

INFOS®

System User's Manual

(RDOS)

093-000114-01

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000114
©Data General Corporation, 1975, 1978
All Rights Reserved
Printed in the United States of America
Revision 01, April 1978
Licensed Material - Property of Data General Corporation

NOTICE

Data General Corporation (DGC) has prepared this manual for use by DGC personnel, licensees, and customers. The information contained herein is the property of DGC and shall not be reproduced in whole or in part without DGC prior written approval.

DGC reserves the right to make changes without notice in the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented, including but not limited to typographical, arithmetic, or listing errors.

**INFOS®
System User's Manual
(RDOS)
093-000114**

Revision History:

093-000114

**Original Release - July 1975
First Revision - April 1978**

The RDOS/INFOS System User's Manual (093-000114-01) replaces:

the INFOS System Programmer's Manual (093-000114-00)
the INFOS System Planning Manual (093-000115-01)
the INFOS Language Interface Manual (093-000127-02)

This document has been extensively revised from revision 00; therefore, change indicators have not been used.

The following are trademarks of Data General Corporation, Westboro, Massachusetts:

<u>U.S. Registered Trademarks</u>			<u>Trademarks</u>
CONTOUR I	INFOS	NOVALITE	DASHER
DATAPREP	NOVA	SUPERNOVA	microNOVA
ECLIPSE	NOVADISC		

Preface

Welcome to the *INFOS® System User's Manual*. This manual will tell you everything you need to know to plan and program your application of the INFOS system. Here you'll find how the system works, what features you can use, what types of things the system will do, and how you can fine tune the system for maximum performance efficiency.

Before you get into all that, however, let's take a minute to look at the INFOS system as a whole -- rather than as individual pieces. First, what is the INFOS system?

This is not an easy question to answer simply; after all, that's what this whole manual is about. Think of the system as a forest which you are viewing from an

airplane. From this altitude, all you can see is a bunch of trees. But, in the INFOS forest, these trees represent your processing options. By choosing the appropriate trees, your programs can work their way through the INFOS forest, drawing on its abundant resources to create, access, and maintain simple or complex data files. As you move closer to the forest, you'll notice numerous saplings scattered throughout. These represent parts of our Real-Time Disk Operating System, or RDOS. That is, the INFOS system is a grown-up RDOS. Finally, you'll find that the roots of the RDOS/INEOS forest grow only on a Data General commercial ECLIPSE computer.



Figure 1. *INFOS Forest*

As you near the INFOS system forest, you'll notice many ways to approach it. You can take the paths marked COBOL, RPG II, FORTRAN IV or 5, Business BASIC, or Macroassembler. You will also pass through parts of the INFOS forest if you follow the path of DG's IDEA system.

Finally, you can also get into the forest on the trail marked INFOS Utilities. These allow you to create, sort, delete, copy, rename, and inquire about your files, as well as initialize magnetic tapes, and retrieve your file specifications at runtime.

This manual is your guidebook through the INFOS forest. It will help you choose between the options which the system allows you. To make this manual as useful as possible, we have organized it into a modular-type format.

Licensed Material - Property of Data General Corporation

Part One is the heart of this manual. It explains, in plain English, all the features and functions available in the INFOS system. Furthermore, Chapter 1 of Part One is the most important chapter in this manual because it presents the basic, *general* information you need to know before you begin planning or programming your INFOS application. Read this chapter carefully; it can save you a lot of grief later on.

Following Chapter 1, you can either read all of the remaining chapters in Part One, or choose the chapters which seem most applicable to your needs. For example, if you already know which access methods will best suit your situation, you can go directly to those chapters. Later, if you wish, you can read the chapters you skipped to learn about other INFOS capabilities. If, however, you are unfamiliar with the details of the options available in the INFOS system, you should read

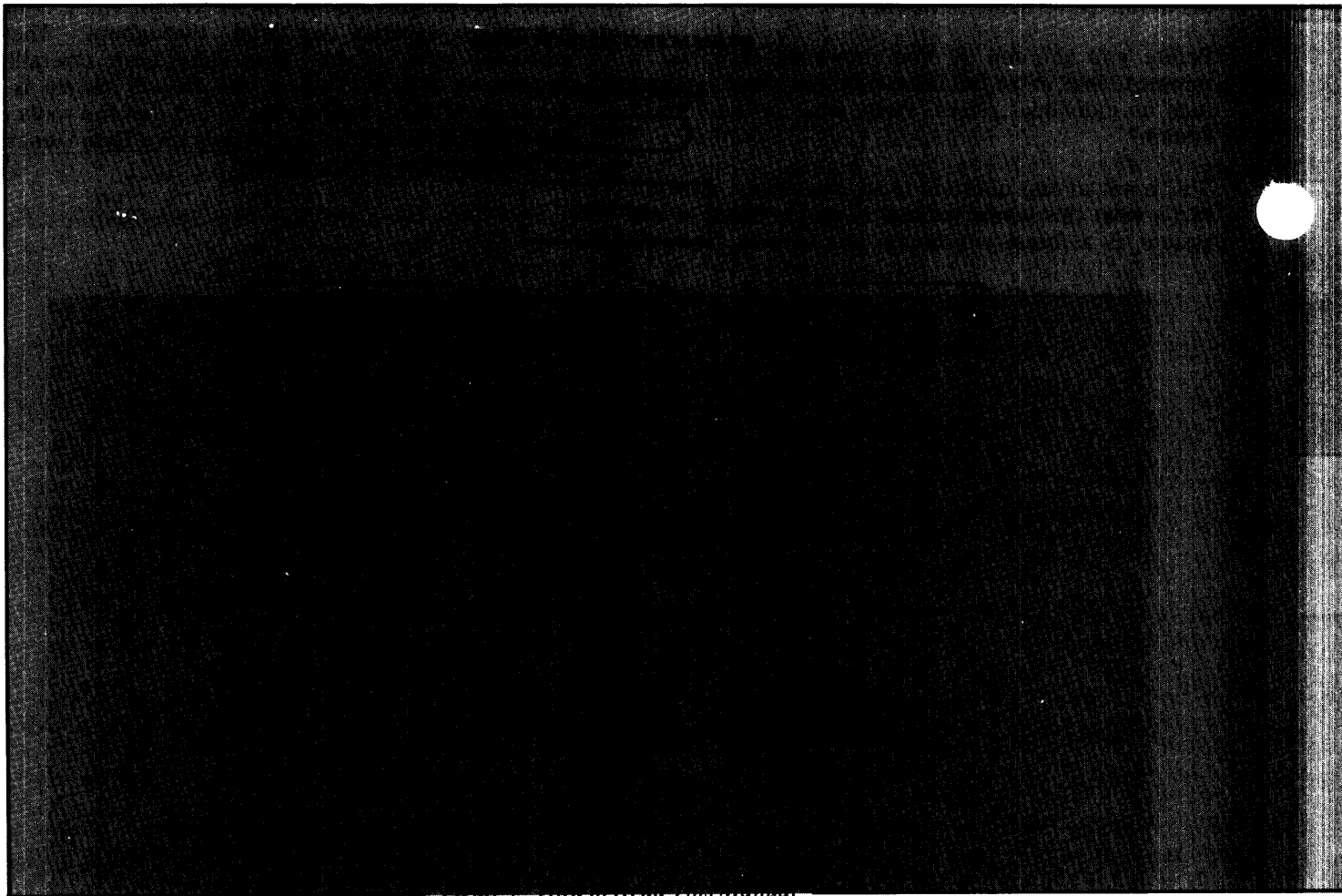


Figure 2: How to Read This Manual

Licensed Material - Property of Data General Corporation

Chapters 2 through 5 to help you decide on the best method for solving your problems. And, no matter which access method you choose, you should read Chapter 6 in Part One. It describes the things you need to know about the RDOS interface with the INFOS system, as well as miscellaneous planning considerations.

So, whether you're a Programmer, a Systems Analyst, or just curious, you should read the appropriate chapters in Part One to learn about the nature of the flora and fauna in the INFOS forest. Then, if it's applicable, you can read the corresponding chapters in Part Two to find out how to make those features and functions available to your programs. (Remember, however, to read Chapter 1 in Part *Two* to learn about general INFOS system programming considerations and Chapter 6 if you need to know about the contents of packets and how to use our Macroassembler.)

Part Three contains the appendixes for this manual. Appendixes A and B contain detailed information about labeled magnetic tapes and things you should consider when you're using the Data Base Access Method -- in other words, further details for your planning considerations. Appendixes C, D, and E describe programming details: the interface between the INFOS system and FORTRAN, explanations of the system's error messages, and a chart of characteristics of the various peripheral devices. We'll refer you to these back-of-the-book sections for additional details, but do not mistake them for "excess" information.

There are many beautiful sights within the INFOS forest, but every trail has its pitfalls. Read this guidebook carefully to find out where they are and how to avoid them.

End of Preface

Contents

Part One: Planning Your RDOS/INFOS System

Chapter 1 - General Information

Access Methods and File Formats	I-1-1
Sequential Access Method	I-1-1
Random Access Method	I-1-1
Indexed Sequential Access Method	I-1-1
Data Base Access Method	I-1-2
Concurrent Tasking	I-1-2
General Considerations	I-1-2
Processing Modes	I-1-2
Create Update Mode	I-1-2
Update Mode	I-1-2
Input Mode	I-1-2
Output Mode	I-1-3
Record Formats	I-1-3
Fixed Length Record Format	I-1-3
Variable Length Record Format	I-1-3
Undefined Length Record Format	I-1-4
Data Sensitive Record Format	I-1-4
Data Transfers	I-1-4
Record Packing	I-1-5
Transferring Fixed Length Records	I-1-5
Transferring Variable Length Records	I-1-5
Transferring Undefined Length Records	I-1-5
Transferring Data Sensitive Records	I-1-5
Buffer Management	I-1-5
Footnotes to Data Transfers	I-1-6
A Word About Volumes	I-1-6
Multivolume Files	I-1-6
How to Process INFOS Files	I-1-7
How to Create a New File	I-1-7
Line Printers and Terminals as INFOS Files	I-1-7
File Naming, Briefly	I-1-7
How to Process an Existing Tape File	I-1-7
How to Process an Existing Disk File	I-1-7
Summary	I-1-7

Chapter 2 - Sequential Access Method (SAM) Files

SAM Disk Files	I-2-1
How to Create SAM Disk Files	I-2-2
How to Open an Existing SAM Disk File	I-2-4
SAM Labeled Tape Files	I-2-4
Initialization	I-2-4
How to Create SAM Labeled Tape Files	I-2-4
How to Open an Existing SAM Labeled Tape File	I-2-7
How to Use Peripheral Devices as SAM Files	I-2-7
What to Do with Unlabeled SAM Tape Files	I-2-7
SAM Summary	I-2-7
How to Process SAM Files	I-2-12
Reading an Existing File	I-2-12
Writing a New SAM File	I-2-12
Appending Records	I-2-12
Rewriting Existing Records	I-2-12
Overwriting Existing Records	I-2-12
Additional Features (Point, SETX, and RELX)	I-2-12
Summary	I-2-13

Chapter 3 - Random Access Method (RAM) Files

How to Create a RAM File	I-3-1
How to Open an Existing RAM File	I-3-3
Processing Your RAM File	I-3-3
Read	I-3-3
Write	I-3-4
Pre-read	I-3-4
Write Immediate	I-3-5
Read Inhibit	I-3-5
Lock and SETX	I-3-5
Hold	I-3-6
FEOV	I-3-6

Chapter 4 - Indexed Sequential Access Method (ISAM) Files

General Concepts	I-4-1
The Database	I-4-2
The Index File	I-4-2
Concurrent Access	I-4-2
Space Management	I-4-3
How to Create an ISAM File	I-4-3
Creating the Index File	I-4-3
Defining Volumes of Your Index File	I-4-5
Defining Your Database File	I-4-6
Defining Volumes of Your Database File	I-4-6
How to Open an Existing ISAM File	I-4-7
Opening Your Index File	I-4-7
Opening Your Existing Database File	I-4-7
Processing Your ISAM File (Background)	I-4-8
Processing Your ISAM File (Operations)	I-4-9

Chapter 4 - Indexed Sequential Access Method (ISAM) Files (continued)

Processing Functions	I-4-9
Read	I-4-9
Write	I-4-9
Rewrite	I-4-9
Delete	I-4-9
Reinstate	I-4-10
Delete Subindex	I-4-10
Utility Functions	I-4-10
Retrieve Status	I-4-10
Retrieve Key	I-4-10
Retrieve High Key	I-4-10
Auxiliary Features	I-4-10
Lock/Unlock	I-4-10
Suppress Database	I-4-10

Chapter 5 - Data Base Access Method (DBAM) Files

General Concepts	I-5-1
Subindexing	I-5-1
Partial Records	I-5-3
Multiple Indexes and File Inversions	I-5-4
Linked Subindexes	I-5-4
Automatic Key Compression	I-5-5
Optimized Distribution	I-5-5
Temporary Indexes	I-5-5
How to Create an INFOS DBAM File	I-5-6
Defining Your Index File	I-5-6
Defining Each Volume of an Index File	I-5-8
Defining Your Database File	I-5-9
Defining Each Volume of Your Database File	I-5-9
How to Open an Existing DBAM File	I-5-10
Processing Your DBAM File	I-5-11
DBAM Access Methods - Keyed, Relative, and Combined	I-5-11
Combining Relative and Keyed Access	I-5-13
DBAM Processing Operations	I-5-15
DBAM Processing Functions	I-5-15
Read	I-5-15
Write	I-5-15
Rewrite	I-5-16
Delete	I-5-16
Reinstate	I-5-16
Define Subindex	I-5-16
Link Subindex	I-5-16
Delete Subindex	I-5-16
DBAM Utility Functions	I-5-16
Retrieve Status	I-5-16
Retrieve Key	I-5-17
Retrieve High Key	I-5-17
Retrieve Subindex Definition	I-5-17
DBAM Auxiliary Features	I-5-17
Lock/Unlock and Suppress Database	I-5-17
Nonspecific Search Keys	I-5-17
Suppress Partial Record	I-5-17

Chapter 6 - The RDOS/INFOS Interface

Considerations When You're Generating an INFOS System	I-6-1
Memory Space	I-6-2
The System Area	I-6-2
Resident INFOS/RDOS Code	I-6-2
System Buffers	I-6-2
Windows	I-6-3
INFOS System File Control Area	I-6-3
The User Area	I-6-5
Summary	I-6-5
I/O Buffer Space	I-6-6
I/O Buffer Management	I-6-8
File Naming	I-6-10
Disk Space Allocation	I-6-10
Using Peripheral Devices as INFOS Files	I-6-11
How to Deal with Unlabeled Magnetic Tapes	I-6-11

Part Two: Programming Your RDOS/INFOS System

Chapter 1 - General Information

Packets	II-1-1
Packet Types	II-1-1
File Definition Packets	II-1-1
Volume Definition Packets	II-1-2
General Processing Packets	II-1-2
Extended Processing Packets	II-1-2
Key Definition Packets	II-1-2
Subindex Definition Packets	II-1-2
Point Processing Packets	II-1-2
Link Subindex Processing Packets	II-1-2
Volume Initialization Packets	II-1-2
Tables	II-1-2
How to Open an INFOS File	II-1-3
System Calls	II-1-3
The Permanent File Specification	II-1-4

Chapter 2 - Sequential Access Method (SAM) Files

How to Open SAM Files	II-2-1
Processing SAM Files	II-2-4
Read Processing Request	II-2-5
Write Processing Request	II-2-5
Rewrite Processing Request	II-2-5
Close and Force End of Volume Requests	II-2-6
Magnetic Tape Control Request	II-2-7
Point Request	II-2-7

Chapter 3 - Random Access Method (RAM) Files

How to Open a RAM File	II-3-1
Steps in Opening a RAM File	II-3-1
Processing Your RAM File	II-3-4
Read Request	II-3-4
Write Request	II-3-4
Close Request	II-3-5
Force End-of-Volume Request	II-3-5
Set Exclusive Use Request	II-3-5
Preread Request	II-3-5

Chapter 4 - Indexed Sequential Access Method (ISAM) Files

How to Open ISAM Files	II-4-1
Steps for Opening in the Create Update Mode	II-4-1
Steps for Opening in the Update Mode without a Database Runtime FDP	II-4-5
Steps for Opening in the Update Mode with a Database Runtime FDP	II-4-7
Processing ISAM Files	II-4-9
Keyed Access	II-4-9
Relative Access	II-4-9
Read Processing Request	II-4-10
Write Processing Request	II-4-11
Rewrite Processing Request	II-4-11
Delete Processing Request	II-4-12
Delete Subindex Processing Request	II-4-12
Reinstate Processing Request	II-4-12
Retrieve Status Processing Request	II-4-13
Retrieve Key and Retrieve High Key Processing Requests	II-4-13

Chapter 5 - Data Base Access Method (DBAM) Files

Opening DBAM Files	II-5-1
Steps for Opening in the Create Update Mode	II-5-1
Steps for Opening in the Update Mode without a Database Runtime FDP	II-5-5
Steps for Opening in the Update Mode with a Database Runtime FDP	II-5-7
Steps for Creating a New DBAM Index without a Database Runtime FDP	II-5-9
Steps for Creating a New DBAM Index with a Database Runtime FDP	II-5-12
Processing DBAM Files	II-5-15
Keyed Access	II-5-15
Relative Access	II-5-15
Combined Keyed and Relative Access	II-5-15
Read Processing Request	II-5-16
Write Processing Request	II-5-17
Rewrite Processing Request	II-5-18
Define Subindex Processing Request	II-5-19
Link Subindex Processing Request	II-5-19
Delete Processing Request	II-5-20
Delete Subindex Processing Request	II-5-21
Reinstate Processing Request	II-5-21
Retrieve Key and Retrieve High Key Processing Requests	II-5-21
Retrieve Subindex Definition Processing Request	II-5-22
Retrieve Status Processing Request	II-5-22

Chapter 6 - Packet Formats

General Packet Information and Conventions	II-6-1
File Definition Packet (FDP)	II-6-1
INFOS FDP Parameters by Access Method/Device/and Processing Mode	II-6-9
Volume Definition Packets (VDP)	II-6-12
INFOS VDP Parameters by Access Method/Device/and Processing Mode.	II-6-15
Volume Tables	II-6-17
General Processing Packet (SAM and RAM files only)	II-6-18
Extended Processing Packet (ISAM and DBAM files only)	II-6-20
Key Definition Packet	II-6-24
Key Tables	II-6-25
Subindex Definition Packet	II-6-26
Point Processing Packet	II-6-27
Link Subindex Processing Packet	II-6-28
Volume Initialization Packet	II-6-31
Magnetic Tape Control Processing Packet	II-6-33

Chapter 7 - How to Use the Macroassembler with the INFOS System

Keyword Parameters	II-7-1
General	II-7-1
Access Method	II-7-2
Formatting Records	II-7-2
I/O Mode	II-7-2
Label Types	II-7-2
Translation Specifiers	II-7-2
Macro Functions	II-7-2
BLDFDP	II-7-2
BLDVDP	II-7-6
BLDPP	II-7-8
BLDPNT	II-7-10
BLDMTC	II-7-10
BLDKDP	II-7-11
BLDSDP	II-7-12
BLDLSP	II-7-13
BLDVIP	II-7-14
BLDULT	II-7-15
The Assembly Language Interface	II-7-15

Part Three: Appendixes

Appendix A - Labeled Magnetic Tapes

General Concepts	III-A-1
Label Types and Levels	III-A-1
User Labels	III-A-6
Volume Initialization and Release	III-A-6
How to Initialize and Release Tapes Through System Calls	III-A-6
Runtime Initialization and Release (General)	III-A-7
Runtime Initialization	III-A-7
Runtime Release	III-A-8
Processing Labeled Magnetic Tapes	III-A-9
Positioning When Writing	III-A-9
Graphic Arts Section	III-A-10

Appendix B - Subindex and Database File Properties

Introduction	III-B-1
Subindexes	III-B-1
Selector Subindexes	III-B-7
Database Files	III-B-8

Appendix C - The INFOS/FORTRAN Interface

General Information	III-C-1
Packet Building Routines	III-C-1
Table Building Routine	III-C-1
Pure Processing Routines	III-C-2
Building/Processing Routines	III-C-2
Arguments	III-C-2
Using DEFAULT (or DEF)	III-C-2
Using NIL	III-C-2
Error Conditions	III-C-3
Using FINFOS.ER	III-C-3
Loading Your Program and the Interface Routines	III-C-3
Call Formats	III-C-3
Packet Building Routines	III-C-4
INFFDP	III-C-4
INFIFDP	III-C-5
INFVDP	III-C-6
INFVIP	III-C-6
INFKEY	III-C-7
INFLSP	III-C-8
Table Building Routine	III-C-8
INFULT	III-C-8
Pure Processing Routines	III-C-9
INFPREP	III-C-9
INFOPEN	III-C-9
INFINIT	III-C-10
Building/Processing Routines	III-C-10
INFOS	III-C-10
INFOX	III-C-11

Appendix D - INFOS System Error Messages

Note to All FORTRAN Programmers	III-D-1
---	---------

Appendix E - Device Characteristics

Illustrations

Figure Caption

1	INFOS Forest	iii
2	How to Read this Manual	iv
I-1-1	Disk File Data Transfer	I-1-4
I-1-2	Transferring Fixed Length Records	I-1-5
I-3-1	RAM Read and Write Sequence	I-3-4
I-3-2	Sequence of Events for a Pre-read Request	I-3-5
I-3-3	Sequence of Events for a Read Inhibit Request	I-3-5
I-4-1	INFOS ISAM File	I-4-1
I-4-2	ISAM Index File	I-4-8
I-5-1	DBAM Two-Level Index	I-5-2
I-5-2	Three-Level DBAM Index	I-5-3
I-5-3	Single Database With Two Indexes.	I-5-4
I-5-4	A Linked Subindex	I-5-4
I-5-5	Single and Multilevel Keys in DBAM	I-5-11
I-5-6	Relative Movement within a DBAM File	I-5-12
I-5-7	Segment of a Multilevel Index Structure	I-5-13
I-6-1	Components of the System Area	I-6-2
I-6-2	Partitioned User Area	I-6-5
I-6-3	User Area Partitioned for a 65K Byte Foreground.	I-6-5
I-6-4	Buffer Space Requirements Per File Opening	I-6-7
I-6-5	Least-Recently-Used Technique for Input	I-6-8
I-6-6	DBAM Index with Sixteen HPNs	I-6-9
II-3-1	FDP and Volume Table Relationship	II-3-1
II-4-1	Open ISAM File in Create Update Mode	II-4-2
II-4-2	Open ISAM File in Update Mode without Database Runtime FDP	II-4-5
II-4-3	Open ISAM File in Update Mode with Database Runtime FDP	II-4-7
II-4-4	Key Table	II-4-9
II-5-1	Open DBAM File in the Create Update Mode	II-5-2
II-5-2	Open ISAM File in Update Mode without Database Runtime FDP	II-5-5
II-5-3	Open DBAM File in Update Mode with Database Runtime FDP.	II-5-7
II-5-4	Create a New DBAM Index without a Database Runtime FDP.	II-5-9
II-5-5	Create a New DBAM Index with a Database Runtime FDP.	II-5-12
II-5-6	DBAM Index Structure	II-5-15
II-6-1	File Definition Packet	II-6-2
II-6-2	Volume Definition Packet	II-6-12
II-6-3	General Processing Packet	II-6-18
II-6-4	Extended Processing Packet.	II-6-20
II-6-5	Key Definition Packet	II-6-24
II-6-6	Sample Key Table	II-6-25
II-6-7	Subindex Definition Packet	II-6-26
II-6-8	Point Processing Packet	II-6-27
II-6-9	Link Subindex Processing Packet	II-6-28
II-6-10	Volume Initialization Packet	II-6-31
II-6-11	Magnetic Tape Control Processing Packet	II-6-33
II-7-1	Index File Definition Packet	II-7-4
II-7-2	Database File Definition Packet	II-7-5
II-7-3	Index File Volume Definition Packet	II-7-7
II-7-4	Database Volume Definition Packet	II-7-7
II-7-5	Extended Processing Packet	II-7-9
II-7-6	Point Processing Packet	II-7-10

Figure Caption

II-7-7	Key Definition Packet	II-7-11
II-7-8	Subindex Definition Packet	II-7-12
II-7-9	Link Subindex Processing Packet	II-7-13
II-7-10	Volume Initialization Packet	II-7-14
A-1	Sequence of Events for Runtime Initialization and Release	III-A-8
A-2	Level 1 ANSI Labels Supported by the INFOS System	III-A-10
A-3	Level 2 ANSI Labels Supported by the INFOS System	III-A-11
A-4	Level 3 ANSI Labels Supported by the INFOS System	III-A-12
A-5	Level 1 IBM Labels Supported by the INFOS System	III-A-13
A-6	Level 2 IBM Labels Supported by the INFOS System	III-A-14
B-1	Subindex with Four Tree Levels	III-B-1
B-2	Four-Level Subindex with Nodes	III-B-4
B-3	Three-Level Subindex with Nodes	III-B-5
B-4	A Database Accessed by Two Unique Indexes	III-B-7
B-5	Single Index Structure with a Selector Subindex	III-B-7

Tables

Table Title

I-1-1	Processing Modes	I-1-3
I-1-2	Record Formats and Access Methods	I-1-4
I-2-1	Steps in Creating a SAM Disk File	I-2-3
I-2-2	Steps in Creating a SAM Labeled Tape File	I-2-6
I-2-3	File Definition Options When You Open a New SAM File	I-2-8
I-2-4	File Definition Options When You Open a New SAM File	I-2-9
I-2-5	Volume Definition Options When You Open a New SAM File	I-2-10
I-2-6	Volume Definition Options When You Open an Existing SAM File	I-2-11
I-2-7	SAM File Processing Summary	I-2-13
I-3-1	Steps in Creating a RAM File	I-3-2
I-3-2	Steps in Opening an Existing RAM File	I-3-3
I-4-1	Steps in Defining Your ISAM Index File	I-4-4
I-4-2	Steps in Defining Each Volume of Your ISAM Index File	I-4-5
I-4-3	Steps in Defining Your ISAM Database File	I-4-6
I-4-4	Steps in Defining Each Volume of an ISAM Database File	I-4-6
I-4-5	Steps in Opening an Existing ISAM Index File	I-4-7
I-5-1	Steps in Creating a DBAM Index File	I-5-7
I-5-2	Steps in Defining Each Volume of a DBAM Index File	I-5-8
I-5-3	Steps in Defining a DBAM Database File	I-5-9
I-5-4	Steps in Defining Each Volume of a DBAM Database File	I-5-9
I-5-5	Steps in Opening an Existing DBAM Index File	I-5-10
I-5-6	Steps in Opening an Existing DBAM Database File	I-5-10
I-5-7	DBAM Processing Operation	I-5-15
II-1-1	FDP Parameters that Become Unchangeable Entries in the File's PFS	II-1-5
II-1-2	FDP Parameters that Become Runtime Entries in the File's PFS	II-1-5
II-2-1	SAM Processing Functions	II-2-4
II-4-1	ISAM Index FDP for Create Update Mode	II-4-3
II-4-2	ISAM Database FDP for Create Update Mode	II-4-4
II-4-3	VDPs for ISAM Index and Database Files Opened in the Create Update Mode	II-4-4
II-4-4	ISAM Index FDP for Update Mode without Database Runtime FDP	II-4-6

Table	Title	
II-4-5	VDP for ISAM Index Opened in the Update Mode	II-4-6
II-4-6	ISAM Index FDP for Update Mode with Database Runtime FDP	II-4-8
II-4-7	ISAM Database Runtime FDP for Update Mode	II-4-8
II-4-8	VDPs for ISAM Index and Database Opened in the Update Mode	II-4-8
II-5-1	DBAM Index FDP for Create Update Mode	II-5-3
II-5-2	DBAM Database FDP for Create Update Mode	II-5-4
II-5-3	VDPs for Index and Database Files Opened in the Create Update Mode	II-5-4
II-5-4	DBAM Index FDP for Update Mode without Database Runtime FDP	II-5-6
II-5-5	VDP for DBAM Index Opened in the Update Mode	II-5-6
II-5-6	DBAM Index FDP for Update Mode with Database Runtime FDP	II-5-8
II-5-7	DBAM Database Runtime FDP for Update Mode	II-5-8
II-5-8	VDPs for DBAM Index and Database Opened in the Update Mode	II-5-8
II-5-9	FDP for New DBAM Index without Database Runtime FDP	II-5-10
II-5-10	VDP for New DBAM Index	II-5-11
II-5-11	FDP for New DBAM Index with Database Runtime FDP	II-5-13
II-5-12	VDP for New DBAM Index	II-5-14
II-5-13	DBAM Database Runtime FDP for New Index	II-5-14
II-5-14	VDP for DBAM Database Opened in the Update Mode	II-5-14
II-6-1	File Definition Packet (FDP)	II-6-3
II-6-2	Volume Definition Packet (VDP)	II-6-13
II-6-3	General Processing Packet	II-6-18
II-6-4	Extended Processing Packet	II-6-21
II-6-5	Key Definition Packet	II-6-24
II-6-7	Subindex Definition Packet	II-6-26
II-6-8	Point Processing Packet	II-6-27
II-6-9	Link Subindex Processing Packet	II-6-29
II-6-10	Volume Initialization Packet	II-6-31
II-6-11	Magnetic Tape Control Processing Packet	II-6-34
A-1	Level 1 ANSI Labels	III-A-2
A-2	Level 2 ANSI Labels	III-A-2
A-3	Level 3 ANSI Labels	III-A-3
A-4	Level 1 IBM Labels	III-A-4
A-5	Level 2 IBM Labels	III-A-5
A-6	ANSI Standard Volume Label Format	III-A-15
A-7	ANSI Standard User Volume Labels	III-A-15
A-8	ANSI Standard HDR 1, EOVS1, EOF 1 Labels	III-A-16
A-9	ANSI Standard HDR 2, EOVS2, EOF 2 Labels	III-A-17
A-10	IBM Standard Volume Label Format	III-A-17
A-11	IBM Standard HDR 1, EOVS1, EOF 1 Labels	III-A-18
A-12	IBM Standard HDR 2, EOVS2, EOF 2 Labels	III-A-19

Part One: Planning Your RDOS/INFOS System

General Information

Sequential Access Method (SAM) Files

Random Access Method (RAM) Files

**Indexed Sequential Access Method
(ISAM) Files**

Data Base Access Method (DBAM) Files

The RDOS/INFOS Interface

Chapter 1

General Information

Access Methods and File Formats

The first thing you have to do in designing your INFOS® application is choose an access method. Why? Because in the INFOS system the file access methods implicitly define the file formats. Obviously you want an access method that will allow you to process your data as efficiently as possible, so the INFOS system offers you four options:

- Sequential Access Method
- Random Access Method
- Indexed Sequential Access Method
- Data Base Access Method

The following paragraphs briefly describe each of these methods. Chapters two, three, four, and five give you more detailed information about how to apply each method to your situation.

