13

\$.DELIMOPT option, in PATCH, 9-14

\$.DISCARD option, in PATCH, 9-31

\$.DISK \$ option, in PATCH, 9-18

\$.DISK option, in PATCH, 9-15

\$.DUMP option, in PATCH, 9-16

\$.END option, in PATCH, 9-31

\$.EOF option, in PATCH, 9-17

\$.EQUATE option, in PATCH, 9-32

\$.ERRLIST option, in PATCH, 9-17

\$.EXECUTE option, in PATCH, 9-17

\$.FILE option, in PATCH, 9-18

\$.GUARD option, in PATCH, 9-19

\$.INSERT option, in PATCH, 9-19

\$.LABEL option, in PATCH, 9-21

\$.LIST option, in PATCH, 9-22

\$.LISTD option, in PATCH, 9-22

\$.LISTI option, in PATCH, 9-22

\$.LISTN option, in PATCH, 9-23

\$.LISTP option, in PATCH, 9-23

\$.MARK option, in PATCH, 9-23

\$.MARKBLANK option, in PATCH, 9-24

\$.MOVE option, in PATCH, 9-25

Help Us To Help You

Publication Title _____

Document Number _____

Date _____

Unisys Corporation is interested in your comments and suggestions regarding this manual. We will use them to improve the quality of your Product Information. Please check type of suggestion:

Addition

Deletion

Revision

Recommended Change (Please identify affected pages) _____

Name _____

Telephone Number _____

Title _____

Company _____

Address (Street, City, State, Zip) _____

Help Us To Help You

Publication Title _____

Document Number _____

Date _____

Unisys Corporation is interested in your comments and suggestions regarding this manual. We will use them to improve the quality of your Product Information. Please check type of suggestion:

Addition

Deletion

Revision

Recommended Change (Please identify affected pages) _____

Name _____

Telephone Number _____

Title _____

Company _____

Address (Street, City, State, Zip) _____

Help Us To Help You

Publication Title _____

Document Number _____

Date _____

Unisys Corporation is interested in your comments and suggestions regarding this manual. We will use them to improve the quality of your Product Information. Please check type of suggestion:

Addition

Deletion

Revision

Recommended Change (Please identify affected pages) _____

Name _____

Telephone Number _____

Title _____

Company _____

Address (Street, City, State, Zip) _____



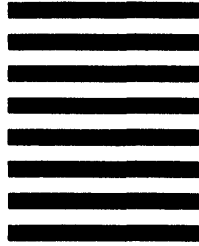
No Postage
necessary
if mailed in the
United States

BUSINESS REPLY MAIL

First Class Permit No. 817 Detroit, MI 48232

Postage Will Be Paid By Addressee

Unisys Corporation
ATTN: Publications
25725 Jeronimo Road
Mission Viejo, CA 92691-9826 USA



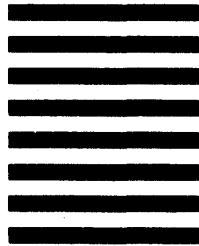
No Postage
necessary
if mailed in the
United States

BUSINESS REPLY MAIL

First Class Permit No. 817 Detroit, MI 48232

Postage Will Be Paid By Addressee

Unisys Corporation
ATTN: Publications
25725 Jeronimo Road
Mission Viejo, CA 92691-9826 USA



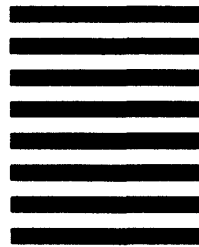
No Postage
necessary
if mailed in the
United States

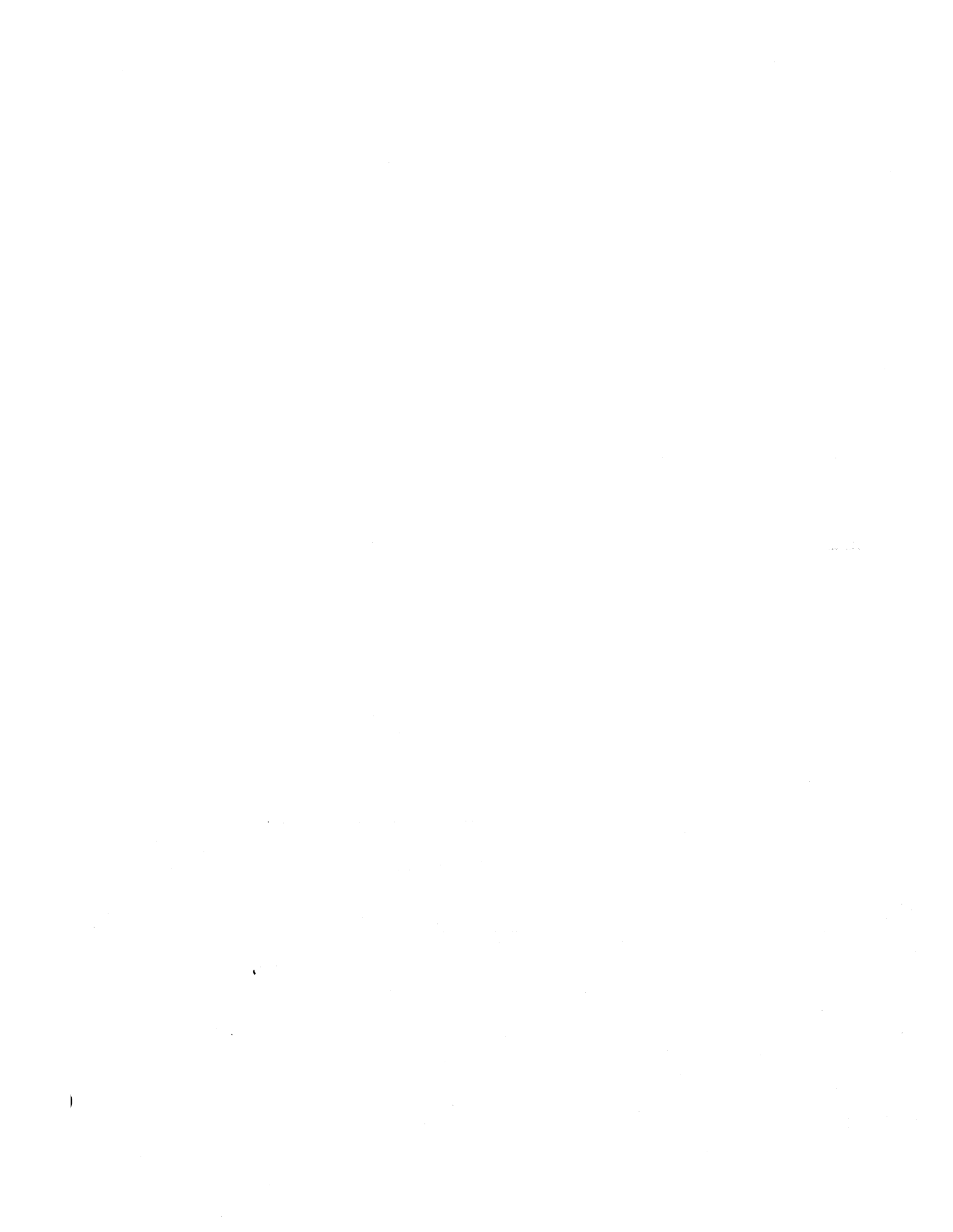
BUSINESS REPLY MAIL

First Class Permit No. 817 Detroit, MI 48232

Postage Will Be Paid By Addressee

Unisys Corporation
ATTN: Publications
25725 Jeronimo Road
Mission Viejo, CA 92691-9826 USA







86000460-100