Sequential Access Method

The Sequential Access Method (SAM) is useful when you want to retrieve data in the same sequence as you recorded it. If most of the information you enter into the computer is recorded on sequential access devices (e.g., magnetic tape), use SAM to create and process your files. Also if you will usually output the data generated in processing your file to a sequential device such as a line printer, a magnetic tape, or an interactive terminal, you should choose SAM. You can output to all of these devices with the INFOS system, or you can store your SAM file on a disk.

Random Access Method

The Random Access Method (RAM) supports a different set of applications than SAM. With RAM, you can directly access any record in your file without having to read any of the other records. This is known as random (or direct) access. The sequence in which the INFOS system stores records in a RAM file has no

bearing on the sequence in which you retrieve them when you want to process them. RAM can do this because the records are written and read according to logical record numbers which you supply.

For example, you could arrange an inventory file by part numbers, which would become the record numbers. If you subsequently enter a new part record and want to retrieve it later, it won't matter where that record is stored in the file; you would simply code the equivalent of "Read part (record) number 7792." With RAM, you don't have to rewrite an entire file to keep the records in order; you just enter your data as you have it and retrieve it in order any time you want.

RAM files must reside on disk, since only disks are capable of performing random access.

Indexed Sequential Access Method

The Indexed Sequential Access Method (ISAM) gives you a simple and efficient way to access your data both sequentially and randomly. For example, suppose you want to update customer records in a receivables file randomly during a billing period, and process the file sequentially at the end of that period to prepare the bills for mailing. To do this, you would store the records sequentially according to customer number. Then you would update the records as necessary (randomly) during the billing period, and access them in order of customer number (sequentially) to prepare your bills. ISAM lets you do this through a feature called *keyed access*.

When you set up an ISAM file, you associate a key (word or number) with each record. You then use these keys during retrieval to either locate records randomly, or determine the order in which you want to process them sequentially. In the example above, the customer number is the key.

Your keys can be made up of either numbers or letters, and can vary in length to give you more efficiency in storing your data. The INFOS system also uses an automatic technique to maintain the key/data association. Therefore, you can gain very fast access to your data with no restrictions on the growth or shrinkage of your file. And you can even associate different data records with the same key value. For instance, if you wanted to access the records of customers within a given geographical area, ISAM could give you a listing by zip codes.

Like RAM files, ISAM files must reside on disk.

Data Base Access Method

Sometimes you'll find that ISAM is inadequate for sophisticated applications. That's why the Data Base Access Method (DBAM) exists. It gives you all the features of ISAM, but extends them in three useful ways:

- First, you may tie your data records to more than one key.
- Second, you can index sets of data records according to different sets of keys.
- Third, you can get to your data records by means of a "compound" key.

For example, suppose you assign unique customer numbers on a nationwide basis, but you service the customers through regional offices. At some point, you may want to process the accounts serviced by each regional office individually as well as consider all the accounts in each region. To do this, you would simply create an additional key/data-record association and use the key: "region name, customer number." Then you could process the records of that region sequentially by customer number, or you could randomly locate a specific account number. Furthermore, if an account is serviced by more than one region, you could access its records through more than one regional index. In other words, each region's records would be organized like a little ISAM file, except that they would all be part of the same database.

Remember that if you use the Data Base Access Method, you only need to store each data record *once*. Thus you can avoid both storage use overhead and the unnecessary pain of maintaining duplicate copies of identical data records. DBAM's flexibility lets you access a record by customer name, account number, location, or any other key you want.

Naturally, to do all this, DBAM files have to reside on disk.

Concurrent Tasking

The INFOS system gives you both foreground and background processing capabilities. In addition, it provides multitasking capabilities in both grounds. You can use the same file in both grounds at the same time, and you can open that file more than once in either ground -- or in both. In other words, you can run tasks concurrently within a ground and between grounds. The INFOS system automatically interleaves concurrent inquiries and processes them very efficiently. So you can access a RAM, ISAM, or DBAM file through several terminals simultaneously. Also you can have the system perform several tasks independently, yet concurrently, on the same file, using either the foreground or background, or both.

General Considerations

The following INFOS concepts and features will apply regardless of which access method(s) you choose. Keep them in mind as you plan your system.

Processing Modes

Now that you've begun to think about how you want to organize and access your file, let's look at the ways you can process the data in that file. Each time you want to manipulate the data in your file, you must choose one of the INFOS system's four processing modes. Your choice will depend on three factors:

- the direction in which you want to transfer data
- whether or not the file exists
- the type of device on which the file resides

Create Update Mode

You can use this mode with any access method, but the file must not exist. So you will use Create Update when you want to create new files. However, you may read from your file, as well as write to it, in this mode.

If you select Create Update for a SAM file, the file must reside on disk.

Update Mode

You can select the Update processing mode for any existing file that resides on disk. This mode allows you to both read from and write to your file.

Input Mode

You can only choose the Input Mode if you are processing an existing SAM file that resides on disk, tape, or an interactive terminal, or if you are processing an existing RAM file. In the Input Mode, you can only read data from the file - you cannot write data to it.

Licensed Material - Property of Data General Corporation

Output Mode

You use the Output Mode to create SAM and RAM files, so you can only use it for a SAM or RAM file that doesn't exist. In this mode you can only write data to the file; you cannot read from the file.

For your reference, the chart in Table I-1-1 summarizes the relationships among access methods, device types, processing modes, and the direction of data transfer.

Record Formats

The INFOS system supports four different record formats -- Fixed, Variable, and Undefined lengths, and Data Sensitive. ISAM and DBAM files use Variable length records for maximum flexibility and processing efficiency. For the same reason, you may only use Fixed length records in RAM files. SAM files, however, can use any of the formats, allowing you to process magnetic tapes generated on other systems and make the most efficient use of your storage space.

Fixed Length Record Format

Each record in a SAM or RAM file with a Fixed Length record format is exactly as long as every other record. RAM uses this format exclusively because it provides the fastest access to your data. The system doesn't have to read all your records to find the one you want, nor does it have to do any special computation to find that record.

For example, say that you are setting up an inventory file and each record will contain the same amount of data about each part on file: the part number, a brief verbal description, the price, and the quantity you have in stock. Each time you buy or sell those parts, you're going to want to update your file. Since the records are all the same length and you want to access them randomly to update the data, a file with fixed length records (i.e., SAM or RAM) would be ideal. It will allow you to maintain the file easily and retrieve specific information as quickly as you need it because the computer can get directly to your data via the part number.

Variable Length Record Format

When you create a SAM or ISAM/DBAM file with Variable Length records, the system will ask you to specify a *maximum* length for the records in that file. After that, no two records need to be the same length, but none can exceed your specified maximum. (The INFOS system keeps track of both the maximum record length for the file, and the exact length of each record in the file.)

You can use only this format for ISAM and DBAM files because each record occupies exactly the space necessary to hold it. So you get maximum flexibility in building and maintaining your files, as well as optimum use of your disk space.

Table I-1-1. Processing Modes

		FILE EXISTS		FILE DOES NOT EXIST	
		INPUT	UPDATE	OUTPUT	CREATE UPDATE
		READ ONLY	READ & WRITE	WRITE ONLY	READ & WRITE
SAM	DISK	YES	YES	YES	YES
	TAPE	YES	NO	YES	NO
	LINE PRINTER	NO	NO	(See note)	NO
	INTERACTIVE TERMINAL	YES	NO	(See note)	NO
RAM - DISK ONLY		YES	YES	YES	YES
ISAM - DISK ONLY		NO	YES	NO	YES
DBAM - DISK ONLY		NO	YES	NO	YES

NOTE: You may only open *existing* peripheral files for output.

You can use this format very effectively in setting up a customer record file. Since you have more information about long-time customers than you do about new ones, the records in the file will be of varying lengths. A Fixed Length record format would waste space every time you put a new customer record in a space designed to hold a long-time customer record; and it wouldn't allow you to expand your record size as you acquired more information about your customers. The Variable Length format, however, will allow you to pack as many records as will fit into the available disk space, with very little wasted space. The system automatically allocates the "right-sized" space when you want to enter a new record (or add information to, or delete it from, an existing one), and then places the record in that space.

Undefined Length Record Format

This format allows you to treat your file as a sequence of bytes, rather than specific-length records. Therefore you can append new data onto the end of the file, or read any section within the file, regardless of the size of the individual records. Obviously, this gives you more flexibility in reading records.

For instance, you may want to read a file whose record format you don't know. If this happens, simply define the file as a SAM Input file with Undefined Length records and specify an expected *maximum* record size.

Note that you should use this format sparingly because of the way the computer transfers Undefined Length records. (We'll give this more detail in the "Record Packing" section of this chapter.)

Data Sensitive Record Format

The length of each record in this format is determined by the occurrence of a delimiter table. The INFOS system will automatically terminate a record when it encounters either a carriage return (`\n`), a line feed (`\f`), or a null character (`\0`). However, you may also use the delimiter table to specify another ASCII character as a record length delimiter for your application. (See the CTR instruction description in the *Programmer's Reference Manual, ECLIPSE-Line Computers* for details on how the system uses translation tables.)

The Data Sensitive record format can be particularly useful when you want to read or write a SAM file on an interactive terminal, and/or when you want to output data from a SAM file to a line printer. For example, if you wanted to enter data into your file from an interactive terminal, you would first define the file as a SAM Input file with Data Sensitive records. Then you would specify the maximum length of your records as *n* characters. Unless you specify some other character for use as a record terminator, the INFOS system will automatically transfer a record to your program each time you depress the carriage return on your terminal.

Table I-1-2 summarizes the relationship between record formats and access methods.

Table I-1-2. Record Formats and Access Methods

Access Method	Available Record Format(s)
SAM	Fixed, Variable, Undefined, Data Sensitive
RAM	Fixed only
ISAM	Variable only
DBAM	Variable only

Data Transfers

When you create an INFOS file, you must specify the length of the data you want transferred between the file and the computer's main memory. This is called the *block size*. Also each time you open an INFOS file to process it, you can specify the number of buffers to be used for the processing run. For a tape file, each buffer you allocate is exactly the same length as the block size you specified when you opened the file. For a disk file, you must allocate buffers which are integer multiples of 512 characters long, because that is the size of the physical sector of the disk where the data is stored. In other words, when you are using *disk* files, you will specify your block size, then the INFOS system will transfer the data from the smallest number of sectors which contain an equal or greater number of characters than your block specification.

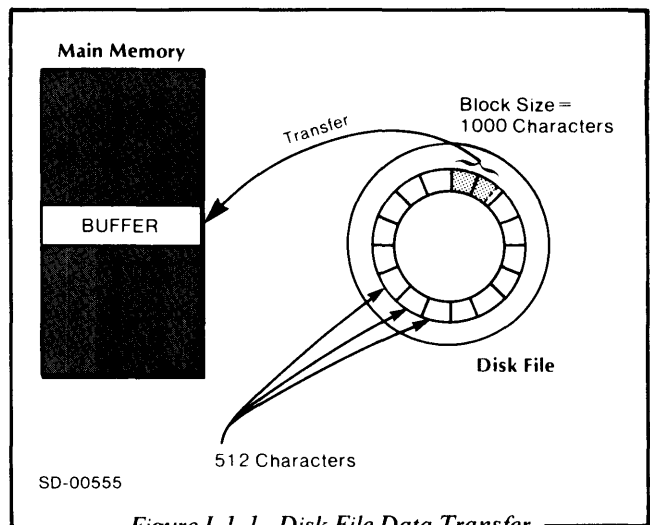


Figure I-1-1. Disk File Data Transfer

Licensed Material - Property of Data General Corporation

For example, if *you* specify a block size of 1,000 characters for a disk file, the *system* will transfer the data from two 512-character sectors of the disk to the computer's main memory. In other words, it will use one buffer of 1024 characters to hold the information you wish to process, even though your data may not completely fill that buffer.

Note that buffer space is *not* taken from your program area. The INFOS system allocates the buffers in the computer's free space, i.e., that memory area not occupied by application programs, operating system code, or INFOS system code. See Chapter 6 for further information about the allocation of buffer space.

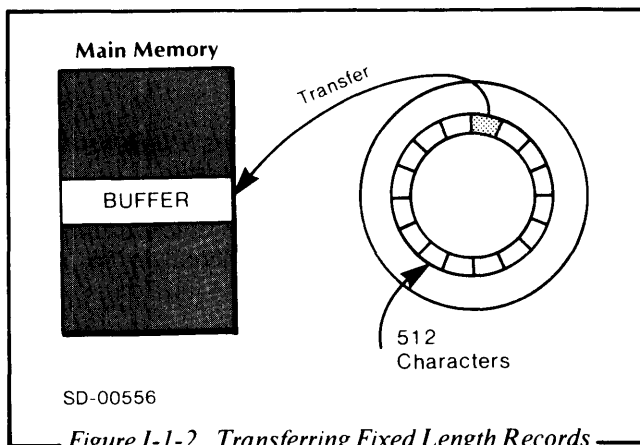
Record Packing

In general, the number of records you can pack into a block depends on the length of the records and the size of the block. You will know both the maximum record length and the block size, since you specify the record format (which helps determine the record length) and the block size when you create/open the file.

As we just mentioned, you can only transfer data from disks in blocks which are integral multiples of 512 characters, but you can transfer data from nondisk devices in fixed *or* variable length blocks. Let's briefly look at how the INFOS system transfers your data as blocks.

Transferring Fixed Length Records

The INFOS system always transfers Fixed length records in Fixed length blocks. Furthermore, Fixed length records require exactly as many bytes in a file as in main memory.



Transferring Variable Length Records

If you want to transfer Variable length records from a nondisk device to a disk, the system will place as many of those records as it can into the space you specify as the block size. The block size of a disk file is always an integral number of disk sectors. Therefore, the INFOS system transfers Variable length records in much the same way as it transfers Fixed length records, except that it allows four (4) bytes more per file block for Variable length records than for Fixed because of system overhead.

Transferring Undefined Length Records

If you specify that a SAM Input file has Undefined length records, the INFOS system considers them to be unblocked. This means that the system considers each block to be one record, and it will transfer only one record at a time. Therefore, you should use this format sparingly.

Undefined length records give you great flexibility because they allow you to process files (especially tape files) created on another operating system and to transfer data between character-oriented devices and the INFOS system. However, this method of transferring blocks of data can be wasteful or inefficient.

Transferring Data Sensitive Records

Data in Data Sensitive record files is always transferred in fixed-length blocks, but, unlike Fixed length records, Data Sensitive records can span blocks. That is, the system sees Data Sensitive records simply as strings of characters terminated by a delimiter character. Therefore, it will fill blocks of Data Sensitive records much more compactly than other blocks. This format is very useful when you want to print your data directly from your file. You just insert a delimiter character every 80 or 132 characters (for a line printer), regardless of the record bounds, and the system will transfer the data in the correct size for the device.

Buffer Management

Because the INFOS system lets you request any number of buffers, it uses a buffer management technique called the Least Recently Used (LRU) method for SAM and RAM processing. For ISAM and DBAM processing, the system uses a variation of the LRU method called hierarchical modulation (see Chapter 4 for further details).

When you specify more than one buffer for a processing run, the LRU method keeps track of how long ago each buffer was used. If all the buffers are in use when you issue a read or write request, the system writes the contents of the buffer you accessed least recently to the file, then places the current data in it.

The INFOS system also uses a “read ahead/write behind” technique for greater efficiency in processing SAM files. This means that as you are processing a sequential file, the system anticipates the data you will want next and brings it into main memory before you call for it. It then holds this data in extra buffers until you want it. The “write behind” process proceeds in just the opposite way. The system realizes that you don’t have any further use for a sequential record which you have finished processing, so it removes processed records from main memory, allowing you to proceed uninterrupted with other records and leaving space for them to occupy.

Footnotes to Data Transfers

1. The INFOS system will not necessarily initiate a data transfer every time you issue a read or write request. This is due partly to the number of records in a block, and partly to the way that the system manages the buffers. For example, if you want to read record 8501, the system will retrieve the block containing that record and transfer it to its first buffer. Then you may want to read record 8503. If that record is in the block which contains record 8501, the system does not need to transfer the data to main memory because it has already done so. Record 8503 is currently in the first buffer.
2. When you begin to design your program, don’t forget to designate a Data Area to and from which the INFOS system can transfer each record you want to process. In this area, the data meets your program; on input, the system moves single records from a buffer to your Data Area, and, on output, it moves them from the Data Area to a buffer. Naturally, your Data Area should be large enough to hold the largest record you expect to process.
3. No matter what access method or peripheral devices you use, you can change the system-assigned values for timeout intervals to suit your needs. These system-assigned values vary for different devices, but, in each case, the system will close your file if it cannot complete a data transfer after twelve (12) of these intervals. We’ll discuss timeout intervals in more detail in Chapter 6.

A Word About Volumes

Throughout this manual, we will use the word “volume”. The INFOS system contains two types of volumes: physical volumes and logical volumes. Physical volumes are the recording media on which you store your file, e.g., disk packs or tape reels. Logical volumes are subsets of your file. For example, this entire manual consists of three files within one physical

Licensed Material - Property of Data General Corporation

volume. That is, we have a Planning file (i.e., Part One), a Programming file (Part Two), and an Appendix file (Part Three). Furthermore, within each file (Part) there are a number of chapters. These chapters are the logical volumes; each is a subset of the entire file (Part). Once you open a Part (file) you can access any of the chapters (logical volumes) within that Part. In other words, every INFOS file consists of at least one (and usually several) logical volumes.

This concept becomes more complicated when you realize that your entire file can reside on many physical volumes. An encyclopedia can illustrate this well. Each book (physical volume) in the encyclopedia set (the whole file) contains many chapters (logical volumes), but it is only a part of the entire set. Therefore, the complete encyclopedia (i.e., the whole file) consists of many, many chapters (logical volumes) and is contained within many books (physical volumes).

To keep things as simple as possible, whenever we use the word “volume” in this manual, we will mean logical volume unless we specifically say physical volume.

Multivolume Files

You can design your INFOS file to occupy as much or as little space as you need. If your files will be on disk, you can let:

- 1 disk contain 1 file, or
- many disks contain 1 file, or
- 1 disk contain many files, or
- many disks contain many files.

If you want to work with labeled tapes, you can let:

- 1 reel of tape contain 1 file, or
- 1 reel of tape contain many files, or
- many reels of tape contain 1 file.

NOTE: You cannot create or process a tape set which contains many files on many reels.

Unlabeled tape files and files on interactive terminals or a printer are normally considered to be single volume files.

Multivolume files are useful because they remove the restrictions on file size which would normally constrain you under RDOS. That is, RDOS files can only exist on one disk (comprising at most 64K (65,535) sectors), or on one reel of tape. Your INFOS file, however, can reside on up to 255 separate devices, increasing the number of disk sectors you can use for file storage from 64K to over 16 million. Looking at it another way, this increases your maximum disk file size to about eight billion characters.

Licensed Material - Property of Data General Corporation

How to Process INFOS Files

How you process your INFOS file depends on whether you want to create a new file or access records from an existing file.

How to Create a New File

When you want to create a new INFOS file, you must:

1. Define its characteristics, that is, specify the access method and record format you want, as well as the physical characteristics of each volume. (When you want to create ISAM or DBAM files, the index - as well as the database - is also a file. This means that you must define the features of the index and all its volumes along with the database and all its volumes.)
2. Select the INFOS options you want; for example, code translation and data verification.
3. Choose the file attributes you need, such as block size and number of buffers.
4. Open the file.

When you have completed these four steps successfully, you will have a fully-defined INFOS file to which you can begin writing data.

You can also create a new INFOS disk file with the ICREATE utility.

Line Printers and Terminals as INFOS Files

Files on line printers and interactive terminals are characteristically single volume files. If you want to use such a file, follow steps, 1, 2, and 3 above, but choose a processing mode based on the direction in which you want to transfer data. If you want to transfer data from a source *to* a line printer or an interactive terminal, open the file in the *Output* processing mode; if you want to enter data *from* an interactive terminal, open the file in the *Input* processing mode.

File Naming, Briefly

The name you give to the first (or only) volume of a SAM or RAM file you have created becomes the whole file's name. ISAM and DBAM files take their symbolic names from the name you give to the first (or only) volume of their *indexes*. You can find more information on file naming in Chapter 6.

How to Process an Existing Tape File

You open an existing tape file in essentially the same way that you create one. That is, follow steps one through three above, then open the file in the *Input* processing mode so that you can read its records.

NOTE: When you are defining the volumes, remember that unlabeled tapes are normally single volume and that labeled tapes are most often multivolume. Don't forget to define *all* the volumes.

How to Process an Existing Disk File

When you create a disk file, the INFOS system automatically produces a permanent file specification which contains the permanent attributes you define at file creation - block size and number of buffers, for example. The system also records the definitions for each volume of the file. While processing an existing disk file, you can temporarily override some of the attributes of the file and select INFOS options for use in the current processing run, but you cannot change any of the file's permanent characteristics, nor any of the volume definitions. For example, you *can* specify a different number of buffers for use in the current processing run, but you *cannot* change the access method, record format, or block size that you specified when you created the file. You will find further discussion of the things which you can and cannot change in Chapters 2, 3, 4, and 5.

Since the system already has the file specifications, all you need to do to open a disk file is:

- For SAM and RAM files:
 1. Specify the file's access method and record format;
 2. Give the file's name - that is, the name you gave to the first volume;
 3. Open the file in the Input or Update processing modes.
- For ISAM and DBAM files:
 1. Specify the file's access method and record format;
 2. Give the file's name - that is, the name you gave to the first volume of the file's index;
 3. Open the file in the Update processing mode.

Summary

This chapter has presented all the introductory considerations you need to know before you begin to plan your particular application of the INFOS system. The rest of this manual will give you the details of the points raised here.

End of Chapter

Chapter 2

Sequential Access Method (SAM) Files

Sequential access files are just what they sound like: the system writes your records one after the other and reads them back according to their physical sequence. Furthermore, INFOS SAM files give you five additional features which make them stronger than the average SAM:

1. Choice of file format - You can set up your records in any of the four INFOS formats -- Fixed, Variable, Undefined, or Data Sensitive -- to accommodate your particular application.
2. Use of most peripheral devices - You can create or store your files on any Data General disk device, magnetic tape, interactive terminal, or line printer. So you don't have to worry about processing your files once you've selected a device.
3. Reduction of I/O wait time (Multibuffering) - You can process your INFOS SAM files very efficiently because when you specify more than one buffer the system automatically overlaps the I/O operation of your program with its data processing functions. That is, the system anticipates your data requirements and brings the next block of data on your access device into a second buffer while it is processing the information in the first buffer. (Remember that the system allocates space for all buffers in the extended memory area, not in your program space.) Also, depending on the type of device on which you're storing your data and the direction in which your program is transferring that data, you can choose the INFOS processing mode which will be most effective for you.
4. Variety of processing options - All SAM processing functions don't apply to all devices (e.g., you can't read from a line printer) but, where appropriate, they do let you read, write, and rewrite data,

append new records to the end of a file, and overwrite a file. The INFOS SAM also has a "Read-After-Write" option which you can use to verify the accuracy of each record written to a SAM tape or disk file.

5. Code translation - INFOS SAM files allow code translation on both input *and* output. That is, the INFOS system can automatically convert any character set you have into any other character set. The INFOS system can do this for an entire record or for certain fields within that record.

The INFOS SAM is not the SAME old thing. The INFOS system significantly increases the number of functions which you normally have under RDOS. It also removes artificial limits on your file's size; your file can exist on up to 255 separate volumes. But these are just the general features of an INFOS SAM file. If you put your SAM file on a disk, you get even more.

SAM Disk Files

To give your SAM file maximum flexibility, put it on a disk. Using a disk will allow you to do input *and* output when you create or use the file. It will also allow you to use the SAM Rewrite Option (as described later in this chapter). When you open an existing SAM file, you need only minimum specifications; the INFOS system automatically uses the information in the permanent file specification. Also, if your file is on a disk, many users can share access to it, using single- or multitask programs, because the INFOS system automatically resolves conflicting requests for data. You may, however, request exclusive use of your disk file whenever you open it. The system will then prevent other programs from accessing your file until you release it or close it.

How to Create SAM Disk Files

To create a SAM disk file, you can use either the Output or Create Update processing mode. (If you also want the ability to read and rewrite records, choose the Create Update mode and specify the Rewrite option.) You can then choose any of the four INFOS record formats. If you choose Fixed Length records, you must specify their exact size; if you choose Variable Length, give the expected size of the longest record in the file. Data Sensitive records allow you to specify your own record terminator, but if you default this choice, the system will automatically terminate a record when it encounters either a carriage return (␣), null (0), or line feed.

Next, you can either indicate a specific block size or let the system automatically set the size at 512 characters. You can also, if appropriate, indicate that your records are unblocked, in which case the system will put just one record in each block.

SAM disk files allow you to specify any number of buffers, and each one will be as long as your block size. However, if you specify two or more buffers, you will greatly reduce your I/O wait time because the system will automatically keep them active. (If you don't specify a number of buffers, the system will only give you one.) Also, if you want to offset your data records from the beginning of a buffer, you can simply specify initial data offset.

Your SAM disk file can consist of as many volumes as you need, but you have to define each volume when you create the file. Part of this definition involves naming each volume, and the name you give to the first volume becomes your file's symbolic name. Another part of the definition allows you to choose an ASCII pad character for each volume. If you don't care what character is used, simply default this choice and the INFOS system will automatically use the null character (0).

You also have to choose whether you want to allocate the file's space in each disk volume contiguously or randomly, and whether you want to initialize the file space. (Initialization reserves disk space and sets all bytes in that space to zero.) We recommend that you specify Disable File Initialization, since initialization of a 64K block volume takes approximately twenty minutes. If you choose contiguous allocation, you must specify the number of physical blocks you want to allocate. If you choose random allocation, you can specify a volume size or you can let the system allocate space as it is available - up to a maximum of 65,535 disk blocks. (Note, however, that if you run out of disk space before a logical volume is full, the system will not send the overflow to the next volume. Refer to Chapter 6 for a more detailed explanation of disk space allocation and volume size.)

Finally, unless you specify a different interval, the INFOS system automatically assigns timeout intervals of three seconds for fixed-head disks, five seconds for moving-head disks, and 15 seconds for magnetic tapes.

Those are the choices and/or specifications you have to make when you want to create a SAM disk file. The chart in Table I-2-1 summarizes these steps and indicates whether they are mandatory or optional.

Table I-2-1. Steps in Creating a SAM Disk File

You Must Do These:	You May Do These:
<p>Choose a processing mode</p> <ul style="list-style-type: none"> ● Create Update or Output <p>Choose a record format</p> <ul style="list-style-type: none"> ● Fixed, Variable, Undefined, Data Sensitive ● Specify exact length of Fixed Records ● Specify expected maximum length of Variable length records <p>Specify block size</p> <p>Choose a number of buffers</p> <p>Define each volume</p> <ul style="list-style-type: none"> ● Name, characteristics, etc. <p>Specify Random or Contiguous space allocation</p> <p>Specify number of physical blocks for Contiguous allocation</p>	<p>(Specify Rewrite option)</p> <p>(Specify delimiter(s) for Data Sensitive records)</p> <p>(default = 512 characters/block)</p> <p>(Indicate records are unblocked)</p> <p>(default = 1 buffer allocated)</p> <p>(Override default pad character)</p> <p>(Specify volume size for Random allocation; default = 65,535 blocks)</p> <p>(Specify initialization or no initialization)</p> <p>(Specify value for timeout intervals)</p> <p>(Specify initial data offset)</p>

How to Open an Existing SAM Disk File

You'll follow the same basic steps when you open an existing SAM disk file as you did when you created it. That is, you must:

- Choose a processing mode; either Input or Update.
- Specify the same record format you used when you created the file. (If you want to, you can specify the Undefined Length record format, but remember that records in this format are only transferred one at a time into the buffers.)
- State the name of the first, or only, volume.

You may specify a different number of buffers for this processing run than you did at creation. However, if you want to use the number you originally specified, don't do anything. The system will automatically allocate the number recorded in your file's permanent specification.

There are also three other features you may use when you open your file, but you must specify them each time you want them:

- code translation (ASCII to EBCDIC, or vice versa, or either one to your own code).
- data verification (you have to open in the Update processing mode to use this).
- exclusive file use (as described in the last section of this chapter).

Finally, note that you cannot change any of your file's *volume* characteristics when you open it.

That's all there is to opening an existing SAM disk file. Before we explain how to process SAM files, however, let's look at how to create and open SAM files on other devices.

SAM Labeled Tape Files

The INFOS system will let you use magnetic tape labels which conform to ANSI standard levels 1, 2, and 3, and IBM standard levels 1 and 2. Using these labels, you can create and process labeled tape files in the following forms:

- single file/single volume
- single file/multivolume
- multifile/single volume

NOTE: You *cannot* create or process a multifile/multivolume tape set.

This section of Chapter 2 describes the general principles involved in initializing, creating, and opening labeled tape files. If you want further details on any of these points, refer to Appendix A at the back of this manual.

Initialization

Before you can create a labeled tape file or open an existing one, you have to initialize your tape reels. The Labeled Magnetic Tape Initialization Utility (call: LBINIT), described in the *INFOS Utilities Users' Manual*, is the easiest and most efficient way to initialize at runtime. However, you can also initialize (and release) tapes through your programs with INFOS system requests.

How to Create SAM Labeled Tape Files

Creating a SAM labeled tape file is not unlike creating a SAM disk file, except that you'll use only the Output processing mode. After you select Output mode, you need to specify the record format you want to use - Fixed, Variable, or Undefined. As with SAM disk files, you have to give the exact record length if you use Fixed length records. If you want to use Variable length records, you can either specify the length of the largest record, or let the system assume that the largest record is four characters less than your block size. However, if you do specify a maximum size, it also cannot be any longer than four characters less than your block size.

Licensed Material - Property of Data General Corporation

And speaking of block size, that's the next choice you have. You can either specify a size, or you can default the choice - in which case the system will give you blocks which are 80 characters long. At this point you can also specify that you want unblocked records - that is, just one record per block. As usual, the system automatically gives you unblocked records if you choose the Undefined record format.

After your block choice, you'll have to choose the number of buffers you'll need. Again, if you don't give the system a specific number, it will give you only one. And if your data records don't start at the beginning of a buffer, you can specify the number of characters by which you want the system to offset them.

Next, you must specify the type and level of the labels you want to write (i.e., ANSI 1, 2, or 3; or IBM 1 or 2). Then you have the option of specifying the following:

- file set identifier
- file expiration date
- file sequence and generation numbers
- accessibility (data security) code

You also have the same option for selecting recording code translation as you have with a SAM disk file. That is, you can convert your data from any recording code you have into any other code.

Your next step is to define each volume (i.e., each physical reel of tape). In other words, if your data file will exist on three reels of tape, you must define all three reels. The first part of this volume definition will be the volume name, consisting of a volume identifier followed by a colon, then a file identifier followed by a null character - for example, MTO: PAYROLLO. The next part of the volume definition includes a volume label, header labels, trailer labels, and, depending on which label type and level you are using, user labels. You can also specify that the system rewind each volume on opening, and you can enable runtime initialization and release for each volume.

Finally, while you're at it, there are four other options you can use for each volume. First, you can specify fixed or variable length blocks. Second, you can choose a pad character other than the system-supplied null (0). Third, you can specify a volume accessibility character. And, fourth, you can choose a different timeout interval than the system-assigned interval of 15 seconds.

That may seem like a lot to do just to create a labeled tape file, but it's not really. It's simply that the INFOS system gives you quite a few options to allow you to use a labeled tape file as effectively as possible. For your reference, the chart in Table I-2-2 shows the mandatory steps and the optional ones.

Table I-2-2. Steps in Creating a SAM Labeled Tape File

You Must Do These	You May Do These
<p>Specify Output processing mode</p> <p>Choose a record format</p> <ul style="list-style-type: none"> ● Fixed, Variable, Undefined, or Data Sensitive <p>Specify exact length of Fixed records</p> <p>Choose a block size</p> <p>Choose desired number of buffers</p> <p>Specify label type and level</p> <ul style="list-style-type: none"> ● ANSI 1, 2, 3 or IBM 1, 2 (See Appendix A) <p>Define each volume</p> <ul style="list-style-type: none"> ● Name, volume label, header labels, trailer labels 	<p>(Specify length of longest Variable record)</p> <p>(Default = 80 character block; maximum = 8192 bytes)</p> <p>(Specify unblocked records)</p> <p>(Default = 1 buffer)</p> <p>(Specify number of characters for initial data offset)</p> <p>(Specify file set identifier)</p> <p>(Specify file expiration date)</p> <p>(Specify file sequence)</p> <p>(Specify generation numbers, accessibility code)</p> <p>(Specify code translation)</p> <p>(Specify rewind tape on open)</p> <p>(Enable runtime initialization and release for each volume)</p> <p>(Choose Fixed or Variable length blocks for each volume)</p> <p>(Select pad character - default = 0)</p> <p>(Specify timeout interval)</p>

Licensed Material - Property of Data General Corporation

How to Open an Existing SAM Labeled Tape File

There are two differences between the opening procedures for an existing SAM labeled tape file and the creation procedures we just explained. First, you must use either the Input or Update processing mode. (Choose Update when you want to append new records to the end of your file.) Second, you must do the following:

- Initialize the tape units;
- Define the file's characteristics;
- Indicate the attributes and INFOS options you want; and
- Define each volume in the file.

Other than this, the procedures are exactly the same. We'll tell you about your processing options for this type of file in the last part of this chapter.

How to Use Peripheral Devices as SAM Files

Here's a handy feature for you. The INFOS system allows you to define a line printer or an interactive terminal as a SAM file. This means that you can use the read and write functions of the INFOS system to transfer data between your program and these peripheral devices. Or, to put it another way, you can transmit data between your program and the terminal, or send data generated by your program to a line printer.

You can also use a single terminal for both input and output. Normally, SAM limits your data transfers to a single direction per file. But you can use the single terminal for both input and output by defining two files like this:

1. Define and open one file as a SAM Output file. When you specify the volume definition, use the system device name \$TTO (\$TTO1 in the foreground). This will allow you to transmit data from your program to the terminal.
2. Define and open the other file as a SAM Input file and use the system device name \$TTI(1). This will allow you to enter data from the same terminal.

If you want to write data to a line printer, simply define and open a SAM output file using the name \$LPT.

You have essentially the same INFOS option for peripheral files as you do for other SAM files. For example, your program can read a SAM file with IBM level two labels and select portions of the data records for output to a \$TTO file. Or the system can automatically translate records from EBCDIC to ASCII during transmission to the terminal.

You'll find more information on physical files and their characteristics in Chapter 6.

What to Do with Unlabeled SAM Tape Files

The INFOS system also allows you to create and process unlabeled SAM tape files. This can be helpful if you want to process a tape file generated under RDOS or on an external system. If you have a tape and you don't know what's on it, merely define and open it as a SAM Input file with Undefined records and specify a relatively large block size (say, 2,500 characters). If you mount the tape on tape unit zero, name the file MT0:0. Then, after you open the file and issue your first read request, the INFOS system will transfer whatever is on the tape until it comes to the first interrecord gap, or until the end of the file, whichever comes first. Again, Chapter 6 has more details on this topic.

SAM Summary

The charts in Tables I-2-3 to I-2-6 summarize the options you have when you open a SAM file. Charts A and B (Tables I-2-3 and I-2-4) show the SAM *file* definition options, and charts C and D (Tables I-2-5 and I-2-6) show the SAM *volume* definition options. After these charts, you'll find the section on how to process SAM files.

Table I-2-3. File Definition Options When You Open a New SAM File

Chart A	Create Update Mode	Output Mode				
	Disk	Disk	Labeled* Tape	Unlabeled Tape	\$TTO	\$LPT
Record Format <ul style="list-style-type: none"> • Fixed • Variable • Undefined • Data Sensitive 	Choose one of the four Record Formats. If you use Data Sensitive, the system will terminate your records by 0,), or , unless you specify other delimiters.		Choose from: Fixed Variable or Undefined only	Choose one of the four record formats.	Data Sensitive (with system-supplied terminators) is the usual format for these devices, but you may also used Fixed and Undefined.	
Block Size	Specify any size block up to 8K bytes. The system will transfer your data in multiples of 512.		Specify any block size you want. If you don't specify a size, the system will use 80 characters. (80 characters is normal for \$TTO; 80 or 132 characters is normal for \$LPT.)			
Blocked or Unblocked Records	Specify unblocked records if you want only one record per block. If you want more than one record per block, you need make no specification. (Note that Data Sensitive records can span blocks and Undefined records are always unblocked.)					
Number of Buffers	Choose any number of buffers. If you specify more than one, the system automatically keeps them filled, greatly reducing your I/O wait time. If you do not specify any buffers, the system will give you only one. For nondisk files, buffer size = block size. For disk files, buffer size equals the next highest multiple of 512 which equals or exceeds block size.					
Data Offset	If your records do not start at the beginning of a block, specify the number of characters by which you want the system to offset the first record.					
Record Length	If your file has Fixed Length records, you must specify the record length. If it has Variable Length records, specify the expected length of the longest record. This length cannot exceed four characters less than the block size. If you do not specify a maximum length, the system assumes that the longest is four characters less than your block size.					
Overwrite or Append Option	This allows you to write at some point other than end of file; however, you will lose whatever existed beyond that point.	Not Applicable				
Rewrite Option	Allows you to update in place using read/rewrite and read/release command sequences.	Not Applicable				
Exclusive Use	If you specify exclusive use, no one else can access the file while you have it open.	Not Applicable				
Code Translation	The system can automatically translate your data from ASCII to EBCDIC, or vice versa, or to your own code during transfers between your program and the buffers.					
Read-After-Write Verification	You can verify each block transferred to your file for recording accuracy.	Not Applicable				
Number of Volumes	Specify the number of volumes if your file will reside on more than one volume. If you do not specify a number, the system assumes that it is a single volume file and will only process the first volume definition.					

NOTES:

*You must also specify the label type and level (i.e., ANSI 1, 2, or 3 or IBM 1, or 2) when you are working with a labeled tape file in the Output mode. In addition, you may specify the File Set Identifier, the Expiration Date, the Sequence and/or Generation Number, a File Accessibility Code, Runtime Initialization, and/or Runtime Release.

Table I-2-4. File Definition Options When You Open a New SAM File

Chart B	Update Mode	Input Mode
	Disk	\$TTI
Record Format <ul style="list-style-type: none"> ● Fixed ● Variable ● Undefined ● Data Sensitive 	Choose the same format you used to create the file. The system will then verify this by comparing it with the permanent file specifications. If it doesn't match, an error will return.	Data Sensitive (with system-supplied terminators) is usual format but you can also use Fixed or Undefined.
Block Size	Do not specify a block size; the system uses the permanent file specification.	Specify the same block size you used to create the file. If you don't specify a size, the system uses 80 characters. (Normal block size for \$TTI is 80 characters; for tape files with Undefined Length records, specify a large block size.)
Blocked or Unblocked Records	If you specified unblocked records (i.e., one record per block), when you created the file, do the same here. Otherwise, the system will transfer as many records as can fit into a block. (NOTE: Data Sensitive records can span blocks.)	
Number of Buffers	Specify any number of buffers. If you don't specify any, the system will give you the number recorded in the permanent file specification.	Specify any number of buffers. If you don't specify any, you will get only one. If you specify more than one, the system automatically keeps them filled.
Data Offset	Do not specify an offset. The system will use the offset recorded in the permanent file specification, if any is there.	Specify the same offset as you did at creation, otherwise you'll probably get unrecognizable data transferred to your program.
Record Length	Do not specify a record size. The system will use the permanent file specification.	Specify the record size you used when you created your file.
Overwrite or Append Option	Specify overwrite if you want to use it on this opening (see Create Update Chart A for cautions).	
Rewrite Option	Specify Rewrite if you want to use it on this opening. Operates same as under Create Update.	
Exclusive Use	Specify Exclusive use if you want it for this opening. (This is not kept in permanent file specifications.)	
Code Translation	Same as when opening a new SAM file.	
Read-After-Write Verification	Specify this if you want it. (Not part of the permanent file specification.)	
Number of Volumes	Do not specify any number of volumes. The system will use the permanent file specifications.	

Table I-2-5. Volume Definition Options When You Open a New SAM File

Chart C	Create Update Mode	Output Mode				
	Disk	Disk	Labeled Tape*	Unlabeled Tape	\$TTO	\$LPT
Volume Name	You must specify a name for each volume when you create a file. The name you give to the file's first (or only) volume becomes the file's symbolic name.					
Allocation Technique ● Random ● Contiguous	Specify Random or Contiguous allocation for each volume in the file. Then specify the number of 512 character blocks which you want the system to allocate. If you do not choose a technique and a number, the system will randomly allocate up to 65,535 blocks per volume.					
Timeout Interval	The system assigns each device a timeout interval, but you may assign any interval you want on a per volume basis. If the system cannot complete an I/O transfer after 15 intervals, it will give you an end-of-file condition.					
Block Format	Disk files always have Fixed Length blocks. (Refer to Chapter 6 for further details.)		Specify Fixed or Variable length blocks.			
Parity			The INFOS system normally generates odd parity. You must specify even parity if you want it.			
Volume Label				Specify that this is a volume of an unlabeled file.		
Pad Character	Specify the character you want INFOS to use to fill the unused portion of a block. If you don't specify a character, the system will use the nonprintable ASCII null (0) character.					

NOTES:

*You may also specify the following for each volume of a labeled tape file which you are processing in the Output mode:

- Volume Accessibility
- Volume Owner Identifier
- 1-9 UVL, UHL, UTL labels (depending on your file's label type and level)
- Runtime Initialization/Release
- Rewind on open

Table I-2-6. Volume Definition Options When You Open an Existing SAM File

Chart D	Update Mode		Input Mode		
	Disk	Disk	Labeled Tape*	Unlabeled Tape	\$TTI
Volume Name	Specify only the name of the first (or only) volume.		Since \$TTI and Unlabeled tapes are generally single volume files, and Labeled tapes are multivolume files, you must name and define each volume each time you open it.		
Allocation Technique • Random • Contiguous	Do not specify an allocation technique. The system will use the permanent file specification.		Not Applicable		
Timeout Interval	Do specify a timeout interval. The system will use the permanent file specification.		The system assigns each device a timeout interval, but you may assign a different one on a per volume basis. If the system cannot complete an I/O transfer after 15 intervals, it will give you an end-of-file condition.		
Block Format	Do not specify a block format. The system will use the permanent file specification.		Specify the same block format you specified when you created the file. If you don't know the format, specify variable blocks.		
Parity			If you specified even parity when you created the file, specify it for this opening.	Not Applicable	
Volume Label				Specify that this is a volume of an unlabeled file.	Not Applicable
Pad Character	Do not specify a pad character. The system will use the permanent file specification.		Not Applicable		

NOTES:

*For each volume of a labeled tape file which you are processing in the Input mode, you may also specify Runtime Initialization/Release and/or Rewind on Open. The system will automatically return the Volume Accessibility and Volume Owner Identification to your program if the volume contains them. And if the volume has any user labels, you can also designate the areas in your program which will receive their contents.

How to Process SAM Files

The INFOS system gives you five basic processing options for a SAM file. You can:

- Read an existing file
- Write a new file
- Append records to the end of an existing file
- Rewrite existing records
- Overwrite a file

Reading an Existing File

When you want to read an existing SAM file, open it in the Input Mode. Your file can reside on any device - disk, interactive terminal, labeled or unlabeled tape - and you can read all of it or only a part. The system will return the length of each record you read, and, after you read the last record, will send you an end-of-file message.

Writing a New SAM File

This function is as simple as reading a file. Simply open the file in the Output mode and specify the length of each record as you write it. As in reading, you can use all devices to write a file - disks, labeled or unlabeled tapes, interactive terminals, or line printers.

Appending Records

If you want to append new records to a file on tape or disk, open the file in the Update mode and issue a write request. This will position you to the end of the file and you can write as many records as the device will hold. If you are working with a labeled tape, the system will automatically place the existing labels from the end-of-file section and the end-of-file label groups in their appropriate places when you close the file.

Rewriting Existing Records

Sometimes you'll want to examine your records and change some of them. To do this, open your disk file in the Update mode, specify the rewrite option, and issue your first read request. After you read each record, you can either leave it unchanged or rewrite it. If you do not want to modify the record, issue a release request; if you want to change, update, or replace the record, issue a rewrite request, then rewrite the record. If your file has Variable length records, make sure that each rewritten record is exactly as long as the one it's replacing. You can also lock the record you are rewriting, preventing other users from accessing the record while you're examining or rewriting it. To use this feature, indicate that you want the record locked each time you issue a read request. The system will then keep the record locked until you issue the next rewrite or release request.

Overwriting Existing Records

The INFOS system also allows you to read records up to any point in your disk file, then write new records over those which exist after the last record you read. To do this, open the file in the Update or Create Update mode and select the overwriting option. You can then read to your desired point in the file and issue a write request. The end of the last record you write then becomes the new end-of-file and you will, therefore, lose any records which previously existed beyond it.

For example, if you open your file and issue several read requests before you issue your first write request, the system will preserve the records you read and the record you write will become the new end-of-file. From this point in the file on, all your subsequent write requests will append new records to this new end-of-file and you will lose any records that were in that part of the file.

Additional Features

When you want to process a SAM file, you'll use one of the five functions described above. In conjunction with these, however, the INFOS system gives you two additional features - Point and SETX - which make the five basic procedures more useful.

Point

You'll use the Point function when you want to change your position in a disk file that you are processing. Here's how it works:

The INFOS system automatically keeps track of your current position within an open file. The system always sets this position just after the last record transferred and gives you feedback about the record's position in the file - i.e., the block number which contains the record and the record's location within that block. You can use the Point function to move from your current position to any other record in the file, as long as you know that record's position. To change positions, issue a Point request which gives your desired relocation position. Be careful when you use Point, however. If you issue a Point request in conjunction with an Overwrite, any record you write at your new location will become the new end-of-file and you will lose everything beyond that point.

You can use Point in any processing mode, but you'll probably find it most useful in Update and Create Update when you want to return to a record which you have read and/or rewritten earlier.

Licensed Material - Property of Data General Corporation

SETX and RELX

This INFOS SAM feature lets you have exclusive use of your file and prevents other users from opening it until you release it. You can gain exclusive access in two ways: either request exclusive use when you open the file or use the Set Exclusive Use (SETX) function request. If no one else has opened the file when you issue the SETX request, the system will give you exclusive access. If the file is already open, you'll get the error message "FILE IN USE". When you don't need to restrict access any further, issue a Release Exclusive Use (RELX) function request to return the file to general use. If you don't issue RELX, the system will not return the file to general use until you close it.

Summary

The INFOS system gives you a lot of flexibility when you're processing a SAM file. Depending on the device your file resides on and the processing mode you choose, you can do just about anything to your data. Furthermore, as you are processing data, the system will also let you know of any exceptional status (such as reaching the physical end of a volume or an unusual transfer length). In addition, if you selected Read-After-Write data verification, it will tell you when the data you have written to your file doesn't exactly match the contents of its originating buffer.

The chart in Table I-2-7 summarizes the available mode(s) and device(s) for each INFOS SAM file processing function.

Table I-2-7. SAM File Processing Summary

Function	Mode and Devices Available
Read	Create Update (D) Update (D) Input (D, L, U, T(I))
Write	Create Update (D) Output (D, L, U, T(O), P) Update (D, L, U)
Read/Rewrite Read/Release	Create Update (D) Update (D)
Point SETX/RELX	Create Update (D) Output (D) Update (D) Input (D)
Devices: D = Disk T(O) = Interactive Terminal (Output) L = Labeled Tape T(I) = Interactive Terminal (Input) U = Unlabeled Tape P = Line Printer	

End of Chapter

Chapter 3

Random Access Method (RAM) Files

The INFOS system's Random Access Method (RAM) lets you directly access any data record in your file without going through all the records before it. That is, unlike SAM files, you don't need to read a lot of records to get to the one(s) you want; you just jump in, read or write the record(s) you want, and jump out. RAM is simple, quick, and easy to use. And a single RAM file can reside on as many as 255 disks.

When you build a RAM file, chances are that your records will all be of the same type - for example, invoice, personnel, or inventory records - and each of these records will have some natural record number associated with it, such as invoice or part numbers. These natural numbers are the keys to RAM because the INFOS system stores and retrieves records by these numbers. So when you want to read or write a RAM record, you just give the record number to the system and it will go directly to the record. But be sure to give your records small numbers (i.e., 1, 2, 3, etc.) because the system assigns records to the physical space corresponding to the record number. For example, the system will put record number 100 into space number 100 on your disk; if this is your lowest numbered record, you will waste spaces 0 through 99. All in all, however, RAM is still the quickest and easiest access method.

To make things even easier, you can create a RAM file with the ICREATE utility and load it with the ICOPY utility, both of which are described in the *INFOS Utilities User's Manual*. And if you're initially loading your RAM file from a source file that is already arranged by record number, you'll have a very efficiently constructed RAM file.

Finally, to give you maximum efficiency, INFOS RAM offers several processing modifiers which can reduce your I/O wait time practically to zero.

How to Create a RAM File

To create a RAM file, first choose a processing mode. You can use either the Output or the Create Update mode, depending on the direction in which you want to transfer data. The Output mode only lets you write to the file, while the Create Update mode lets you both read and write. Both modes give you the same file and volume definition options. Also, no matter which mode you choose, you can only have Fixed Length records and you must specify their length.

Next, you should specify a block size in terms of a number of characters. You can use any size you want, but remember that the system always transfers data in blocks which are multiples of 512 characters. For example, if your records are 125 characters long and you want four records per block, you can specify a block size of 500 characters. But the system will transfer your records in a 512 character block; 500 of these will contain data and 12 will be unused. You can also default the choice of a block size and the system will automatically use a 512 character block. But be careful when you specify a block size; your choice of size can have a significant effect on the efficiency both of your data transfers and of your disk use. You can find further details on record packing in Chapter 6.

After you choose a block size, determine the number of buffers you'll need for I/O transfers. As with block size, you can specify any number of buffers, and if you use two or more you can also use RAM's Pre-read feature (described later in this chapter) to retrieve one record while you are processing another. If you don't specify a number of buffers, the system will give you only one. This could reduce your data transfer efficiency. (In general, buffers are like heads - two are better than one.)

When you are creating a RAM file, you must also specify the number of volumes in your file and name each one. The name you give to the first volume becomes the file's symbolic name. If you don't specify a number of volumes, the INFOS system will assume that the file has only one volume.

You can (and often should) also specify contiguous allocation for the first volume of a multivolume file. This will increase your processing speed for the original file. Then, for the second and subsequent volumes, you should default this choice and let the system allocate space on the disk randomly. When you choose contiguous allocation, you must also tell the system how many 512-character blocks to allocate. The system will then automatically erase all previous information from the blocks you specified, unless you tell it not to. When you specify random allocation, the system will allocate up to 65,535 blocks (of 512 characters) on an "as needed" basis unless you specify a maximum number of blocks for allocation.

In addition, you can specify a timeout interval for each volume you define, or you can let the system use its own - three seconds for fixed-head disks and five

Licensed Material - Property of Data General Corporation

seconds for moving-head disks. (See Chapter 6 for further details on timeout intervals.)

Finally, you have four other options to choose from when you are creating your RAM file. You can:

- use the SETX and RELX functions (described in the "Processing" section of this chapter) to gain exclusive use of your file either on opening or while processing;
- have the system automatically translate any recording code you have into any other recording code;
- request that the system verify each block of data written to your file for recording accuracy;
- specify a pad character for each volume, or by defaulting, let the system use the null (0) character.

This is all you must choose from and/or specify when you want to create a RAM file. In Table I-3-1 we've summarized these things in terms of mandatory ones and optional ones.

Table I-3-1. Steps in Creating a RAM File

You Must Do These:	You May Do These:
Choose a processing mode <ul style="list-style-type: none"> ● Create Update or Output Specify Fixed Length records Specify exact length of records Specify block size Specify a number of buffers Specify number of volumes in file <ul style="list-style-type: none"> ● Name each volume ● Specify Contiguous or Random allocation ● Specify number of blocks for Contiguous allocation 	(default = 512 characters) (default = 1 buffer) (Specify maximum number of blocks for Random allocation) (Specify length of timeout interval) (Request SETX function) (Request code translation) (Request data verification)

How to Open an Existing RAM File

The process for opening an existing RAM file is very much like the one for creating it. However, opening an existing file is simpler because the system-generated Permanent File Specification contains such things as the length of the file's records, the number of buffers you originally wanted, all volume specifications, the allocation technique, the length of the timeout intervals, and the pad character.

As a result, you only have to:

- Choose either the Update or Input processing mode. Update allows you to read and write; Input only lets you read your file.
- Specify that you are opening a RAM file with fixed length records and give the file's symbolic name.
- Choose a number of buffers for use in this processing run. If you want to use the same number you used at creation, default this choice.
- If you want them, specify code translation, data verification, and exclusive use of the file.

That's all there is to it. The system gets the rest of the information it needs from the Permanent File Specification. Table I-3-2 summarizes these steps.

Processing Your RAM File

Once you've created and opened your RAM file, you're ready to process the data in it. In addition to the basic read and write functions, INFOS RAM gives you several processing request modifiers which you can use to improve the efficiency of the basic functions. But, before we get to the modifiers, let's examine what the INFOS system does with a RAM file during the normal read and write sequences.

Read

When you want to *read* a record, you'll code the equivalent of "Read record number xxx." The system will then go through the following steps:

It determines whether the block containing record xxx is in a buffer.

- If it *is*, the system will:

Move it from the buffer into your data area.

- If it is *not*, the system will:

Find a buffer for the block (it may have to write the contents of a buffer to disk to do this).

Transfer the block containing the record from disk to the buffer.

Transfer record xxx from the buffer to your data area.

Table I-3-2. Steps in Opening an Existing RAM File

You Must Do These:	You May Do These:
Choose a processing mode <ul style="list-style-type: none"> ● Update or Input Specify that this is a RAM file with Fixed Length records Give the file's symbolic name	 (Specify a number of buffers) (Request SETX function) (Request code translation) (Request data verification)

Write

When you want to *write* a record, you will first place the record in your data area, then code the equivalent of "Write record number xxx." When you enter this command, the system will:

Determine whether the block that contains the record (or will contain it if it's a new record) is already in a buffer

- If it *is*, the system will:

Move the record from your data area onto the buffer.

- If it is *not*, the system will:

Find a buffer to hold the appropriate block.

Transfer the block to the buffer.

Move the record from your data area to the buffer.

Write the contents of the buffer to the file either when it needs to use the buffer for a subsequent request or as soon as possible, if you have specified Write Immediate.

This basic read and write sequence is illustrated in Figure I-3-1.

That's what the system does with your RAM file when you want to read or write. Note that the first three steps are the same whether you're reading or writing.

Now, all that waiting time for block transfers and buffer emptying can mount up if you've got a lot of data to process. Therefore, in order to reduce your I/O wait time as much as possible, INFOS RAM offers the following processing request modifiers.

Pre-read

If you ask for two or more buffers when processing your file, the RAM Pre-read function lets you specify the number of the record you will want when you have finished processing your current record. The system will then attempt to find a buffer for that next record. If one is empty, the system will bring the block containing the record into the buffer. If all the buffers are full, the system will empty the one whose contents you used the longest time ago. Then it will bring the appropriate block into that buffer. Thus, you won't have to wait for the system to empty a buffer or transfer the block if you use Pre-read.

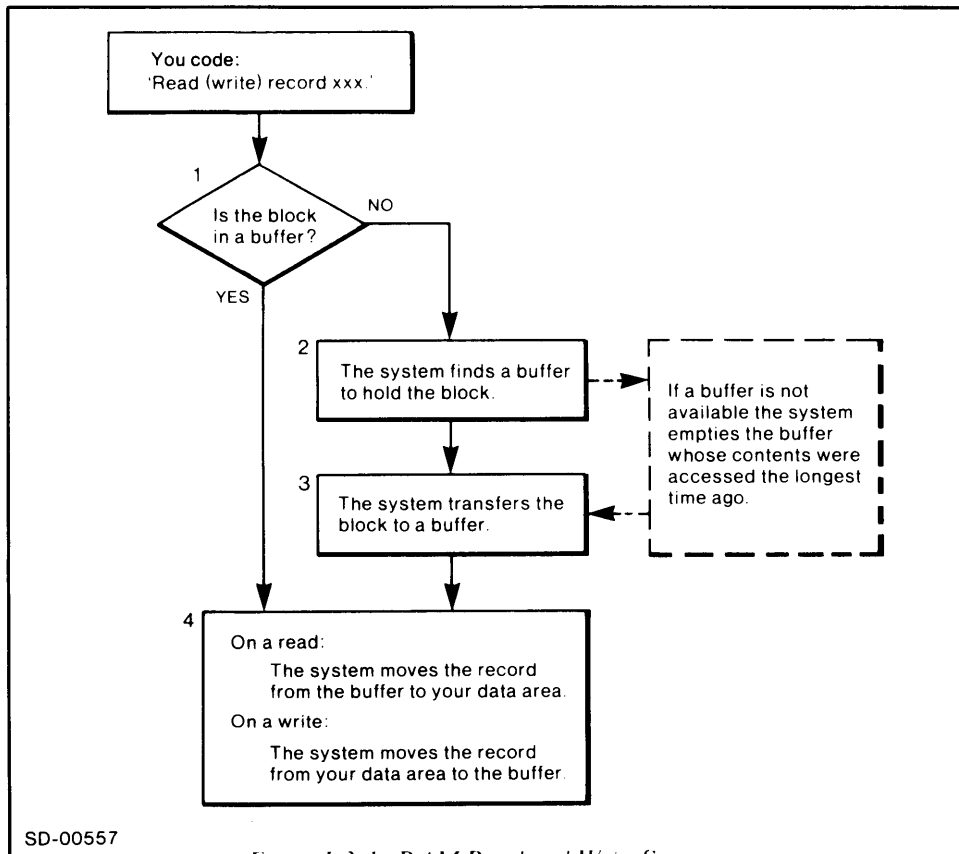


Figure I-3-1. RAM Read and Write Sequence

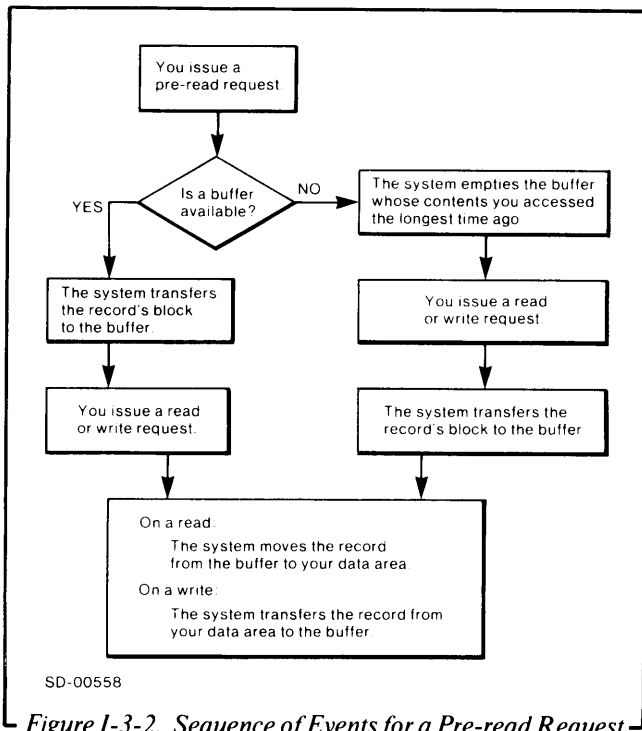


Figure I-3-2. Sequence of Events for a Pre-read Request

For example, say that you want to update records numbered 9, 36, 105, 202, and 203 in your RAM file, and you want to use two buffers. You can initially issue two Pre-read requests, using the record numbers 9 and 36, and the system will bring the blocks containing those records into the first and second buffers. Then, before you begin processing record number 36, you can request Pre-read for record number 105. The system will write the contents of the first buffer to the file in order to empty the buffer and read in record 105. In other words, every time you request Pre-read, the system empties a buffer while you are working on your current record and reads the next specified block into that buffer. This can greatly reduce your I/O wait time because you won't have to wait while the system clears a buffer for the next record.

Write Immediate

You can also modify a *write* command with a Write Immediate request. As you just saw, the system does not normally transfer the contents of a buffer to your file until it needs that buffer for a later processing request. However, if you request Write Immediate, the system writes the contents of the buffer to the file immediately. This is especially valuable when you use it in conjunction with Pre-read because it immediately clears a buffer for the system to use for the next record. Write Immediate thus gives you some control over the transfer process and can be useful in situations where you want to add or change a record as quickly as possible. It can also be useful if you're sharing a file with other users.

Read Inhibit

Read Inhibit can save you time when you are writing records into a block which you know is empty (for example, when you initially load your file). This request tells the system to assign a buffer for that empty block, but *not* to transfer the block to the buffer when you issue your write command. So instead of moving the block back and forth, the system will just move the newly-written record from your data area to the buffer. Then, when it needs the buffer, it will write the whole block to the file. In other words, Read Inhibit clears a buffer for the record, not for its block. This saves you more I/O wait time because the system doesn't have to find the proper block and bring it back into the buffer, then take it back to the file. Be careful with this modifier, however; if you use it to write a record to a block which already contains data, you will erase everything in that block except your newly-written record.

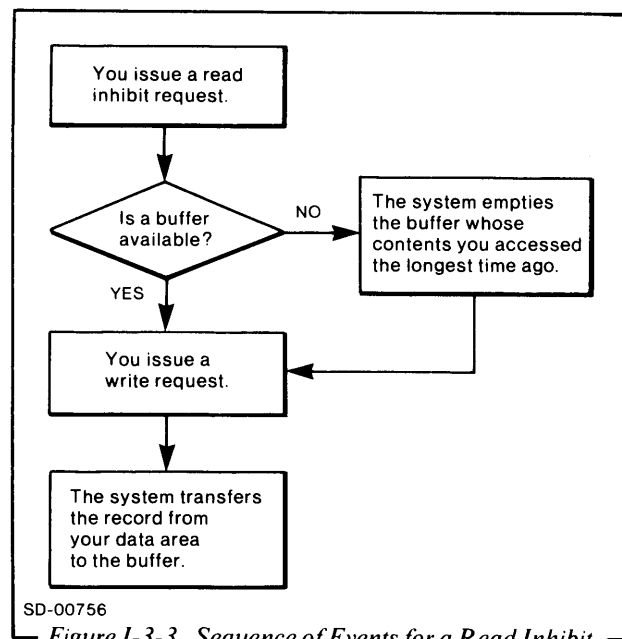


Figure I-3-3. Sequence of Events for a Read Inhibit Request

Lock and SETX

Lock and SETX are variations on the theme of exclusive use. The Lock modifier gives you exclusive access to the block containing the record you're processing. However, you must be careful to unlock each record you lock because no one can use the record until you close the file. The SETX command operates similarly, but gives you exclusive use of your *file*. And, as with Lock, no other user can open your file until you issue a Release Exclusive Use (RELX) request, or until you close the file. You can use both Lock and SETX with all read and write requests.

Hold

If you're working in a multiuser environment, you may want to use the Hold feature in conjunction with your read and write requests. If a record you want to access is in use or locked by someone else, you can issue a Hold request. Then the system will hold your processing request in a queue and give you the record when it becomes available.

FEOV

A Force End of Volume (FEOV) request allows you to close a volume of a multivolume file. This request tells the system to write all the records you've modified to the file, then close it. It can be useful if, for some reason, you cannot or don't want to close your entire file and you need to move a single volume of a multivolume file to another disk drive or processing unit. The system will automatically reopen a volume closed with FEOV if you want to access it again.

End of Chapter

Chapter 4

Indexed Sequential Access Method (ISAM) Files

General Concepts

Back in the dawn of real time, simple sequential and random access methods worked very well. They were, and still are, very useful for particular applications. However, as real-time marched on, people found that they needed on-line, remote-access databases and more complex data organizations. They also wanted to randomly access their data by something other than its relative record number. (A clutch plate is a clutch plate, not a 16216.) Unfortunately, neither sequential nor random file organizations could solve these problems. Thus ISAM, and the notion of *keyed access*, was born.

Now you can access a record by a key, rather than by its physical or logical position. A key is a shorthand way of telling the system which record you want. It can be any piece of data within a record, or it can be an element external to the record. For example, you can use employees' names or Social Security numbers as keys

to a file of personnel records, or you can use data which isn't part of the records (e.g., telephone extension, department name, shoe size).

In the INFOS system, the keys are in a separate file from the data records. That is, an INFOS ISAM file is really *two* files: one for the keys (called the index), and one for the data records (called the database).

These two files can reside on the same disk or each can have its own disk, or each can reside on many disks. In any case, it's not necessary that each disk you use have the same physical characteristics as every other. For example, you may want the index on a fast, fixed-head disk and the database on a slower, moving-head disk. Naturally, the number of disks you use depends on how much storage space you need for the index and database.

Let's look more closely at each part of an ISAM file.

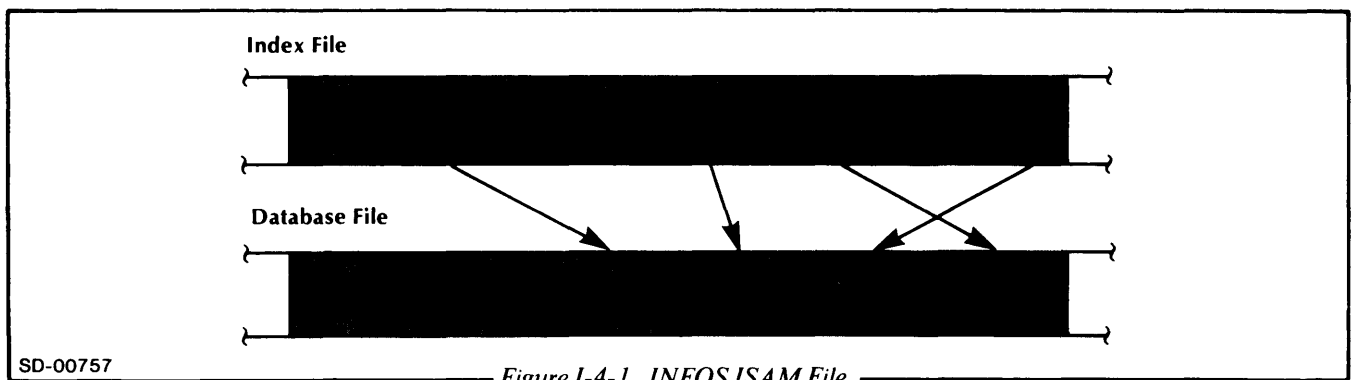


Figure I-4-1. INFOS ISAM File

The Database

The database section of an ISAM file is a simple, random-access file. You can write records to the file in any order and process them as you need them. Furthermore, in the INFOS system's ISAM, you can use variable length records, which make record construction easier for you and more efficient in terms of disk storage space. Your data records can be as long as 6136 characters, or as short as one character, and you can enlarge or reduce your records at any time with a simple rewrite operation. Moreover, after you specify the type of disk you want to store the records on, you never need to worry about where each one is located. The system automatically keeps track of each record and puts new records into whatever space is available on the appropriate disk. Nevertheless, the database part of an ISAM file is simply a random collection of up to four and a quarter billion data records.

The Index File

Likewise, the index file is quite simple: it is merely an ordered file of keys to the database records. As you write each database record, you supply a key to that record. The system automatically puts the keys in order in the index file and keeps track of the location of that key's data record. The most important thing about keys, however, is that *you* supply them so you know what they mean. Your keys need have no meaning to the system; the system merely stores them in your index file and uses them to access your data. To retrieve a record, you just supply the key and the system gives you the record. Thus, you can process your data records randomly by supplying individual keys, or you can process them sequentially according to the order of the keys in the index.

To make your keys even more useful, you can make them up from any combination of ASCII characters -- letters, numbers, special characters such as @, #, \$, or binary strings -- and each key can be up to 255 characters long. Furthermore, each key can be exactly as long as you require, and no two need to be the same length. Of course, if you have a file of, say, personnel records which are keyed by Social Security numbers, all the keys can (and will) be the same length.

You can also construct keys which have no corresponding database records. For example, suppose you have a data-gathering application which will extend over several weeks. You can build an ISAM file with a key for each piece of data you expect to collect, even

though some of them won't be available until later in the project. You simply write the key to the file without a data record, and, later, access that key and perform a rewrite to add that record to the file. This is known as database record suppression, and we'll discuss it further in the processing section of this chapter.

In another application, you may want to use one key to access a group of data records. This concept is called *duplicate keys* and it can come in very handy. For instance, if you maintain a nationwide mailing list, you may want to key records by city name. Your list may contain hundreds of names and addresses for the city of Boston, and you may want to key them all by that word. You would simply specify that you are using duplicate keys when you are writing your index file. Consequently, the system will automatically assign the first BOSTON key an occurrence number of zero. Then it will give the first duplicate key an occurrence number of one (i.e., BOSTON 1). The second duplicate will become Boston 2, and so forth.

The system will continue to assign incremental occurrence numbers each time you write a key called BOSTON. Then, when you want to do a mailing, simply request a read for the duplicate key BOSTON 0. After that, you can read all the Boston records sequentially. However, note that if you use a keyed access without indicating an occurrence number, you'll only get the first occurrence of that key. That is, if you want to read BOSTON 26, you must code the equivalent of "READ BOSTON 26". If you just say "READ BOSTON", you'll only get BOSTON 0.

This is the bottom line for index files: no matter how many keys you have in your index, you can access any of them very rapidly. This gives you the benefits of random access, in addition to the sequential processing capabilities provided by the logical order of the keys in the index.

Concurrent Access

The INFOS system allows many users to open an ISAM data file simultaneously (either through multitasking or use of the foreground/background, or both), because the system automatically interleaves operations aimed at the same file. Furthermore, to prevent conflicts when you're rewriting and/or deleting records, the INFOS system provides Lock and Unlock capabilities so that you may reserve and free records and their associated keys while you're processing them.

Space Management

You can specify the Space Management feature for either the index or the database file (or both), regardless of whether they are on the same or different devices. Space Management tells the system to reuse the space a deleted record used to occupy. Thus your files can grow or shrink according to your needs. For example, suppose you have an inventory file with a relatively stable number of data records which grow and/or shrink because of daily transactions. If you choose Space Management, the system will automatically reuse the spaces originally occupied by database records which it has relocated because of a change in their size. Thus, you'll never have to dump, sort, reallocate, or reload your file to recover lost or unused space.

In summary, the INFOS system's ISAM lets you organize your data randomly, but access it either randomly or sequentially. And since both your data records and your keys are of variable lengths, your entire file becomes more convenient and efficient, especially if you use Space Management.

Now that you're familiar with the general set-up of the INFOS system's ISAM, let's look at how you can create, open, and process your ISAM file.

How to Create an ISAM File

There are two ways to create an ISAM file: you can use the ICREATE interactive utility (as described in the *INFOS Utilities User's Manual*), or you can write a program to create the file. Regardless of which method you use, however, remember that you're actually setting up a *pair* of files: one for your keys (i.e., the index), and one for your database. Therefore, you must define each file separately and, in addition, define each volume for each file separately.

Creating the Index File

To create an Index file, you must first specify that you are defining an Index file with Variable Length records. Then you must request the Create Update processing mode and specify that you are using *one* index level. Next, you have to include a Volume Table Pointer. This tells the system the starting point in the computer's memory of the Volume Definition Table. Finally, you must specify the initial node size. To explain what this means, we must digress slightly.

In Chapter One, we said that the system transfers data in blocks and that, for disk files, these blocks are 512 characters long. In an ISAM file, the system transfers data in *pages*, which can be up to twelve blocks long (i.e., 6144 characters).

In an Index file, pages are made up of *nodes*, which, in turn, are made up of at least enough file space to hold three keys. The system automatically uses these nodes to build your Index file. Or, from another angle, each node contains at least enough space to hold three keys and each page may contain from one to 255 nodes. Naturally, the number of nodes in any given page depends on the size of that page, and the size of the nodes (determined by the number of characters in each key).

Now, you will specify the initial node size in bytes. However, the larger you make your nodes, the more efficient your index processing will be because of the way that the system accesses ISAM files. The only maximum restriction on initial node size is that it cannot be greater than your page size minus six bytes. Further information on nodes and how the system uses them is in Appendix B.

Once you have specified all the above information, the system will save it as part of the Permanent File Specification. Then, in addition to this Permanent File information, you can also specify any or all of the following options:

- *Page size.* You can choose any size page you want (up to the size you specified at system generation) or you can default this choice and the system will give you pages which are 512 characters long.
- *Number of Buffers.* This tells the system how many buffers to use for your I/O processing. If you default this choice, the system will give you two buffers.
- *Space Management.* (See the section on Space Management earlier in this chapter.)
- *Number of Volume Table Entries.* When you're working with a multi-volume index file, you must indicate how many volumes the file contains. If you default this choice, the system will assume that there is only one.
- *Maximum Key Length.* Your keys can range in length from one to 255 characters, and no two keys need be the same length. However, tell the system the expected length of your longest key because it requires that your nodes hold at least three keys, and

Licensed Material - Property of Data General Corporation

defaulting this choice will give you keys of a size such that three will fit in a node. This may be inconsistent with your desired key size, and the system will tell you so in an error message.

- *Database File Definition Pointer.* If you're building a new index file for an existing database, you can modify the number of buffers recorded in the Permanent File Specification for that database. If you do this, however, you must tell the system where, in the computer's memory, the new specifications start.
- *Disable Hierarchical Replacement.* If you are processing your ISAM file randomly with *many* buffers (i.e., more than three or four), use this option. If you are using very *few* buffers, you should use the Hierarchical Replacement feature. It will save you time and make your I/O transfers more efficient because it alters some of the internal steps which the system performs to buffer the index pages.
- *Read-After-Write Verification.* This will tell the system to automatically verify each block of data it writes to your file.

Table I-4-1 summarizes all the steps for defining an Index file.

Table I-4-1. Steps in Defining Your ISAM Index File

You Must Specify These:	You May Specify These:
This is an index file with Variable Length records	(Space Management)
Create Update processing mode	(Read-After-Write Verification)
One index level	
Starting address of Volume Table Pointer	
Initial Node Size	
Page Size	(default = 512 characters)
Number of Buffers	(default = 2 buffers)
Number of Volume Table entries	(default = 1 volume)
Maximum Key Length	(default = size which allows 3 keys/node)
	(Database file definition pointer)
	(Disable Hierarchical Replacement)

Licensed Material - Property of Data General Corporation

Defining Volumes of Your Index File

In addition to defining your entire index file, you also have to define each volume which contains a part of the index. This is considerably easier than defining the file because you only have to give the memory address of the volume name and specify an ASCII character for the system to use as padding for partially filled pages. In addition to these two mandatory specifications you also have the following options:

- *Contiguous Allocation.* If you want the system to allocate space on this volume in one contiguous piece (rather than over many random ones), specify this option and the volume size. Note, furthermore, that you can increase your processing speed by allocating your first volume contiguously to hold your present keys. Then you can use Random Allocation for the remaining (or future) volumes to handle file expansion.
- *Volume Size.* If you choose Contiguous Allocation, you must indicate the number of blocks on this volume that you want the system to immediately allocate for keys. If you choose Random Allocation, you can either specify a maximum volume size or let the system assign a size of 65,535 blocks of 512 characters each. It will then use those blocks as it needs them.

Note that it is *your* responsibility to ensure that you have sufficient disk space for each volume. If you select Random Allocation, we recommend that you specify a volume size if you want any file overflow to go to another volume. If you select Contiguous Allocation, we recommend that you also select Disable File Initialization because initialization is not required and takes about twenty minutes for a maximum-size volume.

- *Disable File Initialization.* For a contiguously allocated file, this option tells the system *not* to automatically erase all previous information in your allocated blocks.
- *Timeout Interval.* This options allows you to establish your own timeout intervals if you wish. Normally, the system-assigned intervals are three seconds for fixed-head disks and five seconds for moving-head disks. The system will return an error if it cannot complete a data transfer within these intervals.

That's all there is to defining each volume of your index file. Table I-4-2 summarizes these steps.

Table I-4-2. Steps in Defining Each Volume of Your ISAM Index File

You Must Do These:	You May Do These:
Specify address of the Volume Name Specify a pad character Choose Random or Contiguous Allocation Specify Volume size	(default = Random) (If contiguous, specify Disable File Initialization) (default = 65,535 blocks) (Specify a timeout interval)

Defining Your Database File

Half of your ISAM file now exists. However, defining the database file and its volume(s) is very similar to defining the index. You only need to know a few new things. First you must tell the system that this is a randomly accessed file with variable length records. And, second, you *cannot* use these index-related options:

- Disable Hierarchical Replacement
- Maximum Key Length
- Initial Node Size
- Database File Definition Pointer.

Other than this, you'll follow the same steps you follow when you create an Index file. See Table I-4-3 for a summary.

Defining Volumes of Your Database File

As usual, you also have to define each volume of your database file. However, these steps are exactly the same as those for the index file, so we'll just summarize them in Table I-4-4.

Table I-4-3. Steps in Defining Your ISAM Database File

You Must Specify These:	You May Specify These:
This is a RAM file with Variable Length records	(Space Management)
Create Update processing mode	(Read-After-Write Verification)
Starting address of the Volume Table pointer	
Number of Volume Table entries	(default = 1 volume)
Block size	(default = 512 characters)
Numbers of Buffers	(default = 1 buffer)

Table I-4-4. Steps in Defining Each Volume of an ISAM Database File

You Must Specify These:	You May Specify These:
The address of the Volume Name	
Pad character	
Contiguous or Random Allocation	(default = Random)
	(If Contiguous, Disable File Initialization)
Volume size	(default = 65,535 blocks)
	(Timeout interval)

How to Open an Existing ISAM File

You've probably already figured out that opening an existing ISAM file is another two-part process (Index and Database), but opening is easier than creating and opening.

Opening Your Index File

There's nothing new here. You follow exactly the same procedures to open your index as you did to create it, except that you don't have to specify an initial node size or a maximum key length. (Those values are already in the Permanent File Specification.) Table I-4-5 should look familiar by now.

Opening Your Existing Database File

If you do not specify a starting address for the database file definition when you open your index, the system will automatically use all the information in the Permanent File Specification to open the file.

If, however, you do specify the database file definition address, then you will have to specify:

- that the file has Variable length records;
- that you will be using the Update processing mode;
- the memory address of your Volume Table;
- the name of your first volume; and
- that this is a randomly accessed file.

You also have the option of specifying a different number of buffers than you recorded in the Permanent File Specification.

Table I-4-5. Steps in Opening an Existing ISAM Index File

You Must Specify These:	You May Specify These:
<p>You're opening an Index file with Variable Length records</p> <p>Update processing mode</p> <p>One index level</p> <p>Starting address of the Volume Table pointer</p> <p>Name of the first volume</p> <p>Number of Buffers</p>	<p>(Read-After-Write Verification)</p> <p>(default = 1 level)</p> <p>(default = number specified in PFS)</p> <p>(Disable Hierarchical Replacement)</p> <p>(Database file definition present)</p> <p>(Database file definition pointer)</p>

Processing Your ISAM File (Background)

Before we tell you about all the different things you can do to your ISAM files, it's important that you understand the two methods of accessing these files. We've already mentioned *keyed* access in the earlier part of this chapter, so now let's look at *relative* access (also called relative position processing).

Whenever you open an INFOS ISAM file, the system automatically sets a *current position*. Current position is like a "home base", on which the system stands as it reaches out to access your file. Initially the system sets this position just above the first key, as shown in Figure I-4-2.

Each time you access a key (and its associated database record), you can set a new current position at that key. Thus, by resetting your current position with each processing operation, you can read your file sequentially using relative motion. Since all positioning in your file thereafter will be relative to your current position, we call this relative access (relative position processing). You can move in any of six directions from your current position:

- Down, which moves you from the initial current position to a position just in front of the first key, and which sends you a warning that you haven't actually accessed a key;
- Down and forward, which gives you access to the first key from the initial current position;
- Forward, which lets you access the next key in the file;
- Backward, which lets you access the key before the current one;
- Upward, which returns you to the system-set initial current position, and, like down, sends you a warning;

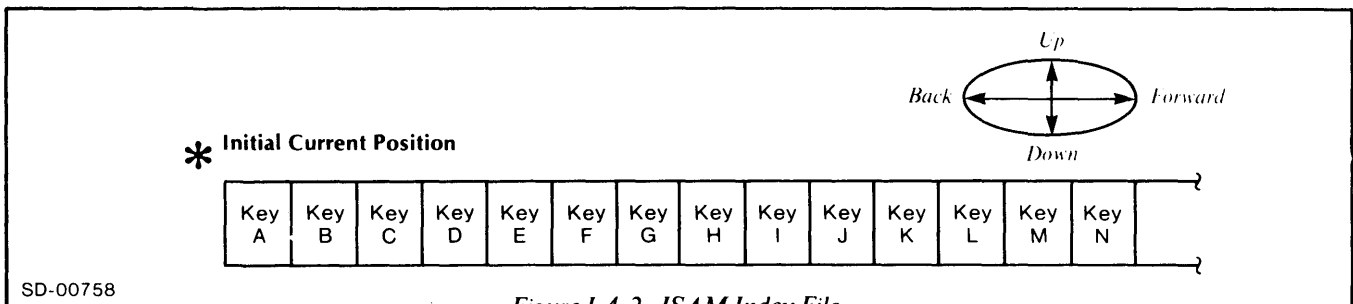
Licensed Material - Property of Data General Corporation

- Static, which lets you access the same key without changing position.

Relative access depends upon your setting or resetting a current position. All motion within your file (except static) will be away from the current position you have set, in whatever direction you indicate. In other words, you need only specify a direction of motion when you are using relative access; you *don't* need to specify a key.

For example, say that you want to rewrite several records, then access your entire file sequentially. Using relative access, you would do the following:

1. *Request a read operation with down and forward relative motion.* This will give you access to the first key and its data record. You should also *set current position* when you request the read, so that you will have a "home base" on the key you accessed rather than on the initial current position. This will let you use STATIC motion for your rewrite, rather than another positioning movement.
2. *Request a rewrite operation with static relative motion.* This is the fastest way to rewrite your file, since the system doesn't have any positioning to do before it processes your request.
3. *Request forward reads with set current position and static rewrites* for each record you want to update until you reach the end of the index.
4. *Request Upward motion*, when you have finished rewriting the individual records you wanted, to reposition yourself to the initial current position.
5. *Request a read operation with down and forward relative motion* to reaccess the first record, then process your entire file sequentially by requesting reads with forward motion; remember also to set a new current position.



SD-00758

Figure I-4-2. ISAM Index File

Licensed Material - Property of Data General Corporation

It's as easy to process a file randomly with keyed access as it is to process one sequentially with relative access. You simply give the system the key for the record you want and you've got it. Furthermore, the concept of current position also applies to keyed access processing and can save you a lot of time when you're rewriting. By setting a current position on the record you want to rewrite, you can read the record before you rewrite it, then use static relative motion to rewrite. This speeds things up because static motion is always faster than the other movements.

For instance, you could use keyed access to perform the read function in our last example. In this case, you would simply issue a read request using the key for each record you want, and set current position at that record. Then you would rewrite with static relative motion as before.

As this example illustrates, you can use keyed or relative access at any time in an ISAM file. However, you cannot combine them in a single operation; that is, you cannot request forward motion to key *x*. You can request a processing operation for key *x*, or you can request an operation with forward motion, but you can't combine the two.

Now that you've got the necessary background information, let's see what you can do with your ISAM file.

Processing Your ISAM File (Operations)

For clarity, we'll break down the ISAM processing operations into three categories:

- Processing Functions - read, write, rewrite, delete, and reinstate;
- Utility Functions - retrieve status, retrieve key, and retrieve high key;
- Auxiliary Features - lock/unlock and suppress database.

The processing functions can modify either the file's structure or its content, but the Utility Functions never modify structure or content. The Auxiliary Features are just that - extra goodies.

Processing Functions

Read

The ISAM read function is the one you'll use to retrieve data from your file via either keyed or relative access. You can do a relative read in any of the six directions described above; the system's motion is relative to your last established current position. For a keyed read, the system gives you the record you request. If you request a keyed read for a duplicate key, you must specify the occurrence number or you will get only the first occurrence of the key.

Write

Unlike read, you can write to an ISAM file only by keyed access. That is, with each write request, you must supply a key which will become a new index entry. If you try to write a key which already exists, the system will send you an error message unless you requested duplicate keys. When you're writing duplicate keys, the system will automatically assign each one an occurrence number; thus all the keys in your index will remain unique.

Rewrite

This function lets you update or change any existing database record (but not its key). It works with both keyed and relative access. Rewriting will not change the structure of the index, but you can change the length of a database record.

Your positioning options under rewrite are the same as those for read; it's just that the data flows in the opposite direction (i.e., to the file, not from it).

Delete

This function gives you the ability to delete records from your Index and Database files and comes in two flavors: physical and logical deletion. Physical deletion uses keyed access to erase the record and/or its key from your file. When you use physical deletion in conjunction with the Space Management feature, you eliminate the need to reorganize your file. The system will automatically reallocate the space left by your deleted records.

When you use logical deletion, records and their index entries are *not* erased, but merely marked as logically deleted. When you access a logically deleted record, the system will let you know that the record has been so marked, but will still give you access to it. If you mark the Index entry, it is called *local* logical deletion; if you mark the database record, it's a *global* logical deletion. You may mark both local and global logical deletion for any record.

For example, if you had a personnel file, you might want to logically delete the records of those employees who were on a leave-of-absence; you're not paying them, but they may return to work. On the other hand, you would physically delete the records of all employees when they resign, are fired, or die. Naturally, if you try to access an element which has been physically deleted, you'll get an error message.

Reinstate

This function lets you remove the marks from the records which you have previously logically deleted; in other words, it reinstates those records to your files. You can reinstate an index entry or a database record, or both, and you can do it via keyed or relative access.

Delete Subindex

This function allows you to physically delete all entries from your Index and database records. You may find this function useful when you no longer need an ISAM file, but you want to retain the file space allocated for it.

Utility Functions

Retrieve Status

The Retrieve Status function lets you determine, through keyed or relative access, the following information:

- the length of a database record and whether or not it has been logically deleted; and
- the length of that record's key, whether it is a duplicate, and whether or not it has been logically deleted.

Licensed Material - Property of Data General Corporation

Retrieve Key

You can use this function to retrieve any key and its length through keyed or relative access. Normally, the system will just give you the data record you request, but you can use this command to find out what the key for a record is. This can be useful when you are processing your file via relative access, moving from record to record, and you want to know the key (or the key's length) for a particular record.

Retrieve High Key

Use of this command will tell you what the ending key in your index is. You can Retrieve High Key through keyed or relative access.

Auxiliary Features

Lock/Unlock

Sometimes you may want to have exclusive access to a key or a database record. For these times, INFOS ISAM offers you an optional protective feature: you can locally lock any key, or you can globally lock the database record. Once you lock an element, no one can access it unless they unlock it. For example, if you are rewriting a data record, you can request a read which will set current position on that record and lock it. Then, you can rewrite it with a static motion and unlock it at the same time.

If you're working in a multiuser environment and you access an element which someone else has locked, you'll get the error message DATA RECORD LOCKED. If you really need that record, you can reaccess it by explicitly stating in your operation request that you want it unlocked. Then the system will remove the lock and perform your operation. You can use the lock/unlock feature with any of the ISAM processing functions described above except REINSTATE.

Suppress Database

In every processing operation, you have the option of not retrieving the data record associated with the key you access. For example, if you only want to establish a current position in the index, just issue a processing request that suppresses the retrieval of the database and sets current position. Or, if you want to create an index entry with no corresponding data record (for, say, each week of a long-term data gathering survey), you can merely issue a write command with database suppression for the key you want to write.

End of Chapter

Chapter 5

Data Base Access Method (DBAM) Files

Preliminary Note: Because DBAM is an extension of ISAM, read Chapter 4 to make sure you understand ISAM before you start this chapter.

General Concepts

The INFOS system's Data Base Access Method (DBAM) is to its ISAM as Superman is to Clark Kent: you get all the features you had in ISAM, but DBAM extends and strengthens those features to make your data processing easier, more versatile, and more efficient. These extended features include:

- Subindexing within a single index file
- Partial records
- Multiple indexes for a single database file
- Linked subindexes
- Temporary indexes
- Automatic key compression
- Optimized data distribution
- Approximate and generic search keys

Despite all these feature extensions, however, the basic structure of a DBAM file is still the same as that of an ISAM file. That is, you will still build two files: one for your index and one (and *only one*) for your

database. Each of these files can reside on one or more disks, or both can reside on the same disk. Your DBAM data records need have no relationship to one another, yet the file structure lets you build your file so that you can access it easily and logically.

DBAM is designed to be convenient. For example, you can access a single data record through many sets of keys in one index, and through many sets of keys in an entirely different index. To understand fully why DBAM is so useful and versatile, let's examine each of its unique features.

Subindexing

You can access a data record through many different paths because the INFOS system lets you build subindexes within your index structure. Often you may want to divide a large collection of information into smaller, more manageable units. For instance, recall the mailing list example we used in the last chapter. In an ISAM file, the keys for all the data records are in a single, or main, index. In our example, we used city names as keys. But, if you wanted to do a mailing to the northeast region of the country, you had to know the names of all the cities in that region. DBAM's subindexes help you avoid this problem.

In DBAM, the main index does not usually contain keys to data records; rather, it contains keys to other keys. In our example, the main DBAM index will contain one index entry for each region of the country. Each of these keys will then point to a subindex as shown in Figure I-5-1.

If you divided the country into nine regions, then the main index will contain nine entries. Each of these will have a subindex defined under it. In other words, level one in the illustration contains nine subindexes.

There are two things to remember about a DBAM index structure. First, each unique division of the index is called a subindex. The general corollary to this is that

you can define one subindex for each entry in any other subindex and the maximum number of subindexes in an index is limited only by the amount of available physical storage space, not by the system. Second, counting the topmost (i.e., the main) index, you can define a maximum of 256 index levels.

Index levels are numbered consecutively from 0 at the top to 255 at the bottom. Level 0 is the main index and it has the same properties as an ISAM index, except that you can define subindexes under it. So, you have the main index at level 0, each of whose entries points to a subindex at level one.

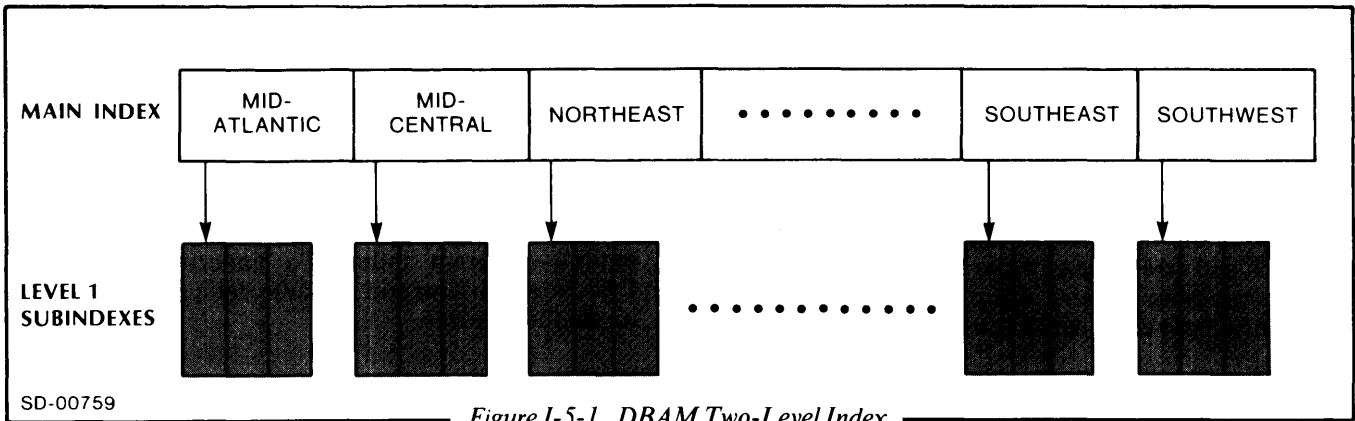


Figure I-5-1. DBAM Two-Level Index

Licensed Material - Property of Data General Corporation

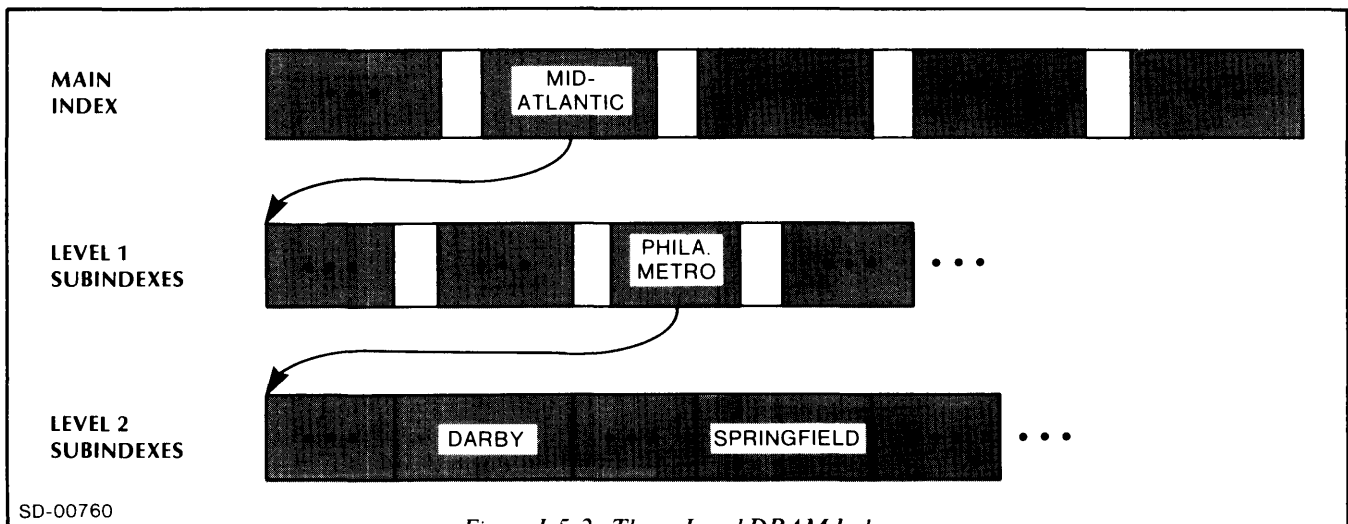
For an example, let's say that you want a highly selective mailing list. Instead of using city names as keys in the level one subindexes, let's use metropolitan regions. This will give you an even finer division of the data records. For instance, in the subindex for the Midatlantic region, you will have index entries such as New York Metro, Rochester Metro, Philadelphia Metro, and Baltimore Metro, as shown in Figure I-5-2.

Each of these keys then points to a subindex at level two. The keys at level two could be city names, which you could then subindex by zip code, and so forth. As you can see, you can construct as many levels of subindexes as you need for your application.

Partial Records

Now let's suppose that your mailing list is part of a larger application which maintains customer records for a large mail order catalog business. For the mailing list

portion of this application, INFOS DBAM will allow you to carry the customer names and addresses in the *partial record* section of the index entries. That is, each entry you create in a subindex can contain a fixed length partial record whose length you can choose, up to 255 characters. All the partial records in one subindex are fixed-length (i.e., they occupy the same amount of space), but you can select this feature for only specific subindexes, and thus avoid wasting too much space. So, if you set up partial records to hold frequently accessed data, you can save a lot of I/O time because you won't ever have to access your database file to get that data. However, note that you can access the data in the partial records only through their associated keys. Also, if the data in your partial records duplicates any database record information, you are responsible for updating both the partial record and the database - the system will not do it for you.



SD-00760

Figure I-5-2. Three-Level DBAM Index

Multiple Indexes and File Inversions

To maintain the customer records for the catalog order business we just mentioned, you can create a whole new index structure (with or without subindexes) for your existing database. That is, you won't have to duplicate any existing data records; a new index structure will give you a different set of paths to get to those records.

The technique for creating a new index is called *File Inversion*. To invert a file, you must first access a database record via an already established key or set of keys. Then you do a write inverted operation to your new index and the system will automatically create a new index entry there. Thus, you will have linked two unique index entries - one in each index - to the same database record.

Figure I-5-3 illustrates a single database with two unique indexes.

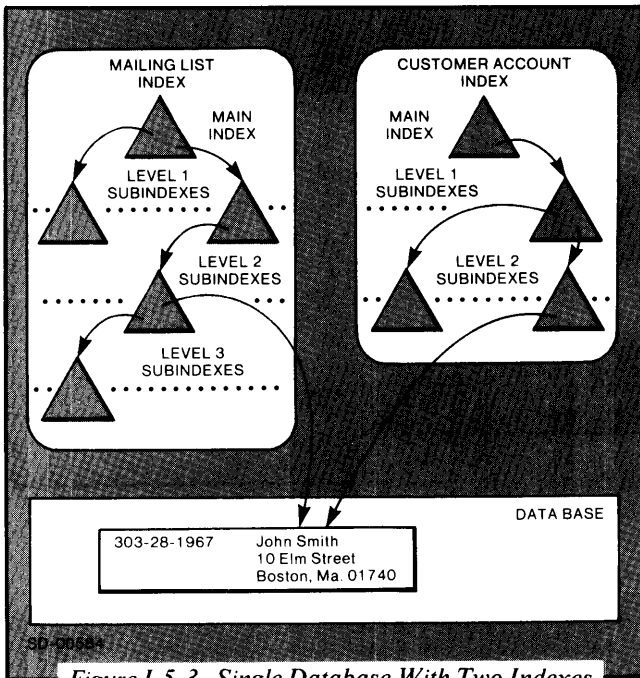


Figure I-5-3. Single Database With Two Indexes

Linked Subindexes

You can also link subindexes in any INFOS DBAM index to provide local data sharing. Just as inverting allows you to create a new index without duplicating your database, linked subindexes let you share data without reproducing a subindex structure. For example, you can build a single DBAM file for use by both your Personnel and Payroll departments. When the Personnel department wants to access a record, it will probably use the employee's name or social security number. Likewise, the Payroll department will most likely use a social security number or an employee number. To save storage space, you can use the Link Subindex feature to create just one social security number subindex which both departments can share, as illustrated in Figure I-5-4.

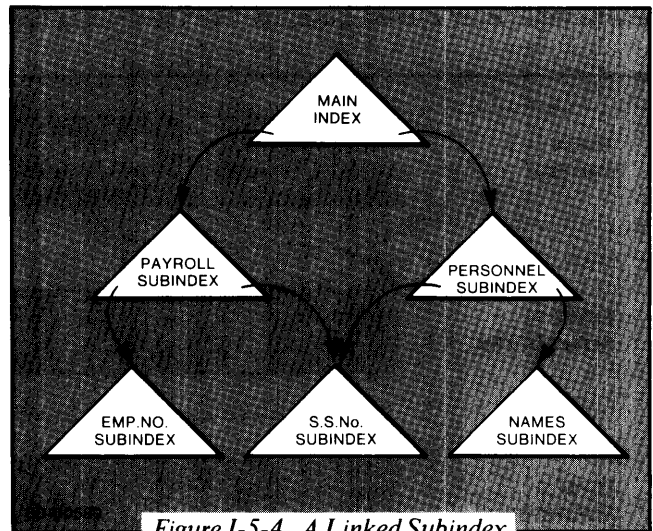


Figure I-5-4. A Linked Subindex

Licensed Material - Property of Data General Corporation

Automatic Key Compression

This DBAM feature can save you a lot of room in your index if you have several keys with identical leading characters. That is, if you have several keys whose first three or more characters are the same, you can have the system automatically store only the unique parts of those keys, along with one entry of the initial, identical part. For example, let's say that you have an inventory of parts whose numbers look like this: A101-PP-0710, where A101 is a department number, PP means product part, and 0710 is the actual part number. With the dashes, this is a twelve-character key. If department A101 is responsible for, say, 6500 product parts, you're going to have 6500 keys, each containing the same eight leading characters, A101-PP-. Why waste over a half million bytes (1000 sectors) of your index space with all that duplication?

When you define the subindex to contain the part number keys for department A101, simply specify automatic key compression. This is what will happen: when you create the first index entry, the system will use the entire key - say, A101-PP-0001. But when you create the second entry and you specify the key A101-PP-0067, only the unique part of the key (67) becomes part of that index entry. Thus, in this instance, you will save eight characters per key, and, in a subindex with 6500 keys, you'll save almost 52,000 character positions in the index.

There are three points to remember about key compression:

- Specify the *entire* key for each index entry you create; the system performs the compression automatically.
- Specify the entire key when you subsequently access that index. For example, code the equivalent of "Read the record whose key is A101-PP-2122", rather than "Read the record whose key is 2122".
- Key compression will not alter your processing speed or efficiency.

Optimized Distribution

This is a handy time-saver for your index or database files, or both. Optimized distribution lets you store the database records or the nodes within your subindexes which you use most frequently on a disk (or disks) with the fastest access time. This technique is especially useful for the root nodes. (A root node of a subindex is like the door to that subindex; it's the first point of contact between a subindex and its next highest index level. Don't worry about this right now. You can find further details on root nodes in Appendix B if you want them.)

For instance, in our mailing list application, you could tell the system to store the subindex containing the customer name and address partial records on a fast, moving-head disk. Thus, in addition to not having to access the database records to produce a mailing list, you would reduce the time it takes to access the subindex itself. Also, if you service some of your customers more frequently than others, you can store their database records on a fast-access disk.

Note, however, that you don't have to keep your original distribution of data across the various speed disks. If you find that you don't access certain records as frequently as you used to, you can simply rewrite their records to a slower disk. Or, if small customers become big customers, you can rewrite their records onto a faster disk. In other words, you can use this feature to save time in accessing your most frequently used records.

Temporary Indexes

Occasionally you may want to create and use one or more indexes only for the duration of a processing run. These are called *temporary indexes*. For example, if you were preparing a special report, you might want to access each entry in our mailing list file, looking for customers who have spent, say, \$100, \$200, and \$300 in the past month. You could create three temporary indexes (one for each category) to store the information you find. There are actually two ways to do this: you could either create three new subindexes for your present main index, or you could create three new index files to contain your special report data. Generally, it's easier to create a subindex for your existing index file than to define an entirely new index file.

Whichever method you choose, however, the procedure will be the same after you have created the (sub)index. When you find a customer who satisfies the criterion for your report (i.e., one who has spent \$100, \$200, or \$300), you will write an index entry in the appropriate (sub)index. Then, after you have accessed all of your existing entries and finished making your temporary (sub)indexes, you can sequentially access those (sub)indexes to produce your special report.

There's just one other important point to keep in mind when you're using temporary indexes: deleting a temporary index does *not* delete any database records whose only access is through that index. However, you will not be able to access those database records unless you tie them to a permanent index entry. That is, if you do not tie a new data record to a permanent index entry, you won't be able to access it after you delete its

temporary (sub)index, but it will continue to occupy space in your file. Therefore, be careful with temporary indexes; don't waste valuable file space by creating inaccessible data records.

Those are most of the extended features that make DBAM unique. There are more, but we'll cover them when we get to the "Processing" section of this chapter. First, however, let's see how you create and open a DBAM file.

How to Create an INFOS DBAM File

Since the structure of a DBAM file is so similar to that of an ISAM file, the steps for creating one are similar too. As with ISAM, you'll build two files - one for the index and one for the database. The only difference between ISAM and DBAM at creation is in the number of steps involved. Therefore, in the creation steps that follow, we will explain only those features which differ from their ISAM counterparts, or which are unique to DBAM. If you are unsure of an unexplained point, refer back to the "Creation" section of Chapter Four.

Defining Your Index File

To begin, you have to specify that you are creating an index file with Variable Length records, and that you are using the Create Update processing mode. Following this, the other required specifications are:

- *Number of index levels:* At runtime, you must indicate the number of subindex levels you expect your index to have. If you aren't sure how many subindex levels you'll want, just specify a relatively large number (5 or 6 will probably do). This won't waste any space in the index, and, if you find that this number is insufficient, you can change it on a subsequent opening.

NOTE: If you specify only one index level at runtime, you will not be able to use *any* subindexes.

- *Volume Definition Table Pointer*
- *Minimum Node Size*
- *Database File Definition Pointer*
- *Number of Volume Table Entries*

Licensed Material - Property of Data General Corporation

- *Number of Buffers:* The default value here is two buffers, but you can choose more if you wish.
- *Page Size:* This is the same as block size in SAM and RAM, except that, because of the way that the system transfers data, you're limited to 12 disk sectors, not 512 characters. The minimum restriction here is that a page must be at least six characters larger than your largest expected node. If you default this choice, your pages will only be 512 characters long.

After you specify all of the above, you get to choose from these eleven index file definition options:

- Read-After-Write Verification
- Space Management
- Enable Hierarchical Replacement
- Maximum Key Length
- Optimized Record Distribution
- Key Compression
- Partial Record Length: All partial records in your index will be the same length, but you can choose that length, up to 255 character, to suit your needs.
- No Subindexes: If you don't want to use subindexes, you can specify so here and you'll save four bytes of disk space per key in your index.
- Temporary Index: If you want to use them, say so.
- Permanent Data Records: If you're creating a data record through a temporary index and you want to keep it after you delete the temporary index, specify it here.
- High Priority Node: If you're working with a complex index structure with many subindexes, you can use this option to keep in memory the root node of an important index or subindex. (See Appendix B for further explanation.)

Table I-5-1. Steps in Creating a DBAM Index File

You Must Specify These:	You May Specify These:
This is an index file with Variable length records	(Space management)
Create Update processing mode	(Read-After-Write verification)
Number of index levels	(default = 1 level)
Volume Definition Table pointer	
Number of Volume Table entries	(default = 1 volume)
Minimum node size	(default = 6 bytes less than block size)
Database file definition pointer	
Page size	(default = 512 characters)
Number of buffers	(default = 2 buffers)
	(No subindexes)
	(Disable hierarchical replacement)
	(Maximum key length)
	(Optimized record distribution)
	(Key compression)
	(Partial record length)
	(Temporary indexes)
	(Permanent data records)
	(High priority node)

Defining Each Volume of an Index File

As in ISAM, you must define each volume which is part of your file. There are only two required specifications here: the volume name pointer and the pad character. Then there are five options:

- Volume Size
- Contiguous or Random Allocation
- Device Timeout Interval

- Volume Merit Factor: This is related to Optimized Record Distribution. If you selected that option for your file, you must assign a merit factor to each volume in the file. Simply put, this means that you must tell the system the merit factor of the highest priority volume first; the other volumes must follow in order of decreasing merit factors. Also, when you write a record, you must specify a merit factor, thereby choosing which data volume will hold the record.
- Disable File Initialization

Table I-5-2. Steps in Defining Each Volume of a DBAM Index File

You Must Specify These:	You May Specify These:
Volume name pointer Pad character	(Contiguous allocation; default = random) (Volume size; default = 65,535 blocks) (Timeout interval) (Volume merit factor) (Disable file initialization)

Defining Your Database File

There's nothing new here. See Table I-5-3 for a summary of your options.

Table I-5-3. Steps in Defining a DBAM Database File

You Must Specify These:	You May Specify These:
Random access method Variable length records Create Update processing mode Volume table pointer Number of volumes Page size Number of Buffers	(Space management) (Read-After-Write verification) (default = 1 volume) (default = 512 characters) (default = 1 buffer) (Optimized Record Distribution)

Defining Each Volume of Your Database File

Again, there are no new concepts here. See Table I-5-4 for a summary.

Table I-5-4. Steps in Defining Each Volume of a DBAM Database File

You Must Specify These:	You May Specify These:
Volume name pointer Pad character Volume size Contiguous or random allocation	 (default = 65,535 blocks) (default = random) (Timeout interval) (Volume merit factor) (Disable file initialization)

How to Open an Existing DBAM File

There are very few differences between opening a DBAM file and opening an ISAM file, as the charts in Table I-5-5 and I-5-6 illustrate.

Table I-5-5. Steps in Opening an Existing DBAM Index File

You Must Specify These:	You May Specify These:
This is an index file with Variable length records Update mode Number of index levels Address of volume table pointer Number of buffers	(Read-After-Write verification) (default = permanent file specification) (default = permanent file specification) (Disable hierarchical replacement) (Database file definition present) (Database file definition pointer)

Table I-5-6. Steps in Opening an Existing DBAM Database File

You Must Specify These:	You May Specify These:
This is a RAM file with Variable length records Update mode Volume table address Name of first volume	(Different number of buffers from the permanent file specification)

Processing Your DBAM File

DBAM Access Methods - Keyed, Relative, and Combined

Like ISAM, DBAM lets you get to your data through keyed or relative access; unlike ISAM, however, you can also *combine* these two access techniques. When you use keyed access in DBAM, the system applies the key or keys you specify from the top of the index structure. That is, it applies the first key to the main index, the second key to the first appropriate subindex, the third key to the next subindex, etc. When you use relative access, your motion will be relative to your last established current position. (Current position in DBAM means the same as it did in ISAM.) When you combine keyed and relative access in a single operation, the system always performs the relative motion first; then it applies the key(s) from there.

That's the general idea, but let's examine each access technique a little more closely.

Keyed access in DBAM means that you give the INFOS system a key or a *set* of keys which will lead you to a subindex entry, which, in turn, may or may not

lead you to a database record. (If you're using partial records, for example, you may not *want* to see the entire data record.) Furthermore, you can access a single data record through more than one key or set of keys. Figure I-5-5 illustrates a few possible DBAM keyed access paths.

If you wanted to retrieve the "Product List", you would simply code the equivalent of "READ PRODUCTS". This is called a *single level key* because you're only using one index level to get to your desired data. To retrieve the "Parts List" record, however, you would have to code: "READ UNITS, PARTS". This is a *multilevel key*; you use it to go from the main index (on one level) to a subindex (on another level), and then to the record. Finally, note that you can retrieve the data record "EXTENDER A01" by coding the equivalent of either "READ COST, EXTENDER" or "READ UNITS, COMPONENTS, A01". Thus you can see that keyed access in DBAM is different from that in ISAM because, in DBAM, your keys can move you through multiple subindexes before you get to the record you want. In ISAM, remember, you went directly to the data record using only one key.

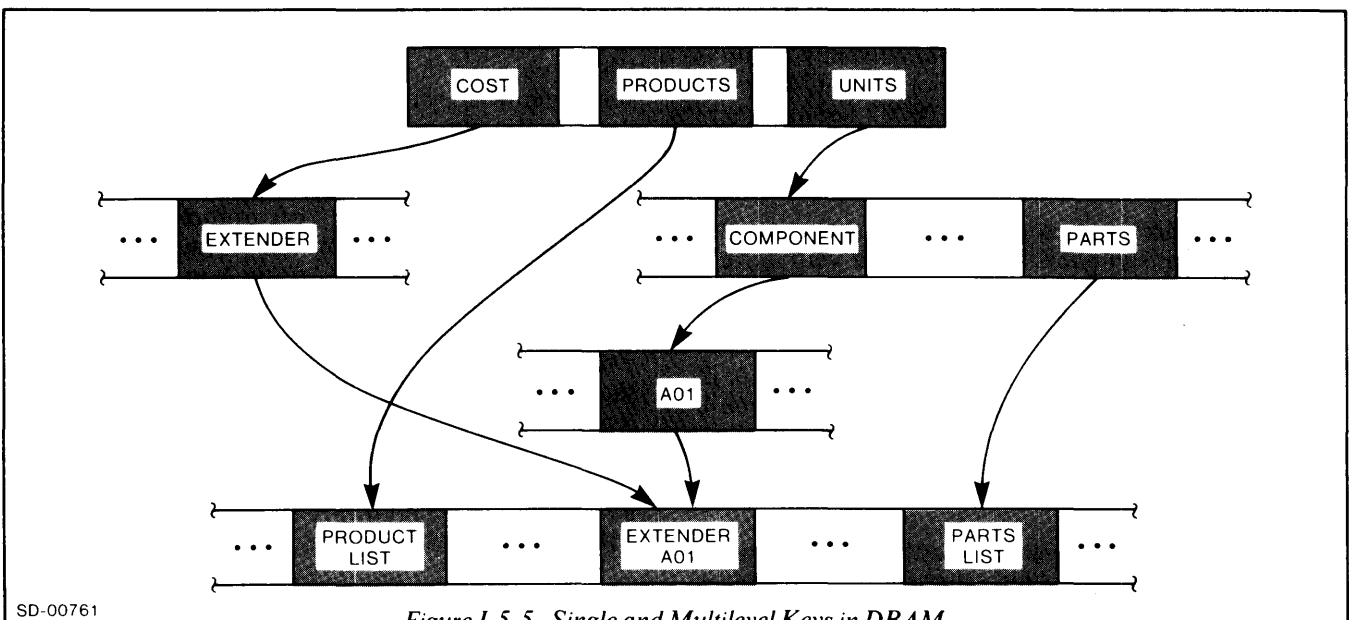


Figure I-5-5. Single and Multilevel Keys in DBAM

Relative access in DBAM is also somewhat different from that in ISAM. In DBAM, you're still moving in a direction relative to your last established current position, but you can move in *eight* different directions instead of just six. We'll use Figure I-5-6 to help illustrate these directions.

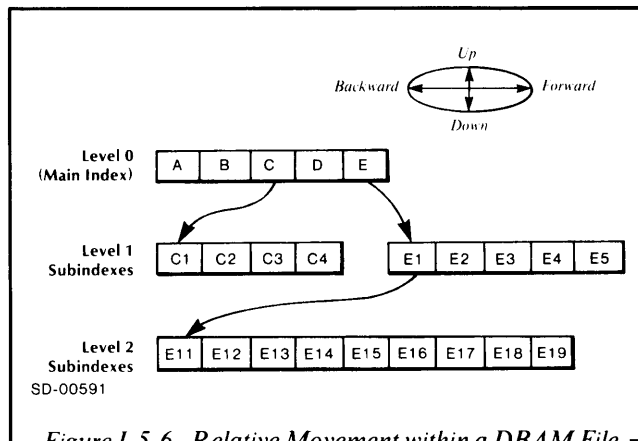


Figure I-5-6. Relative Movement within a DBAM File

Forward relative motion gives you the next entry in the subindex containing your last established current position. For example, if your current position is on entry C1, forward motion will give you C2. If you also request *set current position*, the system will set it on C2; otherwise, it will stay on C1. If your current position is on C4 and you request forward motion, you'll get the error message "END OF SUBINDEX" because there is no entry in the subindex beyond C4, and you cannot cross subindex boundaries using only forward motion.

Backward relative motion gives you the previous entry in your current subindex. For instance, if your current position is on entry C4 and you request backward motion, you'll get C3. Setting current position with that request will move it to C3; otherwise the system will return you to C4 when you're through with C3. If you're on C1 and you request backward motion, you'll get the error message "ILLEGAL REL MOTION" because there is no other entry in the subindex before C1.

Upward relative motion gives you the index entry on the next highest level to which your current subindex is linked. In other words, if you are on entry C4 and you request upward motion, you'll get entry C. Again, you have to set current position on C when you request movement if you don't want the system to return to C4. Naturally, if you are on C and you request upward motion, you'll get a "TOP LEVEL ERROR" message since there is no higher subindex.

Up and forward motion moves you up to the next highest level and forward to the next entry in that level. For instance, if your current position is on E14 and you request up and forward motion, you will access entry E2. (Don't forget to set current position again.) Now, if your current position is on E2 and you request up and forward, you'll get the message "END OF SUBINDEX" because, even though the system can move you up a level to E, it cannot go forward; there is no entry on that level beyond E. Naturally, if you are on E, you cannot move up and forward - there's no place to go.

Up and backward motion moves you up to the next highest level and backward to the entry just before your current subindex's source. For example, if you are on C3 and you request up and backward motion, you'll get B. (Set current position, if you want to.) Similar to up and forward, if you are on E14 and you request up and backward, you'll get an error message because the system can move up to E1, but it cannot move backwards from there.

Downward motion will move you to just in front of the first entry on the next subindex level. In other words, if you are on E, downward motion will put you just in front of E1. However, this command does not automatically set a new current position; you have to specify that if you want it. Downward motion is handy when you want to sequentially access each entry in a subindex; you just move down, set current position there, and read forward.

NOTE: You can only use this command for entries that have subindexes. You cannot, for example, move down from E3 because it has no subindexes. If you try to, you'll get the error message "SUBINDEX NOT DEFINED".

Down and forward relative motion moves you down to the next subindex and forward into the first entry in that subindex. For instance, if you are on entry C and you request down and forward motion, you'll get C1. If you are in the system-set, initial current position (i.e., just above the main index), down and forward motion will give you entry A.

Static motion accesses information stored in the index entry where you set your last current position. This information can be data and/or partial records. The only times you cannot use static motion is when you are in the initial current position or when you are just in front of the first entry in a subindex.

Licensed Material - Property of Data General Corporation

These are the only relative motion commands you can make; you'll get an error message if you try anything else. Therefore, if you are on D and you want to access E1, you have to code the equivalent of: "READ FORWARD, SET CURRENT POSITION" and then "READ DOWN AND FORWARD, SET CURRENT POSITION." This seems like a pain, and it is. That's why DBAM lets you combine relative and keyed access.

Combining Relative and Keyed Access

When you combine keyed and relative access in a single operation in DBAM, you effectively reduce your total access time because neither you nor the system have to do as much to access the index entry you want. That is, when you combine relative and keyed access, the relative motion takes you to the subindex that will be the starting point for the keyed access. Therefore the system doesn't have to start at the top of the index and apply the keys you supply, one at a time, until it gets to the entry you want.

Since you are using the relative motion part of a combined request just to establish a starting point for the keyed access, you'll only need three directions of relative motion: up, down, and static. In any single request, you can only explicitly specify one of these directions; the system always applies keyed access downward from the starting subindex. Static motion tells the system that the starting subindex is the one in which you last set current position. For example, in Figure I-5-7, a static motion request for entry C5 from a current position on C1 would tell the system to look for C5 in the subindex containing C1.

We'll use Figure I-5-7 to help illustrate combined keyed and relative access.

Upward motion tells the system to move up a level before starting keyed access. That is, if you were positioned on entry A41 and you coded the equivalent of "READ UP, A2, SET CURRENT POSITION", the system would move from subindex level 6 to subindex level 5, then search for entry A2. When it found A2, it would set a new current position on it. If you didn't want to set current position on A2, you would just code "READ UP, A2"; the system would return you to A41 when it finished accessing A2.

Downward motion, obviously, will start keyed access at the next lower subindex level. So, if you were on entry E5, you could code the equivalent of "READ DOWN, E25" and, if you wanted to, "SET CURRENT POSITION". This would move you from entry E5 to entry E25 and, if you specified it, set a new current position.

The fun comes in combining keyed and relative access to randomly retrieve any entry in a subindex. For example, in Figure I-5-7, how would you get from a current position on E1 to entry C3? Well, you would code the equivalent of "READ UP, C, C3, SET CURRENT POSITION". This positions you on entry C3. (If you just said "READ UP, C, C3" the system would return you to E1 after you accessed C3.)

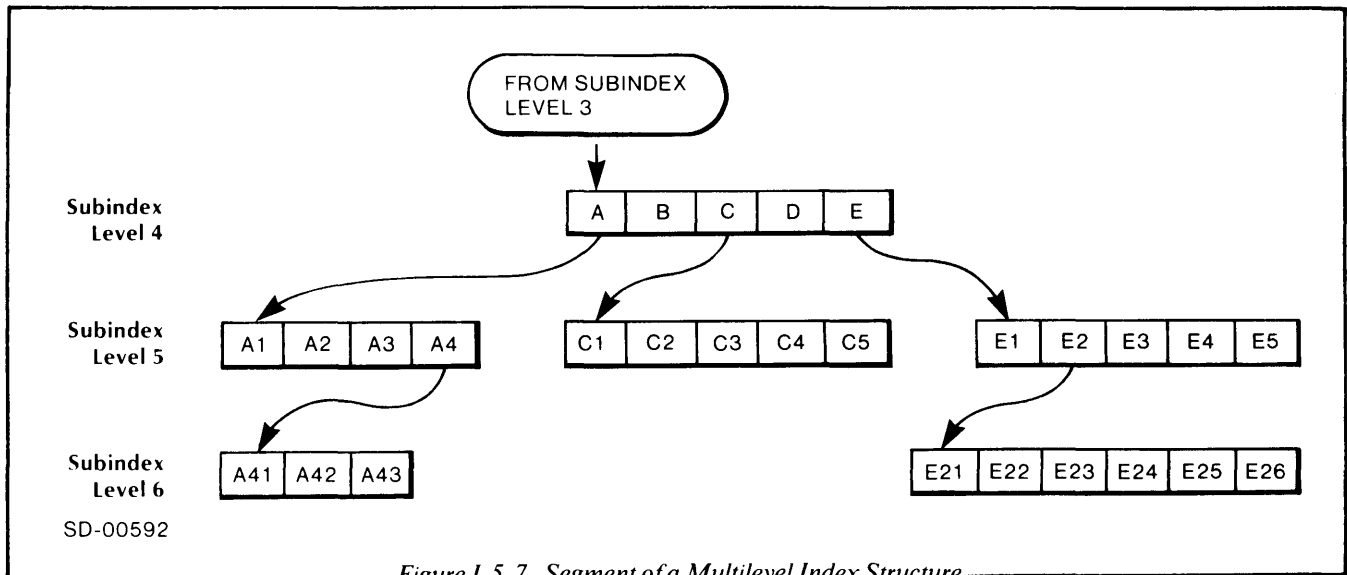


Figure I-5-7. Segment of a Multilevel Index Structure

Now let's say that your current position is on E1 and you want to access entry A41. You could code the equivalent of any of the following:

READ UP, A, A4, A41 (This will return you to E1 after you access A41)

READ UP, A, A4, A41
SET CURRENT POSITION (This will get you to A41 and reset current position there)

READ UP, A, A4, SET CURRENT POSITION
POSITION READ A41 (This will set your new current position at A4 after you access A41)

READ UP, A, SET CURRENT POSITION
POSITION READ A4, A41 (This will return you to A after you access A41)

As you can see, you can reset your current position anywhere along your route from one subindex entry to another.

Now let's look at an example of relative motion in two steps. If you wanted to go from entry E23 to entry D, you would code the equivalent of:

READ UP, SET CURRENT POSITION (if desired),
READ UP, D, SET CURRENT POSITION (if desired).

Licensed Material - Property of Data General Corporation

Notice that you can't go directly from E21 to D in one read motion; you can only specify one motion in each single request.

To summarize, you can use all three accessing techniques - relative, keyed, or combination - in a single program that accesses a DBAM file. Sometimes you'll only want to use one way, i.e., relative motion to sequentially access each entry in an index level, but often you'll find that the combination of relative and keyed will save you access and frustration time.

Remember these four things when you're accessing your DBAM file:

1. Keep track of your current position and, generally, reset it each time you move.
2. Specify only one relative motion direction at a time.
3. Specify complete keyed access paths.
4. Unless you specify otherwise, the system will apply keys downward from the top level of the index.

As long as you keep these points in mind, you should have no trouble getting to the index entries or data records you want.

DBAM Processing Operations

All the ISAM Processing Functions, Utility Functions and Auxiliary Features are available in DBAM. (See Table I-5-7.) However, all these features and functions are enhanced because of the differences between the ISAM and DBAM internal structures and access methods. DBAM also gives you three more Processing Functions, one more Utility Function, and two additional Auxiliary Features. The next sections tell you how DBAM's operations differ from ISAM's.

DBAM Processing Functions

Read

You can read a DBAM file through keyed, relative, or combined access, which allows you to retrieve data from your file easily and quickly. On any read, you can set current position, suppress the retrieval of a partial and/or a database record, and lock or unlock an index entry or a database record. If you access an index entry which has no database record associated with it, you'll get an error message. However, you can avoid this by

requesting suppress database. If you want to read only the database record but not its partial record, you can request suppress partial record. If you (locally) lock an index entry for a database record which has other access paths, other users *can* get to that record via those other paths. If you (globally) lock a data record, no one else can access it through any access paths unless they or you explicitly unlock it.

Write

Each write operation creates a new index entry in a subindex. Therefore, with each write request, you must supply the key that you want to become part of the new entry. As in ISAM, you can only write to a DBAM file through keyed access. If you supply a multilevel key for a write operation, the system will begin writing at the last level of that key. And, unless you specify invert or suppress database, the system will write a new database record to go along with the new key you write. If you specify suppress database, the system will only write the index entry; if you specify invert, the system will write just the index entry and link it to an existing database record. You can also use write in conjunction with optimized record distribution to write a record on the most efficient type of disk.

Table I-5-7. DBAM Processing Operations

ISAM Features Available in DBAM	Additional DBAM Features
<ul style="list-style-type: none"> ● Processing Functions <ul style="list-style-type: none"> Read Write Rewrite Delete Reinstate ● Utility Functions <ul style="list-style-type: none"> Retrieve Status Retrieve Key Retrieve High Key ● Auxiliary Features <ul style="list-style-type: none"> Lock/Unlock Suppress Database 	<ul style="list-style-type: none"> ● Processing Functions <ul style="list-style-type: none"> Define Subindex Link Subindex Delete Subindex ● Utility Functions <ul style="list-style-type: none"> Retrieve Subindex Definition ● Auxiliary Features <ul style="list-style-type: none"> Nonspecific Search Keys Suppress Partial Record

If you are writing new entries in a subindex which contains partial records, the system will expect you to write partial records in your new entries unless you specify suppress partial record for each new entry. Remember also that you must specify duplicate keys if you want to write them; otherwise you'll get an error message. Finally, you cannot write over an existing database record with this command - you must use rewrite.

NOTE: If the system cannot find enough space on a volume to write a new database record, but it can write the index entry, it will write the index entry and send you an error message. However, it will make no attempt to correct the fact that the index entry is not connected to a database record. Therefore, you may get an error message if you attempt to access the key without specifying Suppress Database. To correct this situation, issue a Delete request for the key in question.

Rewrite

This function lets you update an existing partial or database record. As in ISAM, your options for rewriting are the same as for reading; the flow of data just goes in the opposite direction. Rewrite will not create a new index entry; a key must already exist for the record you want to rewrite. To rewrite a partial record only, you specify suppress database.

You can also use rewrite to relocate a record to a different speed disk. And a rewrite inverted will link an existing index entry which is not currently associated with a data record to a record which does have at least one access path.

You can rewrite through keyed, relative, or combined access.

Delete

DBAM delete is the same as ISAM delete: you can physically or logically delete an index entry by local deletion, or a database record by global deletion. However, you cannot physically delete a database record which is accessible through more than one index entry. If you try to do this, you'll only delete the index entry you used to get to the record. You can use any access method for a logical deletion, but you can use only keyed access to physically delete a record or an index entry.

Reinstate

There are no differences between ISAM reinstate and DBAM reinstate; both allow you to remove logical deletion flags from keys or data records.

Define Subindex

When you create a DBAM file, you define your index structure and its main level. However, if you want to use subindexes, you have to define each one individually in your program. This involves specifying the following for each subindex:

- Minimum node size
- Maximum key length
- Partial record length
- Key compression (if you want it)
- No subindexes (if you don't want any subindexes under this one)
- Merit factor for the subindex root (similar to the volume merit factor described above in "Defining Your DBAM Index File" and applicable only if you previously chose optimized record distribution).

Link Subindex

Earlier in this chapter we described your ability to link a subindex to other entries within its index file. This processing request will carry out that link through keyed, relative, or combined access.

Delete Subindex

This function lets you physically or logically delete a subindex from an index entry. You can use all three types of access to get to the index entry whose subindex you want to delete. Once you get to the desired subindex, the system determines whether or not it's linked to any other entry in the index (including its own subindexes). If it is, the INFOS system will unlink the desired subindex but leave it intact. If the subindex you want to delete is not linked to any others, the system will delete it.

DBAM Utility Functions

Retrieve Status

Similar to the ISAM feature of the same name, Retrieve Status lets you determine the status and length of any record through all forms of access. That is, this request operates exactly like a read request, except that you'll get a record's status instead of the record itself. Specifically, the system tells you the following:

- The length of the database record
- The length of the record's key

Licensed Material - Property of Data General Corporation

- The partial record length
- Whether the record's key is a duplicate
- Whether the record is locked
- Whether the record or its key have been locally or globally deleted

Retrieve Key

Using keyed or relative access, or both, you can use this function to find the key for any given record and its occurrence number. If you use relative positioning with this request, the system will tell you the key after it finishes positioning. As in ISAM, this function is handy when you're sequentially processing a file with relative access and you want to know the key for a particular record. If you want to physically delete a record which doesn't contain its own key, this function is necessary because you have to specify a key in order to physically delete a record.

Retrieve High Key

You can find the *final key* in any subindex in a multi-indexed file with this request. (The final key is that which has the highest binary value in the index.) You can also position to any key in the subindex with this request, and you can use any of the forms of access.

Retrieve Subindex Definition

This request will tell you the parameters which were used to define a particular subindex. After the system locates an index entry through one of the access techniques, it will tell you the parameter for that entry's subindex. This is frequently useful in helping produce a backup file for a database. In order to reconstruct a database, a program has to be able to determine the parameters of each subindex it encounters.

DBAM Auxiliary Features

Lock/Unlock and Suppress Database

You can use Lock/Unlock and Suppress Database with a DBAM file just as you can with ISAM. If you don't remember what these do, refer to the Auxiliary Features section of Chapter 4.

Nonspecific Search Keys

Nonspecific search keys allow you to access a subindex even if you don't know the exact key you're looking for. They come in two kinds: Generic and Approximate search keys. A Generic key will give you the first record whose key matches the one you supply, up to the length of the generic key. For example, if you have an invoice file keyed by date, the keys could look like this: 760601, 760602, 760603, etc. (i.e., year, month, day). If you wanted all the invoices sent out in June, 1976, you would give the generic key 7606, and the system would give you the first record whose key began 7606. Then you would sequentially process your file from that point to get the rest. If there were no invoices for June, you would get an error message, "DATA BASE RECORD NOT PRESENT". If there *were* invoices for June, you'd get the record of the first one, whether it was 760601 or 760624. In other words, generic keyed access seeks only keys which begin with exactly the same key characters.

Approximate keys, however, retrieve the first record with an equal *or greater* key value. For example, if you wanted the first invoice sent after July first (760701), you would specify the approximate key 760700 and the system would find the first invoice written on or after July 00, even if it wasn't in July. That is, if the first invoice wasn't written until August eighth (760808), the system would return it to you because there were no keys between 760700 and 760808. However, if you had tried a generic keyed access using 760700, you would have gotten only an error message because the system would not have found any keys which matched 760700.

Suppress Partial Record

This feature allows you to suppress retrieval of the partial record associated with the key and/or data record you're processing. For example, assume you have a Personnel file whose partial records contain salary information. If you want to rewrite an employee's data record to include some new information such as a change in marital status, you can specify Suppress Partial Record to avoid having to rewrite the partial record. This feature is also useful when you want to read an entire database record, not just that part stored in the partial record. (Note, however, that if your partial records contain data which is also in the data record, you are responsible for updating both the partial and the database record.)

End of Chapter

Chapter 6

The RDOS/INFOS Interface

Yes, folks, this is the infamous Chapter 6. As you've read through this manual, you've come across hints about information that "will be explained in greater detail in Chapter 6". Well, we're finally going to reveal those details, including:

- Considerations when you're generating an INFOS system
- Buffer space and buffer management
- File naming
- Disk space allocation
- Using peripheral devices as INFOS files
- How to deal with unlabeled magnetic tapes

We call this chapter the RDOS/INFOS interface because, as we mentioned in the Preface to this manual, the INFOS system uses the RDOS system's file handling abilities as a base for all its operations. Therefore, you have to be aware of certain RDOS system facilities and limitations when you're designing and programming your INFOS application. In other words, this chapter is a compendium of all the system-related information which wasn't particularly pertinent to our earlier discussions.

Considerations When You're Generating an INFOS System

As you know, the INFOS system is a logical extension of the file handling facilities of RDOS. It requires a mapped ECLIPSE computer with a minimum of 128K bytes of main memory. (Don't worry too much about this -- many ECLIPSEs give you up to 512K bytes.) In general, however, the overall effectiveness of your INFOS system depends on how you balance your peripheral equipment and main memory, as well as your typical job mix.

The "other guys' " systems frequently limit you to a single system configuration to satisfy a variety of job mixes. Not so with INFOS; it lets you configure a number of different variations to satisfy the variety of your job mix requirements, and it lets you store all your configurations on a single 'system disk'. Then, once you become familiar with the CLI, you can release one variation and boot another into operation in just a few minutes.

To generate your INFOS systems, use the procedures in our *How to Load and Generate Your RDOS System*, (manual number 093-188). System generation (sysgen) is a straightforward, simple operation in which you supply numeric value responses to sysgen questions. The values you enter determine the size and performance of each INFOS system you generate. Now, to make the most of sysgen, let's examine how you can use the computer's memory.

Memory Space

Think of the computer's memory as having two distinct logical parts: the System Area, which contains the system software and data areas, and the User Area, which contains your application programs. How you set up each of these areas determines how efficiently your system will perform.

The System Area

Your system area will never need more than 64K bytes of memory, and the more you shrink the system memory requirements, the more memory you'll have for your User Area. You control the total size of the System Area through the sysgen dialog. During this dialog, you'll have to specify how much memory you want to allocate to each part of the System Area, i.e., for 'windows', INFOS file control space, system buffers, and resident INFOS/RDOS code. Your specifications here will significantly affect system size and performance, so we'll examine each factor. Figure I-6-1 will help you visualize the components of the System Area.

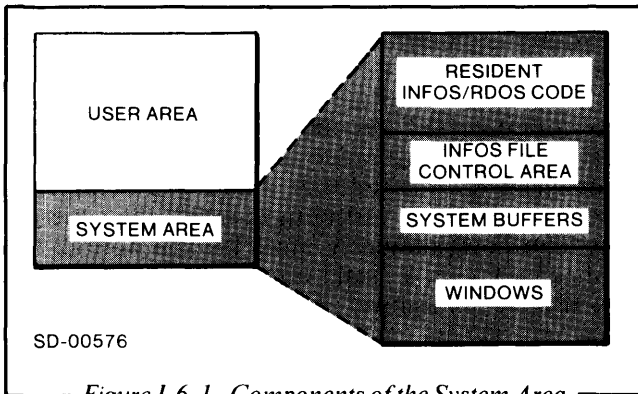


Figure I-6-1. Components of the System Area

Resident INFOS/RDOS Code

The resident code portion of the INFOS system occupies about 4.5K bytes of memory. In the *How to Load and Generate...* manual, there's a list of the various RDOS component sizes which you can use to calculate how much memory the resident RDOS code will take. Notice that you can cut down the size of resident RDOS code if you exclude drivers for peripheral devices which your programs will never reference.

The resident code area also includes space for *stacks* and *cells*. *Stacks* keep track of the progress of each of your programs and make it possible for your programs' tasks to execute concurrent system calls. The system also uses them for spooling and INFOS disk I/O. You can specify a maximum of ten stacks, but you must allow one stack for the system and one stack for each

'ground' in your User Area. That is, if you're using background and foreground, you must specify (at least) three stacks -- one for the foreground, one for the background, and one for the system; if you're just using background, you need only two. Beyond these mandatory specifications, however, you have two options: you can specify one stack for spooling (if appropriate), and/or you can allocate one stack for each task in either ground which will execute system calls concurrently with other tasks.

Each stack requires 530 bytes of memory, and each time you boot the system in, it will give you the number of stacks you specified when you generated the system. During processing, the system uses the stacks as they are needed; that is, a stack must be available to execute a requested function each time that a task (or the INFOS system) issues a system call. If a stack isn't available, the system can't execute the call until another call finishes and releases its stack. Generally speaking, the more stacks you specify for a multitask program, the faster your programs will run. However, be careful: allocation of too many stacks may restrict you when you try to set up your User Area.

Cells are 32-byte pieces of memory which help the system keep track of all your requests. You'll automatically get three cells for each stack you allocate. Beyond this, the INFOS system needs four extra cells of its own and, if you want to spool your output, you'll need two more cells. You can specify up to 64 extra cells at sysgen, and, within limits, the more you specify, the better the system will perform.

System Buffers

System buffers are 542 bytes long. The INFOS system uses them for RDOS data file buffering, to hold the map directory for randomly allocated volumes, and to hold RDOS/INFOS system overlays. In general, the number of system buffers determines how many system overlays will be in memory and how long they'll be there.

The system automatically allocates at least six buffers for RDOS functions; two for each stack you specify. However, at sysgen, you can specify up to 63 extra buffers. As a rule of thumb, you should specify a sufficient number of extra system buffers to hold the overlays that the system needs to carry out an average function. (Note, however, that the number of buffers needed is a dynamic variable that depends on your processing functions.) In general, an optimum number of buffers for SAM processing is seven; for RAM, the optimum is six; and for ISAM/DBAM, approximately eighteen should be adequate. If you're using Space Management, increase each of these totals by two for each volume you are processing.

Licensed Material - Property of Data General Corporation

Windows

During the sysgen dialog, the system will ask you to specify a maximum ISAM/DBAM page size in increments of 2K, 4K, or 6K bytes so that it can allocate enough space for 'windows'. The INFOS system uses two windows to look at the User Area, and two to look at the System Area. Each User window is 2K bytes long, but you can choose a System window length of 2K, 4K, or 6K bytes. Therefore, the *total* space the system needs for windows will be either 8K, 12K, or 16K bytes, according to this formula:

$$(2 * ps) + 4 = \text{Window space in K bytes}$$

where 2 is the number of system windows, ps is the maximum page size (2K, 4K, or 6K bytes), and 4 is the number of K bytes which the system needs for user windows. Thus, the page size you specify at sysgen becomes the maximum physical transfer length for ISAM and DBAM files.

Our experience has shown that an ISAM/DBAM index page size of 1024 bytes is quite efficient for most applications. This means that you'll frequently be able to construct a system using the 2K page size specification. (If you are using big pages, you should also consider specifying Key Compression.) Appendix B of this manual will give you the formulas for calculating ISAM and DBAM page and node sizes, or you can use the INDEXCALC utility (described in the *INFOS Utilities Users' Manual*) to find the optimum page and node size for various ISAM and DBAM index files.

INFOS System File Control Area

The area in which the INFOS system builds a set of control blocks for a file is known as the File Control Area. If the system doesn't have enough room in this area to construct these blocks, you won't be able to open your file. At sysgen, you will specify the size of the File Control Area in increments of 1K bytes to a maximum of 16K bytes. Let's see how to figure out this size.

The INFOS system uses three types of control blocks: File Control Blocks (FCBs), Volume Control Blocks (VCBs), and Buffer Control Blocks (BCBs). You have to use various combinations of these blocks to open each SAM, RAM, index, and database file. For SAM, RAM, and database files, you need:

- 1 FCB;
- 1 VCB for each volume in the file; and
- 1 BCB for each I/O buffer you have allocated in your User Area.

In addition, if you want to use code translation in a SAM or RAM file opening task, you must allow space for the system to build an appropriate translation table in the File Control Area. Make sure that you've accounted for a full set of control blocks for each task that opens a SAM or RAM file. Each type of control block in a SAM or RAM file uses the following number of bytes:

SAM and RAM Files	
• FCBs	304 bytes
• VCBs	Number of volumes * 182
• BCBs	Number of buffers * (70 + (block size + 511)/128)
• Translation Tables	256 bytes per table

For example, to open a single volume SAM file with one buffer and an 80-character block size, you'll need 561 bytes. Here's how we figure it:

One FCB	=	304 bytes
One VCB	=	182 bytes (1*182)
One BCB	=	75 bytes $(1 * (70 + (80 + 511) / 128) = 1 * (70 + 5))$
Total	=	561 bytes

Each type of control block in an ISAM or DBAM database file needs the following number of bytes:

ISAM/DBAM Database Files	
• FCBs	256 bytes
• VCBs	Number of volumes * 182
• BCBs	Number of buffers * (70 + (page size + 511)/128)

The set of control blocks for an *index* file consists of:

- Two FCBs (one for the task and one for the INFOS system);
- One VCB for each volume of the file; and
- One BCB for each I/O buffer you're using.

The system builds a complete set of control blocks for the database file and the index file when your first task opens an ISAM or a DBAM file. Each subsequent task in an open ISAM or DBAM file needs only its own index FCB because it shares the original set of control blocks with the other tasks for that file. You can use these formulas to figure out how much control space you'll need for your ISAM indexes:

ISAM Index Files	
• System FCBs	256 bytes
• Task FCBs	Number of index users*324
• VCBs	Number of volumes*182
• BCBS	Number of buffers*(70 + (page size + 511)/128)

So, an ISAM file which has a one-volume index with a page size of 512 bytes and two buffers, and a two-volume database with a page size of 512 bytes and one buffer needs 1616 bytes:

ISAM Index	
One system FCB	= 256 bytes
First task FCB	= 324 bytes (1*324)
VCBs	= 182 bytes (1*182)
BCBs	= 156 bytes
	$(2*(70 + (512 + 511)/128) = 2*(70 + 8))$
Subtotal	= 918 bytes

ISAM Database File	
FCBs	= 256 bytes
VCBs	= 364 bytes (2*182)
BCBs	= 78 bytes
	$(1*(70 + (512 + 511)/128) = 1*(70 + 8))$
Subtotal	= 698 bytes
Grand Total	= 1616 bytes (918 + 698)

NOTE: The next task to use this open file will only need 256 bytes (i.e., one task FCB) because it shares all the other previously allocated control blocks.

Now, when you're dealing with a *multiple-indexed* DBAM file, the system will allocate one full set of control blocks for each index that you open, but only one set of control blocks for the database. That is, your first DBAM opening task will need a full set of index *and* database blocks, but if you open the same database with another index name, you only need to account for a set of *index* control blocks; subsequent opening tasks will share the database blocks allocated at the first task's opening.

Licensed Material - Property of Data General Corporation

Each DBAM index you open uses the following control space:

DBAM Index Files	
• System FCBs	256 bytes
• Task FCBs	(Number of index users*304) + (Number of index levels*20)
• VCBs	Number of volumes*182
• BCBS	Number of buffers*(70 + (page size + 511)/128)

For example, let's say that you're opening a multiple-indexed DBAM file which has a two-volume index with five levels, a page size of 1024 bytes, and four buffers; also, its database has six volumes, six buffers, and a page size of 2048 bytes. The first task that opens the file will need 3240 bytes:

DBAM Index	
System FCBs	= 256 bytes
Task FCBs	= 404 bytes ((1*304) + (5*20))
VCBs	= 364 bytes (2*182)
BCBs	= 328 bytes
	$(4*(70 + (1024 + 511)/128) = 4*(70 + 12))$
Subtotal	= 1352 bytes

DBAM Database	
FCBs	= 256 bytes
VCBs	= 1092 bytes (6*182)
BCBs	= 540 bytes
	$(6*(70 + (2048 + 511)/128) = 6*(70 + 20))$
Subtotal	= 1888 bytes
Total	= 3240 bytes

Note here, also, that your next task to open this file under the same index will only need its own task FCB, i.e., 404 bytes. However, if you wanted to open the same database under a different index, you would have to account for space for that index's blocks.

For instance, let's open the database in the above example through a three-volume index with four index levels, a page size of 1024 bytes, and two buffers. In this case, the system will need an additional 1350 bytes of memory:

DBAM Index	
System FCBs	= 256 bytes
Task FCBs	= 384 bytes ((1*304) + (4*20))
VCBs	= 546 bytes (3*182)
BCBs	= 164 bytes
	$(2*(70 + (1024 + 511)/128) = 2*(70 + 12))$
Total	= 1350 bytes

Licensed Material - Property of Data General Corporation

Finally, a note on what happens when you close a file. When a task closes a SAM or RAM file, the system releases the file's control space for use by other tasks. In multiple-opened ISAM and DBAM files, however, the system does not release the space occupied by the control blocks until the last task using the file closes it. That is, if a task closes a multiple-opened ISAM or DBAM file while other tasks are still using the file, the system only releases the space for the closing task's FCB.

The User Area

Once you've figured out how much memory space you'll need for the System Area, subtract your answer from the total amount of memory and you'll know how much you have left for your program and your I/O buffers. Of course, the amount of memory you *need* in the User Area depends on your job mix. For example, if your job mix requires only one application program, you can use a simple background configuration; if it requires two programs, you'll probably want to partition the User Area into a background/foreground configuration, as shown in Figure I-6-2.

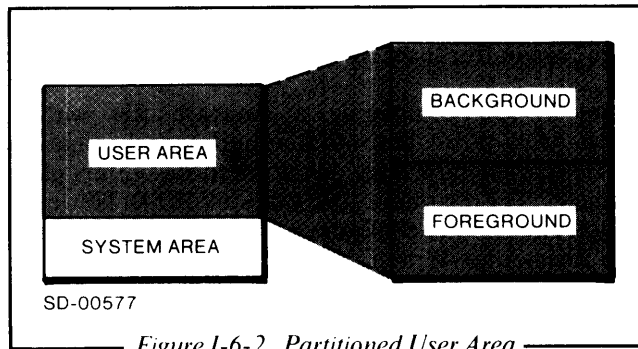


Figure I-6-2. Partitioned User Area

Even though both 'grounds' are the same size in Figure I-6-2, they don't need to be, and, in fact, you will specify the size of each ground at runtime with the CLI's SMEM command. In general, the size of your programs plus their buffer requirements will tell you where to set the partition. For instance, suppose that your largest foreground program is a single-task, 60K byte DBAM program which uses four 1024-byte index

buffers and two 512 byte database buffers. You'll need 5120 total bytes for your buffers, which means that the total memory requirement for the foreground is 65K bytes, as illustrated in Figure I-6-3.

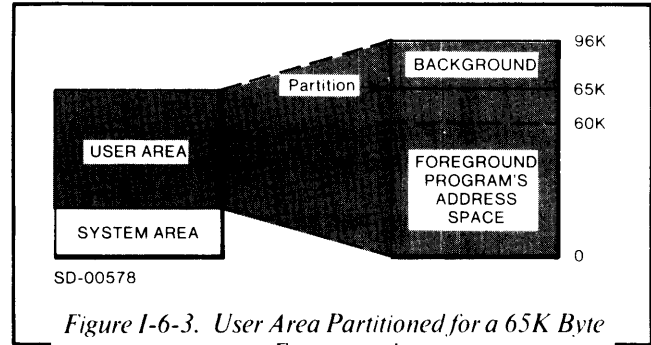


Figure I-6-3. User Area Partitioned for a 65K Byte Foreground

The buffer space (in either 'ground') which your application needs will basically depend on how many INFOS files your program will have open at any one time. The next section of this chapter will show you how to calculate buffer space.

The difference between your program size and the size of its 'ground' is the INFOS I/O buffer area or the ground's Virtual Memory. When you open a file, in fact, you may get the error message "OUT OF VIRTUAL MEMORY", which means that there isn't enough room in the ground's virtual memory to build the buffers for the file you're trying to open. Most of the time, however, you can solve this problem simply by resetting the partition with a CLI SMEM command.

Summary

Figuring how much memory the system will need and how much you'll have left involves a series of trade-offs. Fortunately, the INFOS system lets you vary your parameters rather widely to get the most efficient system configuration. Play with your values before you start to generate a system; very often you'll find that what you want to do fits easily into your User Area. But, if it gets tight, just remember all the options you have considering page size, window size, number of stacks and cells, and system buffers. By properly manipulating all of these variables, you can mold your system to fit your application like a glove.

I/O Buffer Space

In the previous section, we mentioned that the system allocates I/O buffers in a ground's virtual memory. Any task in either ground can open any INFOS file, but the amount of virtual memory that the system needs for a program in either ground depends on the following:

1. The number of unique files you want to open, and the number of concurrent openings of any of these.
2. The block or page size of each unique file you're going to open.
3. The number of buffers for each unique file.
4. Whether or not you requested Read-After-Write verification.

Items 2, 3, and 4 are self-explanatory, but let's look at number one. For a task to open any SAM or RAM file, there must be room in virtual memory to allocate the number of buffers requested by the task. In other words, each task uses its own buffers, it does not share them with other tasks. Therefore, if there is no room in virtual memory for the number of buffers a task requests, the system cannot perform that task.

In ISAM and DBAM, however, this is not the case. The first task that opens an ISAM/DBAM file will get its requested number of buffers, and each subsequent opening for that file will use those same buffers, but no more than those, no matter how many the subsequent opening requests. Therefore, if you're working in SAM or RAM, make sure you've got enough virtual memory space to hold all the buffers for all the tasks you're going to want to perform. If you're working in ISAM or DBAM, make sure that the number of buffers you request in your first task is sufficient for your later tasks.

Licensed Material - Property of Data General Corporation

Also, with regard to number four above, your opening task in ISAM and DBAM must request Read-After-Write verification if you want to use it in any subsequent tasks. If the first task does not specify Read-After-Write, you will not be able to use it later for that file, even though you ask for it. The system will simply ignore your subsequent request.

Finally, similar to what happens in the System Area, the INFOS system will release the buffer space for a SAM or RAM file when your last task closes the file. However, it will not release ISAM/DBAM buffer space until the last task using the file closes it.

Figure I-6-4 shows you how to compute buffer space requirements (in bytes) for the opening of each type of file, both with and without Read-After-Write verification.

For example, referring to Figure I-6-4, suppose you want to open a SAM file which has three buffers and blocks that are 512 characters long, and you want to use Read-After-Write. You'll need 3072 bytes worth of virtual memory for the opening task: $2 * 3 \text{ buffers} * 512 \text{ bytes per block}$. (But don't forget to make allowances for later tasks which will open that file -- they will need their own buffer space, too.)

Now if you're opening a database file without Read-After-Write, but with four buffers and a page size of 1024 bytes, how many bytes are those buffers going to need? The answer is 4096 bytes ($4 * 1024$). However, if you choose this configuration, you won't be able to use Read-After-Write for later tasks in this file, nor can you use more than these four buffers. So plan ahead and you won't unnecessarily limit your processing capabilities.

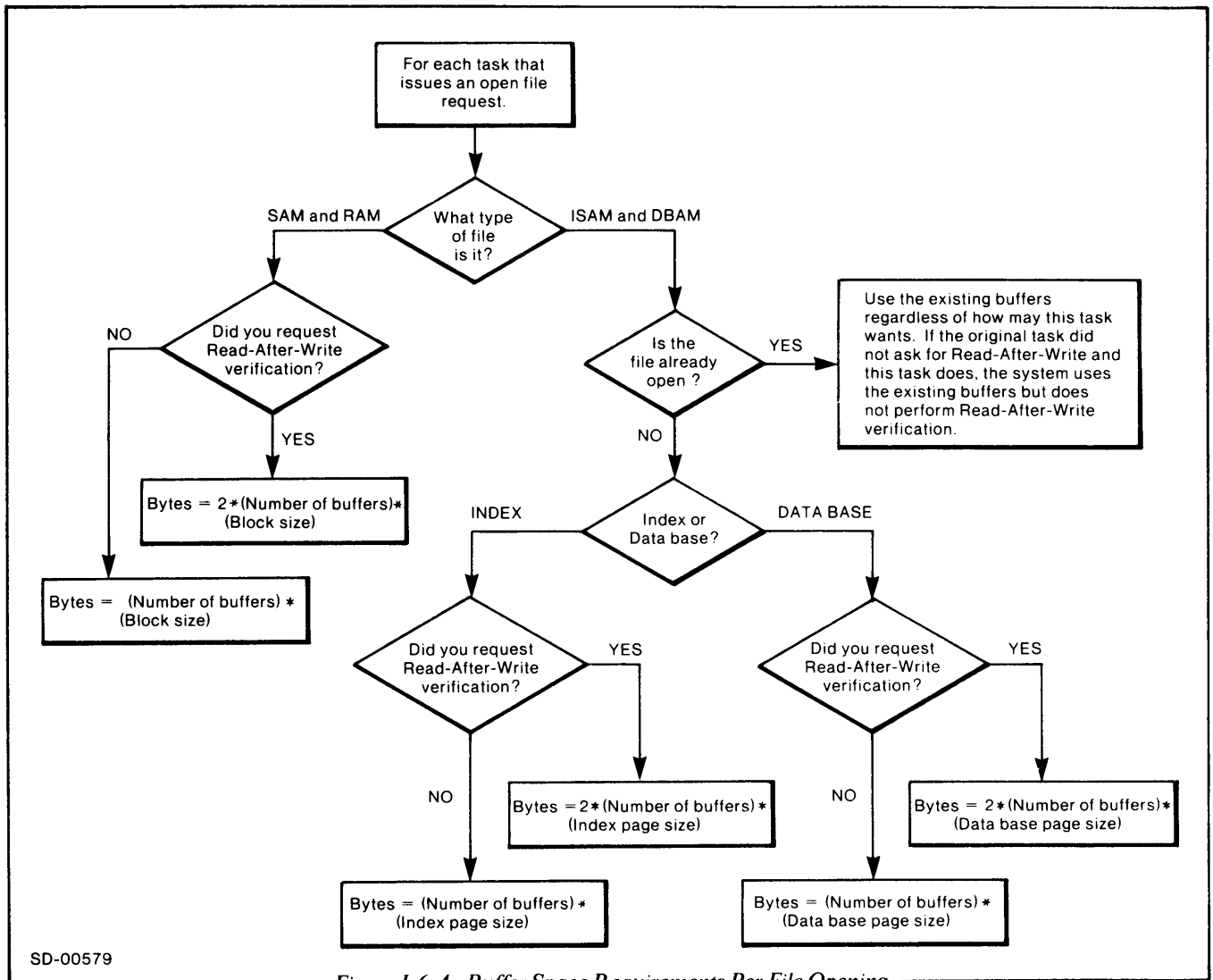


Figure I-6-4. Buffer Space Requirements Per File Opening

I/O Buffer Management

In general, the INFOS system regulates buffers according to the Least-Recently-Used technique (LRU). Under this technique, the system keeps track of the buffer whose contents it accessed the longest time ago. This, of course, is the least recently used buffer. Figure I-6-5 illustrates the LRU technique for input; the technique for output is the same, except that the data movement between your data area and the buffers goes in the opposite direction.

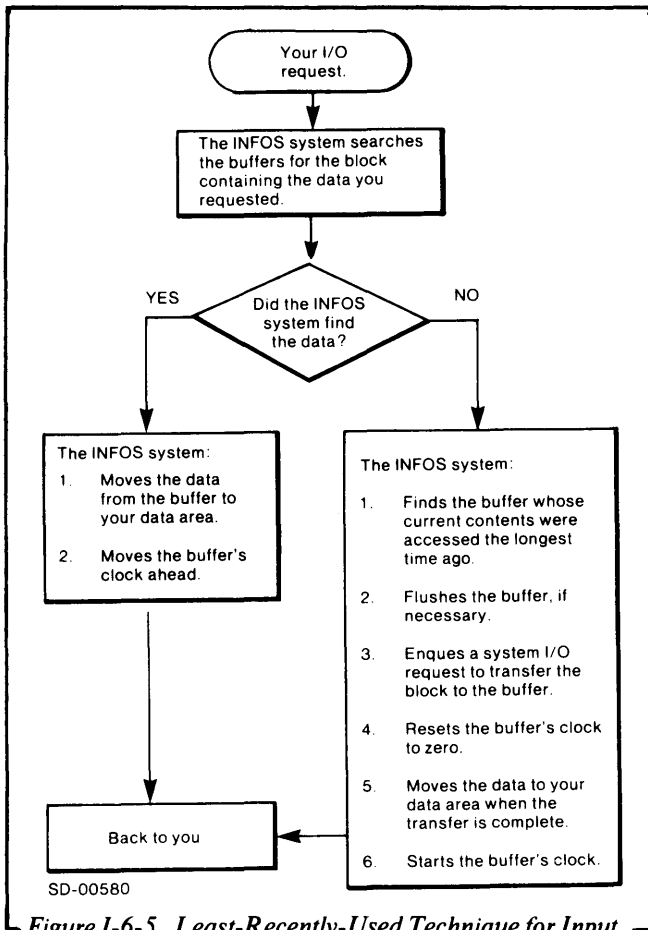


Figure I-6-5. Least-Recently-Used Technique for Input

Due to the nature of SAM file processing in the Input mode, the INFOS system automatically prereads the next block you're going to want while you're processing your current one (as long as you allocate more than one buffer). Thus, SAM file buffers are always kept full. On Write requests in the Output mode, the system will automatically write the contents of a buffer to your file as soon as that buffer is full. (Note, therefore, that you will speed up your processing considerably by allocating more than one buffer.) For SAM file processing in the Update mode and all RAM file processing, the system uses the basic LRU technique.

ISAM and DBAM index file buffers, however, can be a slightly different story. Normally, the system manages these buffers according to a hierarchically modulated LRU, but you can disable the modulation at runtime. When you're designing your file, you can specify certain root nodes as High Priority Nodes (HPNs), and, when your first task opens your DBAM file, you can request that the system not hierarchically modulate the LRU. (Root nodes, remember, are the 'doors' into each subindex level; we'll explain hierarchical modulation in a minute.)

In certain DBAM applications, you're going to want to access your file as quickly as possible. One way to get maximum access speed is to keep your root nodes in memory as long as possible because they are the subindex contact points you need most frequently to locate index entries. So, when you design your DBAM file, you can specify that the root nodes of certain critical subindexes are HPNs. Then, when you first access these nodes, the INFOS system will keep them in buffers as long as possible. In other words, HPNs modify the more general LRU technique. Therefore, because of the way that the system handles HPNs, your first task should ask for a relatively large number of I/O buffers.

We use the term 'relatively large' because the number of buffers for HPNs depends on your application and how you set up your index levels. Sometimes you'll want to use one buffer per HPN (if you've got enough virtual memory), at other times, you'll only want enough buffers to hold the HPNs in the portion of the index structure which your program is processing. Note, however, that you need your HPN buffers *in addition* to those which you normally need to access your DBAM index structure. Figure I-6-6 illustrates an index structure with sixteen HPNs.

If you have enough virtual memory and sufficient room in the system's File Control Area for the buffer control blocks, your first task could open the file in Figure I-6-6 with 25 I/O buffers; sixteen for the HPNs and the rest for normal index access. If this isn't practical, and if your program only uses one section of the index (i.e., the left-, center-, or right-most branch in Figure I-6-6), then you could specify about fifteen I/O buffers when your first task opens the file. This would insure that the system would keep the HPNs for the section you're using in memory, and still have enough buffers left over for your normal index processing.

Licensed Material - Property of Data General Corporation

and the rest for normal index access. If this isn't practical, and if your program only uses one section of the index (i.e., the left-, center-, or right-most branch in Figure I-6-6), then you could specify about fifteen I/O buffers when your first task opens the file. This would insure that the system would keep the HPNs for the section you're using in memory, and still have enough buffers left over for your normal index processing.

Alternatively, if you're short on buffers but you still want top speed, use the system default of hierarchical modulation for the LRU when you open the file, and don't designate any HPNs when you create it. Thus you won't fill all your buffers with root nodes, but rather the system will automatically use the buffers for the most applicable nodes in each subindex level. In other

words, the system will use the same buffers over and over (according to LRU) on various index levels, essentially giving you the root nodes. However, since you don't have to specify HPNs, the system doesn't tie up buffers with some root nodes which you may or may not want to use as frequently as some others.

In general, your buffering decisions are going to come down to a tradeoff between speed and space. For many applications, the basic LRU technique is perfectly adequate and efficient. However, if you want to squeeze all the speed out of your system that you can, you can always modify the LRU in a DBAM file with High Priority Nodes, or hierarchical modulation, or, occasionally, both. You can even combine these with key compression and relative processing to give you top speed.

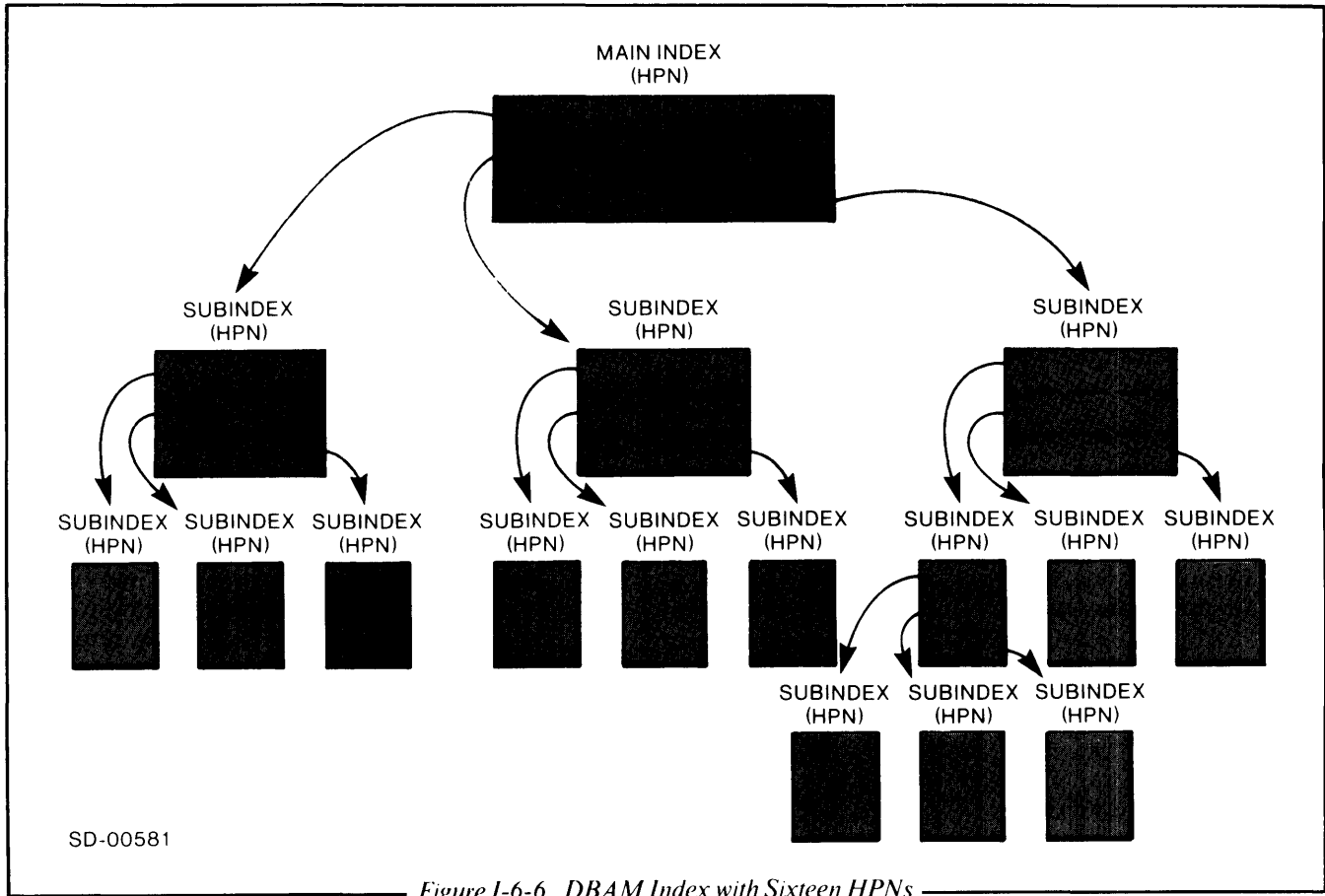


Figure I-6-6. DBAM Index with Sixteen HPNs

File Naming

To access all devices and disk files, you have to give the system their full filename; to access cassettes and magnetic tapes, you need their full file number. A filename can consist of upper or lowercase letters, numbers, and/or the \$ character; for example, you could name your file Payroll, or MT0, or \$LPT. Your filename can contain any number of these characters, although the system will only look at the first *ten*.

You can also append an extension to a filename. An extension is another set of alphanumeric characters, including \$; again, however, the system only looks at the first *two* characters in an extension. In addition, you must put a period (.) between the filename and the extension; for example, PAYROLL.PS. Your only limitation here is that you shouldn't name a source file "(filename).SV", because of the confusion it would cause with a system-named save file.

The system itself also appends extensions to filenames to indicate the type of information they contain and to distinguish them from other files connected to the same source file. For instance, if you named your source file A.SR, the system might produce files from this source called:

- A.RB Relocatable Binary file
- A.SV Core Image (Save file)
- A.LS Listing file
- A.OL Overlay file
- X.VL Permanent File Specification for volume A
- X.IX Permanent File Specification for all the indexes for the database file X created by A.

Finally, after you give an index file a name at creation, you must always refer to that index and its database by the full creation name. For example, if you name an index file DP0:1, you must always call it DP0:1; you can't merely call it "1", even though you may be working within that directory.

Disk Space Allocation

If your INFOS file resides on a disk, you can allocate the space on that disk randomly or contiguously. When you choose random allocation, the system allocates disk sectors as they are needed, regardless of their physical location on the disk. Hence, the system keeps sector addresses on the disk in sector address maps. Therefore, the system may require many disk accesses to retrieve a given record: one (or more) to retrieve the record address from the map, and one to retrieve the actual record. These potential multiple accesses can seriously affect performance for a large file, but random allocation does have two major advantages:

- You can use your disk space much more efficiently because the system doesn't allocate sectors until it actually needs them.
- There's a better chance of finding room for a full volume of randomly allocated data than for contiguous data because the largest contiguous block of free disk sectors will almost always be smaller than the total number of free disk sectors.

With contiguous allocation the system allocates adjacent blocks of disk sectors when you create your file. This gives contiguous allocation four advantages:

- The system doesn't need sector disk maps because it can directly compute sector addresses.
- You can often use multiple sector transfers because the sectors are physically adjacent. Hence a moving-head disk doesn't have to move very much at all and your access time improves considerably.
- Your writing operations are more efficient because the system makes room for your entire body of data when you create your file; therefore it doesn't have to find room for new data during the processing operation.
- You know at the start that there is enough room for your whole file or whatever section you want to put there.

If you're having trouble choosing an allocation technique for your file, remember that different volumes of your file can have different allocation specifications. For example, if you have a database file on two volumes, you can allocate the first one contiguously with a size adequate to accommodate the amount of data you expect to have. Then you can choose random allocation for the second volume to handle any unexpected growth.

Using Peripheral Devices as INFOS Files

When you want to use peripheral devices (e.g., line printers, paper tape readers, teletypes) as INFOS files, you're going to find that your block size is controlled by the physical limitations of the device. That is, you cannot use a block size larger than the number of characters in a line of the device, except in certain circumstances. For example, you cannot use a block size larger than 80 or 132 characters if you're using a line printer because the printer will just print any characters beyond that length on top of each other and you'll get a blob at the end of each incomplete line.

However, as we said, there is a way to avoid this. You can use any size block you want with an 80-character line printer as long as you include a carriage return or a line feed delimiter at least every 80 characters. A simple analogy here would be a typewriter. Every 80 characters or so, you have to hit the carriage return so that your words stay on the page. And so it is when you're using a line printer (or a similarly limited device) as an output file. Simply insert a carriage return (or a line feed) at least every 80 characters and you can use any size block you want.

Something else to consider when you want to use a peripheral device as an INFOS file is that you can adjust the timeout intervals to suit your application. That is, the system has different timeout intervals built into it for each type of peripheral device. When the system cannot do output or get input after one of these intervals, it sends a message to your console. If it then can't complete your processing function after twelve intervals, the system closes that file. For example, suppose you're writing to a line printer as a SAM output file and the printer jams. The system will try to write, but won't be able to. So, after fifteen seconds, it will send you the message "DEVICE TIMEOUT ON FILE (filename)", and, at the same time, will try to write again. If the system goes through this cycle twelve times and still cannot complete the write request, it will close the file. However, if you've got a job that you want to process in a hurry, you may not want to wait fifteen seconds to find out that there's something wrong; in this case, you can change the timeout interval to two or three seconds so that you know immediately when there's a problem.

For the record, here are the timeout intervals for several types of peripheral devices:

Device	Interval
Teletype (input)	Indefinite (The system will wait forever for you to input data)
Teletype (output)	30 seconds
Paper tape printer/reader	30 seconds
Punched card reader	10 seconds
Line printer	15 seconds
CRT	10 minutes
Magnetic tape	15 seconds
Cassettes	10 minutes
Fixed-head disks	3 seconds
Moving-head disks	5 seconds

NOTE: See Appendix E for a full chart showing device characteristics.

How to Deal with Unlabeled Magnetic Tapes

When you want to read data from an unlabeled magnetic tape, simply specify the name of the drive on which you mounted it, and then a sequence number for the section of the tape you want. For example, if you have an unlabeled tape and you mount it on tape drive number one, its name becomes MT1. Then, if you want to read the data which lies between the beginning of the tape and the first tape mark, you'd code the equivalent of "READ MT1:0". (Data segments on a tape are numbered consecutively starting with zero.) If you wanted to read the second section of the tape, you'd code the equivalent of "READ MT1:1". That's all there is to it. The system will start reading wherever you tell it to, and will keep reading until it comes to a tape mark, even if that mark is at the other end of the tape.

By the way, you can also use the above procedure to read the header labels on a labeled magnetic tape. Just pretend it's an unlabeled tape (the system won't know if you don't tell it) and consider the header labels as the first section on the tape.

End of Chapter

Part Two: Programming Your RDOS/INFOS System

General Information

Sequential Access Method (SAM) Files

Random Access Method (RAM) Files

**Indexed Sequential Access Method
(ISAM) Files**

Data Base Access Method (DBAM) Files

Packet Formats

**How to Use the Macroassembler with
the INFOS System**

Prefatory Note

We have designed the Programming section of this manual for use by experienced Programmers. You cannot learn *how* to program here, just *what* to program. Furthermore, our discussion is geared primarily toward Assembly Programmers, although those of you who will be using FORTRAN will also find the information pertinent. (Appendix C, *The FORTRAN-INFOS Interface* will fill any gaps left by the main text.)

If you're going to use COBOL or RPG II, you will find that the details in this section are not *directly* applicable, but they will give you a general idea of what you need to include in your programs, as well as insight into the way that the COBOL and RPG II runtime routines communicate with the INFOS system. For further details on Data General's COBOL and RPG II and their applications, we recommend that you also read our *COBOL Reference Manual* (number 093-180) or our *RPG II Programmer's Reference Manual* (093-117).

End of Preface

Chapter 1

General Information

Packets

Your program and the INFOS system communicate with each other through Packets and Tables. A packet is a group of logically related control parameters, from which you specify those features (both required and optional) which you want to use in a particular system call. For example, to do a read operation, your program must supply a Processing Packet to tell the system how to find and where to place a record in memory. You can also specify various options in that packet, for example, record lock or space management. When you make the system call to do the read, you indicate the address of the Processing Packet. And, after the system completes the read operation, it places the length of the record it read (along with other pertinent information about the record's status) in the Processing Packet before it returns to your program.

Each type of packet has a specific format. That is, each packet is a certain number of words long, and each parameter has a specific location within each packet. Some parameters only occupy a single bit location in a given word, while some occupy as many as three words. Furthermore, most packets contain at least one two-word field which is used as a pointer field. There are two types of pointers:

- *Word* pointers, which point to the first word of such system elements as other packets or Volume or Key Tables; and
- *Byte* pointers, which point to the first byte of elements which you supply, such as names or user tables.

Note also that if a packet contains a pointer field and the particular operation you are performing does not require that address (or if you want to use a system

default value for a particular parameter), you must enter -1 in the first word of the field. If you don't, the INFOS system will consider the contents of that field to be a valid address, and it may attempt to use that address in processing the call.

Packet Types

The INFOS system uses a different type of packet for each of the following functions:

- File definition
- Volume definition
- General processing (SAM and RAM)
- Extended processing (ISAM and DBAM)
- Key definition
- Subindex definition
- Point processing
- Link subindex processing
- Volume initialization

Since this manual will refer to each of these packet types, we will briefly describe each one now. For further details on the format and contents of each one, however, refer to Chapter 6 in this part of this manual. For descriptions of the calls you use to build each packet type, see Chapter 7 in this part (for assembly language) or Appendix C (for FORTRAN).

File Definition Packets

You must build a File Definition Packet (FDP) for each INFOS file which your program opens. The FDP contains a description of the file. If the file does not exist, but will reside on a disk, the system will use most of this information to build the file's Permanent File Specification. Then, on subsequent openings of that file, the system will refer to the Specification for this information. If your file is not disk-resident, however, you will have to respecify the file on each opening, since the system does not maintain a Permanent File Specification for this type of file.

Volume Definition Packets

A Volume Definition Packet (VDP) describes the physical characteristics of a file. When you open any INFOS file in the Output or Create Update mode, you have to build a VDP for each volume of the file, then put all the VDPs for the file into a Volume Table. Once you've created the file and defined the volumes, you only have to give the system the address of the VDP for the first volume in the file when you want to reopen the file for updating. The INFOS system automatically records the volume definitions and uses them when your program needs them.

When you open a *tape* file, however, you must build a Volume Table consisting of one VDP for each reel of tape. Unlike disk files, the system does not permanently record tape file VDPs.

Similarly, you must build a VDP for each TTY and printer file which you open for processing, since these are single-volume files by definition.

General Processing Packets

This type of packet contains the information which the system needs to perform a SAM or RAM file processing request. It also provides the means for the system to return general status information to you. Therefore, whenever you issue a call, you must include the address of the appropriate General Processing Packet.

Extended Processing Packets

When you issue an ISAM or DBAM function request, you must supply the address of the Extended Processing Packet containing the information which the INFOS system needs to successfully process the call. Since this type of packet is just an extension of a General Processing Packet, it will also return general status information to you.

Key Definition Packets

Each Key Definition Packet describes a single key. You must build one packet for each key, then place all the packets in a contiguous Key Table. The INFOS system will use each key in the table for processing at the subindex level indicated by the packet's relative

Licensed Material - Property of Data General Corporation

position in the table. That is, for a keyed access, the system will apply the first key in the table at subindex level 0, the second at subindex level 1, and so forth. Thus, whenever you use keyed access for an ISAM or DBAM processing request, you will set one field of the Extended Processing Packet to point to the first word of your key table.

Subindex Definition Packets

You will only use this type of packet when you issue a Define Subindex function request. As you might think, the contents of this packet describe the subindex you wish to define. You point to this packet through the Extended Processing Packet which you use to initiate the request.

Point Processing Packets

This packet describes the existing point within a SAM disk file where you want the system to position your program. You can only use this packet with a Point function request when you're processing a SAM disk file.

Link Subindex Processing Packets

This packet tells the system the key and the subindex you wish to link. The only time you'll use this packet is when you want to issue a Link Subindex function request.

Volume Initialization Packets

This packet tells the system how you want to initialize a given labeled tape file. You must build one of these packets for each labeled tape volume you wish to initialize. Full details on labeled tape initialization procedures are in Appendix A.

Tables

The INFOS system uses two types of tables. One type is simply a contiguous collection of logically similar information -- for example, a User Header Label Table, which contains one or more user-supplied header labels. The other type consists of a contiguous sequence of packets. An example of this is a Volume Table, which is made up of one or more Volume Definition Packets.

How to Open an INFOS File

You can use the following set of steps to open any INFOS file in any processing mode:

1. Construct the packets and tables required by the access method, the processing mode, and the device type you are going to use, as explained in the other chapters in this section.
2. Issue a Pre-open system call as described in the following section. The INFOS system will examine the contents of the FDP and resolve all the defaulted parameters before it returns to your program.
3. When the system returns, issue an Open system call as described below. The system will then construct the required control blocks and tables in its file control space, allocate the I/O buffers in virtual memory, assign a pseudo channel number to the file, and return that number in AC1. You should make sure this number is in AC1 whenever you issue subsequent system processing calls for this file.

If the INFOS system encounters an error while processing a Pre-open or Open request, it may return one of the following error codes to AC2.

NOTE: You may also receive an RDOS File System error on Pre-open or Open; see the *RDOS Reference Manual* for a list of those.

Common Pre-open Error Messages

Code	Description
212	SYSTEM FILE PROCESSING ERROR
213	UNRESOLVED RESOURCE CONFLICT
221	SYSTEM FILE OPEN ERROR
241	PRE-OPEN CLOSE ERROR
405	VERSION CONFLICT ERROR

Common Open Error Messages

Code	Description
206	FILE IN USE
207	FILE LOCKED
212	SYSTEM FILE PROCESSING ERROR
213	UNRESOLVED RESOURCE CONFLICT
215	DUPLICATE SYSTEM FILE
216	SYSTEM FILE READ ERROR
217	SYSTEM FILE WRITE ERROR
220	ILLEGAL FILE NAME
221	SYSTEM FILE OPEN ERROR
223	INSUFFICIENT FREE SPACE FOR OPEN
226	NO SUCH VOLUME
237	ILLEGAL TRANSFER REQUEST
241	PRE-OPEN CLOSE ERROR
242	FILE CLOSE ERROR
244	VOLUME ALREADY EXISTS
251	NAME TOO LONG
254	DEVICE NOT SUPPORTED

System Calls

You make INFOS system calls in the same way that you make RDOS system calls, except that you must first load AC2 with a pointer to the first word of the packet associated with that call. The system calls and their associated packets are as follows:

- To Pre-Open a file:

AC2 = Pointer to File Definition Packet

```
.SYSTEM
.INFOS
error return
normal return
```

- To Open a file:

AC2 = Pointer to File Definition Packet

```
.SYSTEM
.OINFOS
error return
normal return
```

NOTE: When the INFOS system successfully completes an .OINFOS call, it will return a pseudo channel number for the file to AC1.

- To initialize volumes of a labeled tape file:

AC2 = Pointer to Volume Initialization Packet

```
.SYSTEM
.IINFOS
error return
normal return
```

- To process a file:

A) Pointer to either General Processing Packet
AC2 = (for SAM/RAM) or Extended Processing Packet (for ISAM/DBAM). Also, when you use the following call, you must place the file's pseudo channel number in AC1.

```
.SYSTEM
.INFOS argument
error return
normal return
```

The arguments for the .INFOS call are:

Octal Value	Argument	Function
00	.POINT	Move your position to a new point in a SAM disk file.
01		Reserved for sysem use.
02	.FEOV	Force End-Of-Volume.
03	.ICLOSE	INFOS Close.
04	.SETX	Set Exclusive Use.
05	.RELX	Release Exclusive Use.
06	.TRUNC	Truncate a block.
07	.IREAD	INFOS Read.
10	.IWRITE	INFOS Write.
11	.DEFSI	Define Subindex.
12	.LNKSI	Link Subindex.
13	.DELRC	Delete a record.
14	.DELSI	Delete a Subindex.
15	.RETST	Return data record status.
16	.RETHK	Return high key in subindex.
17	.RETKY	Return key.
20	.REINS	Reinstate a logically deleted record or index entry.
21		Reserved for system use.
22		Reserved for system use.
23		Reserved for system use.
24		Reserved for system use.
25	.REWRT	Rewrite.
26	.RETDF	Return subindex definition.
27	.PRERD	Pre-Read.

Finally, whenever the INFOS system takes the error return to any system call, it returns an error code to AC2. (Appendix D contains explanations of all INFOS error codes.)

The Permanent File Specification

When you create an INFOS disk file by opening it in the Output or Create Update mode, the system builds a Permanent File Specificaion (PFS) from the information you supply in the File Definition Packet (FDP) used in the Pre-open system call. Each FDP contains two types of parameters: those which are unchangeable, and those which apply only to this opening of the file. The unchangeable parameters define the file's logical and physical characteristics; for example, volume size and record format. Table II-1-1 shows these unchangeable FDP parameters and their default values (if any).

The second type of parameters, shown in Table II-1-2, become part of the PFS, but you can override their PFS value for any opening. Thus, when you close the file, the system remembers the values you specified for these parameters, but lets you alter them on subsequent openings. Thus you can choose some of the features for one opening, and others for another. An example of this type of parameter is the number of I/O buffers. The number of buffers you specify at file creation becomes part of the PFS, but you can specify any other number at any subsequent opening of the file. You can also default the number of buffers for any opening after file creation, and the system will allocate the number recorded in the PFS.

B) Alternatively, you may use the following processing call:

AC2 = Address of the Processing Packet
AC0 = Octal value of your desired argument

.SYSTEM
.INFOS 77
error return
normal return

Table II-1-1. FDP Parameters that Become Unchangeable Entries in the File's PFS

Feature	Parameter	System Default Value
Unblocked Records	F1UBR	Blocked Records
Access Method	F1AM1 F1AM2	None
Record Format	F1FT1 F1FT2	None
Space Management	F2SPM	No Space Management
Optimized Distribution	F2ORD	No Optimized Distribution
Block or Page Size	FDBLK	512 Bytes
Record Length	FDLEN	Block Size
Number of Index Levels	FDNIL	One
Number of Volume Table Entries	FDNVD	One
Initial Node Size	FDINS	Page Size Minus 6 Bytes
Maximum Key Length	FDMKL	None
Partial Record Length	FDPRL	Zero
Root Node Merit Factor	FDRMF	Zero
Key Compression	IXPKC	No Key Compression
No Subindexes	IXNSI	Subindexing Allowed
High Priority Node	IXHPN	No High Priority Node

Table II-1-2. FDP Parameters that Become Runtime Entries in the File's PFS

Feature	Parameter	System Default Value
Read-After-Write Verification	F1RAW	No Verification
Overwrite	F1OVR	Write At End-Of-File
Exclusive File	F1EXF	Nonexclusive Use
Rewrite Mode	F1RER	No Rewriting
Disable Hierarchical Replacement	F2DHR	Not Disabled
Volume Table Pointer	FDVTP	None
User Input Translation Table Pointer	FDUIT	No Translation on Input
User Output Translation Table Pointer	FDUOT	No Translation on Output
Selective Field Translation	TCSFT	No Selective Field Translation
Data Sensitive Delimiter Table Pointer	FDDSD	Null, Carriage Return, Form Feed
Selective Field Translation Table Pointer	FDSFT	No Selective Field Translation
Number of Buffers	FDBUF	2 for Indexes 1 for others
Processing Mode	F1PM1 F1PM2	None

End of Chapter

Chapter 2

Sequential Access Method (SAM) Files

This chapter contains procedure-oriented descriptions of how to open a SAM file under various conditions, and how to use the SAM processing functions. Please refer to Chapter 2 of Section One or to Appendix A if you have any questions about the options discussed here.

How to Open SAM Files

You can open your SAM file in the following ways:

Disk files:

- In the Create Update mode (for all functions)
- In the Output mode (for writing only)
- In the Update mode (for all functions)
- In the Input mode (for reading only)

Labeled Magnetic Tape files:

- In the Output mode (for writing)
- In the Input mode (for reading)

The procedure for opening any of these files is the same; just follow these steps:

1. Set up the following:
 - File Definition Packet (FDP) (Figure II-6-1) (Use macro call BLDFDP; see Chapter II-7)
 - Volume name for the first file volume (FDVTP of the FDP)
 - One Volume Definition Packet (VDP) for each volume (Use macro call BLDVDP; see Chapter II-7 and Figure II-6-2)
 - Volume Table (i.e., concatenate the VDPs)
2. Put the address of the FDP in AC2.
3. Issue a Pre-open (.PINFOS) system call.
4. When INFOS returns:
 - Make sure that the address of the FDP is in AC2.
 - Issue an Open (.OINFOS) system call.
5. When INFOS returns, move the file's pseudo channel number from AC1 to your program area.

The difference between the opening procedures for the various modes lies in the contents of the FDPs and the VDPs. So, when opening your SAM file, follow the above procedures, and set up the FDPs and VDPs according to the appropriate charts following.

**FDP for Disk Files Opened in the Create Update or Output Modes
(See Table II-6-1)**

You must specify the following:

1. Either Create Update mode (for all functions), or Output (for writing only) (F1CRU or F1OUT of FDFL1)
2. SAM access method (F1SAM of FDFL1)
3. A record format; either Fixed, Variable, Undefined, or Data Sensitive (in FDFL1)
4. A) Exact length of Fixed length records (if used) (FDLEN)
B) Expected maximum length of Variable length records (if used) (FDLEN)
C) Pointer to delimiter table for Data Sensitive records (if used) (FDDSD)
5. Address of the Volume Table (FDVTP)

You may specify the following:

System default values for these options:

- | | |
|--|---|
| 1. Block size (FDBLK) | 512 bytes |
| 2. Number of buffers for this opening (FDBUF) | One buffer, recorded as a changeable part of the file's PFS |
| 3. Rewrite option (Create Update mode only) (F1RER of FDFL1) | You may not modify records at this open |
| 4. Read-After-Write verification (F1RAW of FDFL1) | No verification for this opening |
| 5. Exclusive file use (F1EXF of FDFL1) | Others may use this file while you do |

**VDP for Disk Files Opened in the Create Update or Output Modes
(See Table II-6-2)**

You must specify the address of each volume's name (VDVNP)

You may specify the following:

System default values for these options:

- | | |
|---|--|
| 1. Volume size (VDVSZ) | 65,535 blocks (Make sure that you have at least this much available disk space if you choose the default.) |
| 2. Contiguous allocation (ICCTG of VDIVC) | Random allocation |
| 3. Disable file initialization (ICDFI of VDIVC) | If contiguous allocation, each block is null filled |
| 4. Pad character (VDPAD) | Null (binary 0) |
| 5. Device timeout intervals (VDDTO) | 3 seconds for fixed-head disks;
5 seconds for moving head disks |

FDP for SAM Disk Files Opened in the Input or Update Modes
(See Table II-6-1)

You must specify the following:

1. Input mode (for reading only) or Update mode (for all functions) (F1INP or F1UPD of FDFL1)
2. SAM access method (F1SAM of FDFL1)
3. Same record format which you used at file creation (FDFL1)
4. Address of the Volume Table (FDVTP)

You may specify the following:

1. Number of buffers for this opening (FDBUF)
2. Read-After-Write verification (Update mode only) (F1RAW of FDFL1)
3. Exclusive file use (F1EXF of FDFL1)

System default values for these options:

- Number recorded in the file's PFS
- No verification for this opening
- Others may use this file while you do

VDP for SAM Disk Files Opened in the Input or Update Modes
(See Table II-6-2)

You must specify the address of the name of the first file volume (in VDVNP). The system will then use the contents of the PFS.

FDP for All Labeled Magnetic Tape Files
(See Table II-6-1)

You must specify the following:

1. SAM access method (F1SAM of FDFL1)
2. A record format: either Fixed, Variable, or Undefined (See FDFL1)
3. Output mode (for writing), or Input mode (for reading) (See FDFL1)
4. Address of the Volume Table (FDVTP)
5. Label type and level (See FDTCF)

You may specify the following:

1. Block size (FDBLK)
2. Number of buffers (FDBUF)
3. Record length (FDLEN)
4. HDR1 Label contents (FDFS1)

System default values for these options:

- 80 bytes
- One buffer
- Records will equal block size
- Volume identifier for the first volume of the file

VDP for All Labeled Magnetic Tape Files (See Table II-6-2)	
You must specify the following:	
1.	The address of the name of the first volume of the file (VDVNP)
2.	Pad character (VDPAD)
3.	Enable runtime initialization and release (if applicable) (VDIVC)
You may specify the following:	System default values for these options:
1.	Device timeout interval (VDDTO) 15 seconds
2.	Address of User Label Tables (VDVLT, VDHLT, VDTLT) No use of user labels

Table II-2-1. SAM Processing Functions

	Sequential Access Devices		Direct Access Devices			
	Input Mode	Output Mode	Input Mode	Output Mode	Update Mode	Create Update Mode
READ	X		X		X	X
WRITE		X		X	X	X
REWRITE*					X	X
RELEASE*					X	X

*Only valid if you specified Rewrite in the FDP

Processing SAM Files

You can issue processing requests as soon as you open your SAM file, but the functions available depend on the device on which the file resides and the processing mode you're using. If your file resides on a sequential access device, you can only read from the file in the Input mode or write to it in the Output mode. However, if your file resides on a direct access device, you can read, write, rewrite, or release your file, as illustrated in Table II-2-1. Note that you must provide a General Processing Packet for each processing request you make. You will find the packet format in Table II-6-3 and instructions for building one under BLDPP in Chapter II-7.

You can use either the .INFOS argument system call or the .INFOS 77 call to process your SAM file. Furthermore, no matter which processing function you use, the system will tell you if it encounters any of the following exception conditions:

- If you have reached the physical end-of-file,
- If a record being written is too large to fit into your specified block size,
- If the Read-After-Write cycle failed on a previous write or rewrite request,
- If the physical length of data transfer is less than your specified block size,
- If the length of the data you're transferring is longer than a particular sequential device can handle.

NOTE: A condition causing an exceptional status return may have occurred prior to the processing request in which the system returns the status.

Licensed Material - Property of Data General Corporation

In addition, for read and write operations, the system will always return the length of the data it transfers to your program area (in bytes), the block address of the record read, and the number of bytes by which the record is offset from the beginning of the block.

The following section lists everything you have to know about and do for each type of SAM processing request and the error messages which the system will return to you if required. (For further explanation of all INFOS error messages, see Appendix D.)

SAM Read Processing Request

1. Specify the address of your data area in PRDAT of the General Processing Packet (Table II-6-3).
2. Specify Lock (in PFLOC of PRSTA in the General Processing Packet) if you want to prevent other users from accessing the record you're reading. Note, however, that if you want to use Lock, you must also specify Rewrite in FDFL1 of the FDP when you open the file, and your file must reside on a direct access device.

Common READ Request Error Messages

Code	Description
200	ILLEGAL FUNCTION
201	VARIABLE LENGTH TRANSFER ILLEGAL ON THIS DEVICE
206	EXCLUSIVE FILE
207	FILE LOCKED
210	FILE NOT OPEN
211	PERIPHERAL CONFLICT
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
232	ILLEGAL CLOSE
243	RDOS OPEN ERROR
244	VOLUME ALREADY EXISTS
254	DEVICE NOT SUPPORTED
256	INPUT END VOLUME ERROR
415	ILLEGAL LABEL
416	ILLEGAL LABEL SPEC
417	VOL ID DOESNT MATCH
424	BLOCK COUNT INCORRECT
425	RECORD FORMAT CONFLICT
426	FILE SEQ NUMBER

SAM Write Processing Request

1. Specify the address of your data area in PRDAT of the General Processing Packet (Table II-6-3).
2. Specify (in PRLLEN) the length of any Data Sensitive, Undefined, or Variable length records which you are writing.

Common WRITE Request Error Messages

Code	Description
200	ILLEGAL FUNCTION
201	VARIABLE LENGTH TRANSFER ILLEGAL ON THIS DEVICE
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
232	ILLEGAL CLOSE
233	PHYSICAL I/O ERROR
255	OUTPUT END VOLUME ERROR
415	ILLEGAL LABEL
416	ILLEGAL LABEL SPEC
417	VOL ID DOESNT MATCH
423	EXP DATE NOT EXPIRED
424	BLOCK COUNT INCORRECT
425	RECORD FORMAT CONFLICT
426	FILE SEQ NUMBER

SAM Rewrite Processing Request

NOTE: You may request Rewrite only if you specified it in FDFL1 of the FDP when you opened this file and if your file resides on a direct access device.

1. Specify the address of your data area in PRDAT of the General Processing Packet (Table II-6-3).
2. Specify (in PRLLEN) the length of the record you are modifying. Note that you cannot change the length of a record with this request; the modified record must be exactly the same length as the record read.

Common REWRITE Request Error Messages

Code	Description
200	ILLEGAL FUNCTION
201	VARIABLE LENGTH TRANSFER ILLEGAL ON THIS DEVICE
203	ILLEGAL FUNCTION FOR DEV
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
234	RESIDUAL DISK ERROR

SAM Close and Force End of Volume Requests

There are no required specifications for these requests. However, if your file resides on magnetic tape, you may leave the tape positioned behind the file by specifying -1 in PRDSP of the General Processing Packet. If you do not specify it, the system will automatically rewind the tape.

Common CLOSE Request Error Messages

Code	Description
201	VARIABLE LENGTH TRANSFER ILLEGAL ON THIS DEVICE
210	FILE NOT OPEN
212	VL FILE PROCESSING ERROR
217	VIRTUAL MEMORY EXHAUSTED
221	VL FILE OPEN ERR
222	VL FILE CLOSE ERR
232	ILLEGAL CLOSE
234	RESIDUAL DISK ERROR
242	FILE CLOSE ERROR
255	OUTPUT END VOLUME ERROR
415	ILLEGAL LABEL
416	ILLEGAL LABEL SPEC
417	VOL ID DOESNT MATCH
424	BLOCK COUNT INCORRECT
425	RECORD FORMAT CONFLICT
426	FILE SEQ NUMBER

Common FORCE END-OF-VOLUME Request Error Messages

Code	Description
201	VARIABLE LENGTH TRANSFER ILLEGAL ON THIS DEVICE
210	FILE NOT OPEN
212	VL FILE PROCESSING ERROR
213	UNRESOLVED RESOURCE CONFLICT
217	VIRTUAL MEMORY EXHAUSTED
221	VL FILE OPEN ERR
222	VL FILE CLOSE ERR
224	LOGICAL END OF FILE
226	NO SUCH VOLUME
232	ILLEGAL CLOSE
242	FILE CLOSE ERROR
255	OUTPUT END VOLUME ERROR
256	INPUT END VOLUME ERROR
415	ILLEGAL LABEL
416	ILLEGAL LABEL SPEC
417	VOL ID DOESNT MATCH
424	BLOCK COUNT INCORRECT
425	RECORD FORMAT CONFLICT
426	FILE SEQ NUMBER

Licensed Material - Property of Data General Corporation
Magnetic Tape Control Request

NOTE: Do not mix these requests with other SAM processing requests for the same file. That is, to process a magnetic tape, you must either use all magnetic tape control requests, or all standard requests, Mixing the two may have undesirable effects on your file.

1. Build a Magnetic Tape Control Processing Packet (see Table II-6-11 and BLDMTC in Chapter II-7).
2. Specify in PRCFC what you want the system to do with your tape. Your options are:

Parameter Function

MCSFF	<i>Space forward to next tape mark and position behind it.</i>
MCSBF	<i>Space backward over two tape marks, then forward over one; position there.</i>
MCRD	<i>Read a record, starting at the present position. (Specify the record length in PRNWD.)</i>
MCWRT	<i>Write a record, starting at the present position. (Specify the record length in PRNWD.)</i>
MCWEF	<i>Write an end-of-file mark at the present position, then position directly behind it.</i>
MCREW	<i>Rewind the tape to the load point, then return to you.</i>
MCSFR	<i>Space forward n records (specified in PRNWD) or until encountering an end-of-file or end-of-tape mark.</i>
MCSBR	<i>Space backward n records (specified in PRNWD) until encountering a tape mark or the beginning of the tape.</i>
MCERS	<i>Erase a two and one-half inch strip of tape, beginning at the current position.</i>

3. If you are transferring records, specify the address of your data area in PRDAT.
4. If the system encounters an I/O error while processing your tape, it will set PFMTR in PRSTA. (Leave PRSTA blank when initiating your request.) It will also return the contents of the mag tape status registers in PRCFC; these flags are described in the *Programmer's Reference Manual - Peripherals* (number 015-021).

For an error during a Read or Write, the system will return the actual transfer length in PRNWD; for a Space Forward or Backward error, PRNWD will contain the number of records spaced over.

Common MAGNETIC TAPE CONTROL Request Error Messages

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
253	MAG-TAPE I/O ERROR

Point Request

NOTE: You may only issue this request when your file resides on a direct-access device.

1. Specify the Point mode in PRMOD of the Point Processing Packet (see Table II-6-8).
2. Specify the desired record's block record in PRHLB.
3. Specify how far the desired record is offset from the beginning of its block (in PRBOF).

Common POINT Request Error Messages

Code	Description
200	ILLEGAL FUNCTION
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
224	LOGICAL END OF FILE
232	ILLEGAL CLOSE
255	OUTPUT END VOLUME ERROR
256	INPUT END VOLUME ERROR

End of Chapter

Chapter 3

Random Access Method (RAM) Files

This chapter provides procedure-oriented descriptions of how to open and process a RAM file. Refer to Chapter 3 in Section One of this manual for further elaboration on the points discussed here.

How to Open a RAM File

You can open a RAM file in four different ways:

- In the Output mode (only writing allowed)
- In the Create Update mode (all processing functions allowed)
- In the Input mode (only reading allowed)
- In the Update mode (all processing functions allowed)

Depending on what you want to do with your file after you open it, you can specify either the Output or the Create Update mode when you create the file. To open an existing file, you can specify either the Input or Update mode. However, the procedure you follow to open the file will be the same regardless of the mode which you choose. Only the contents of the FDP and the VDPs will vary according to the mode. The steps for opening a file and the contents of the FDP and VDPs for each processing mode follow.

Steps in Opening a RAM File

1. Set up the following:
 - FDP (see Table II-6-1 and BLDFDP in Chapter II-7)
 - Volume names for each file volume (in FDVTP of the FDP)
 - One VDP for each file volume (see Table II-6-2 and BLDVDP in Chapter II-7)
 - A Volume Table (i.e., concatenate the VDPs)
2. Put the address of the FDP in AC2.
3. Issue a Pre-open (.PINFOS) system call.
4. When INFOS returns:
 - Make sure that the address of the FDP is in AC2.
 - Issue an Open (.OINFOS) system call.
5. When INFOS returns, move the file's pseudo channel number from AC1 to your program area.

Figure II-3-1 shows how the FDP and the Volume Table relate.

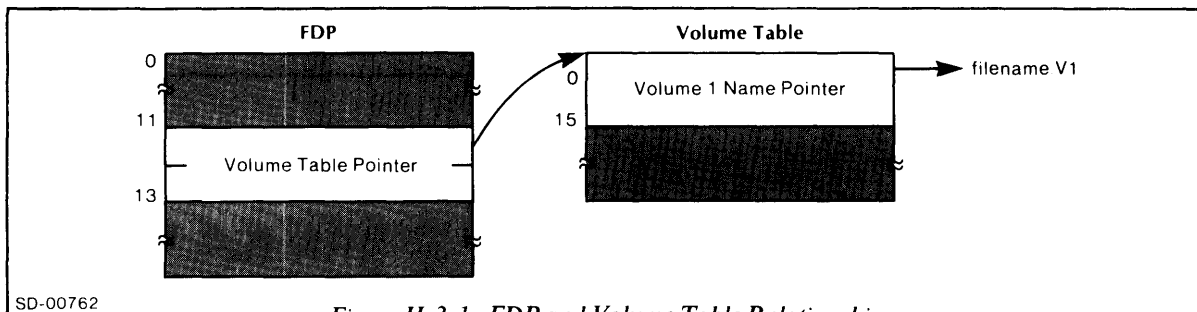


Figure II-3-1. FDP and Volume Table Relationship

FDP for RAM Files Opened in the Output or Create Update Mode
(See Table II-6-1)

You must specify the following:

1. RAM access method (in FDFL1)
2. Fixed length records (in FDFL1)
3. Either Output mode (for writing only) or Create Update mode (for all processing functions) (in FDFL1)
4. The starting address of the Volume Table (FDVTP)

You may specify the following:

System default values for these options:

- | | |
|---|--|
| 1. Number of file volumes (FDNVD) | One volume |
| 2. Code translation (in FDTCF) | No translation done |
| 3. Block size (FDBLK) | 512 bytes |
| 4. Record length (FDLEN) | Records are same length as block size |
| 5. Number of buffers (FDBUF) | One buffer |
| 6. Read-After-Write verification (in FDFL1) | No verification for this opening |
| 7. Set Exclusive Use (in FDFL1) | Others may access this file while you do |

VDP for RAM Files Opened in the Output or Create Update Mode
(See Table II-6-2)

You must specify the following:

1. The address of the name of the first file volume (VDVNP)
2. A pad character (VDPAD)

You may specify the following:

System default values for these options:

- | | |
|-------------------------------------|--|
| 1. Volume size (VDVSZ) | 65,535 blocks (Make sure that you have at least this much available disk space if you choose the default.) |
| 2. Contiguous allocation (in VDIVC) | Random allocation |
| 3. Device timeout interval (VDDTO) | 5 seconds for moving-head disks;
3 seconds for fixed-head disks |

FDP for RAM Files Opened in the Input or Update Mode
(See Table II-6-1)

You *must* specify the following:

1. RAM access method (in FDFL1)
2. Fixed length records (in FDFL1)
3. Either the Input mode (for reading only) or the Update mode (for all processing functions) (in FDFL1)
4. The address of the Volume Table (FDVTP)

You *may* specify the following:

System default values for these options:

- | | |
|--|---------------------------------------|
| 1. Number of buffers (FDBUF) | Number recorded in the PFS |
| 2. Read-After-Write verification (Update mode only) (in FDFL1) | No verification for this opening |
| 3. Set Exclusive Use (in FDFL1) | Others may use this file while you do |
| 4. Number of file volumes (FDNVD) | Only one file volume recognized |

VDP for RAM Files Opened in the Input or Update Mode

You must specify the address of the name of the first volume in the file in VDVNP. The system will use the contents of the PFS for the remaining information.

Processing Your RAM File

You can issue any one of the following processing requests as soon as you open your RAM file:

- Read
- Write
- Close
- Force end-of-volume
- Set Exclusive Use
- Preread

For read and write requests, the system will always return the length (in bytes) of the record transferred or written, the block address of the record, and the number of bytes by which the record is offset from the beginning of the block. In addition, the system will return the following exceptional status information for all processing requests:

- If it encounters a physical end-of-file while processing,
- If the read-after-write verification cycle failed twice on a previous request.

NOTE: You will receive exceptional status returns after the fact. That is, the condition for which you get the return will have occurred in a previous request.

The following section lists the procedures for each type of RAM processing request and the error messages the system may return to you. (For further explanation of all INFOS error messages, see Appendix D.)

To build a General Processing Packet, see BLDPP in Chapter II-7.

RAM Read Request

1. Specify the address of your data area in PRDAT of the General Processing Packet (Table II-6-3).
2. Specify the number of the record desired in PPREC.
3. A. Specify Lock if you want to prevent access by other users while you are reading the record by setting PFLOC in PRSTA.
B. If the record you specify is already locked by another user, you may set PFHLD in PRSTA to specify that your request be held in a queue until the record becomes available.

Common READ Request Error Messages

Code	Description
200	ILLEGAL FUNCTION
206	EXCLUSIVE FILE
207	FILE LOCKED
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
227	NO HOLD ON LOCKED REQUEST
231	RAM ACCESS OUTSIDE FILE
243	RDOS OPEN ERROR
244	VOLUME ALREADY EXISTS
254	DEVICE NOT SUPPORTED

RAM Write Request

1. Specify the address of your data area in PRDAT of the General Processing Packet (Table II-6-3).
2. Specify the number of the record you want to write in PPREC.
3. Specify Lock to prevent access by other users while you are processing the record by setting PFLOC in PRSTA.
4. Specify Unlock to make the record available to other users when this write request is completed by setting PFUNL in PRSTA.
5. Specify Write Immediate to write the record to your file as soon as possible, by setting PFWIF in PRSTA.
6. Specify Read Inhibit if you don't want the system to read the other records in the block into a buffer by setting PFRIN in PRSTA.

Common WRITE Request Error Message

Code	Description
200	ILLEGAL FUNCTION
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
227	NO HOLD ON LOCKED REQUEST
230	NO MORE DISK SPACE
231	RAM ACCESS OUTSIDE FILE
234	RESIDUAL DISK ERROR
255	OUTPUT END VOLUME ERROR

RAM Close Request

There are no required or optional parameters for a close request. Simply supply a blank General Processing Packet so the system will have a place to return exceptional status information, if there is any.

Common CLOSE Request Error Messages

Code	Description
210	FILE NOT OPEN
212	VL FILE PROCESSING ERROR
213	UNRESOLVED RESOURCE CONFLICT
217	VIRTUAL MEMORY EXHAUSTED
221	VL FILE OPEN ERR
222	VL FILE CLOSE ERR
231	RAM ACCESS OUTSIDE FILE
234	RESIDUAL DISK ERROR
242	FILE CLOSE ERROR
255	OUTPUT END VOLUME ERROR
256	INPUT END VOLUME ERROR

RAM Force End-of-Volume Request

1. Specify the number of the volume you wish to close in PRDSP of the General Processing Packet. Note that the first volume of any RAM file is always numbered zero, and the system recognizes the other volume numbers by the sequence in which you enter their VDPs into the Volume Table.

Common FORCE END-OF-VOLUME Request Error Messages

Code	Description
210	FILE NOT OPEN
212	VL FILE PROCESSING ERROR
213	UNRESOLVED RESOURCE CONFLICT
217	VIRTUAL MEMORY EXHAUSTED
221	VL FILE OPEN ERR
222	VL FILE CLOSE ERR
224	LOGICAL END OF FILE
226	NO SUCH VOLUME
231	RAM ACCESS OUTSIDE FILE
232	ILLEGAL CLOSE
242	FILE CLOSE ERROR
255	OUTPUT END VOLUME ERROR
256	INPUT END VOLUME ERROR

RAM Set Exclusive Use Request

Note that you can gain exclusive access to your file either by setting F1EXF in FDFL1 of the FDP, or during file processing, by setting PFLOC in PRSTA of the General Processing Packet. If no one else is using the file, you may use the file exclusively until you release it or close it. If, however, another user has opened the file prior to your request, you will receive error code number 206: EXCLUSIVE FILE.

RAM Preread Request

1. Specify the number of the record you want in PRREC of the General Processing Packet (Table II-6-3).
2. Specify Read Inhibit to inhibit the transfer of the block containing the record to an I/O buffer by setting PFRIN in PRSTA. (This is handy when your next request will be a write; it speeds up your processing operations.)

Common PREREAD Request Error Messages

Code	Description
200	ILLEGAL FUNCTION
206	EXCLUSIVE FILE
207	FILE LOCKED
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
227	NO HOLD ON LOCKED REQUEST
231	RAM ACCESS OUTSIDE FILE
243	RDOS OPEN ERROR
244	VOLUME ALREADY EXISTS
254	DEVICE NOT SUPPORTED

End of Chapter

Chapter 4

Indexed Sequential Access Method (ISAM) Files

This chapter will tell you how to open an ISAM file under various conditions and how to use the different processing requests. This section is procedure-oriented, so please refer to Chapter 4 in Section One if you have any questions about the options explained here.

How to Open ISAM Files

You can open your ISAM file in any one of the following ways:

- In the Create Update mode; or
- In the Update mode *without* a Database runtime File Definition Packet; or
- In the Update mode *with* a Database runtime FDP. (This is handy when you want to use a different number of buffers than you recorded in the Permanent File Specification.)

Steps for Opening in the Create Update Mode

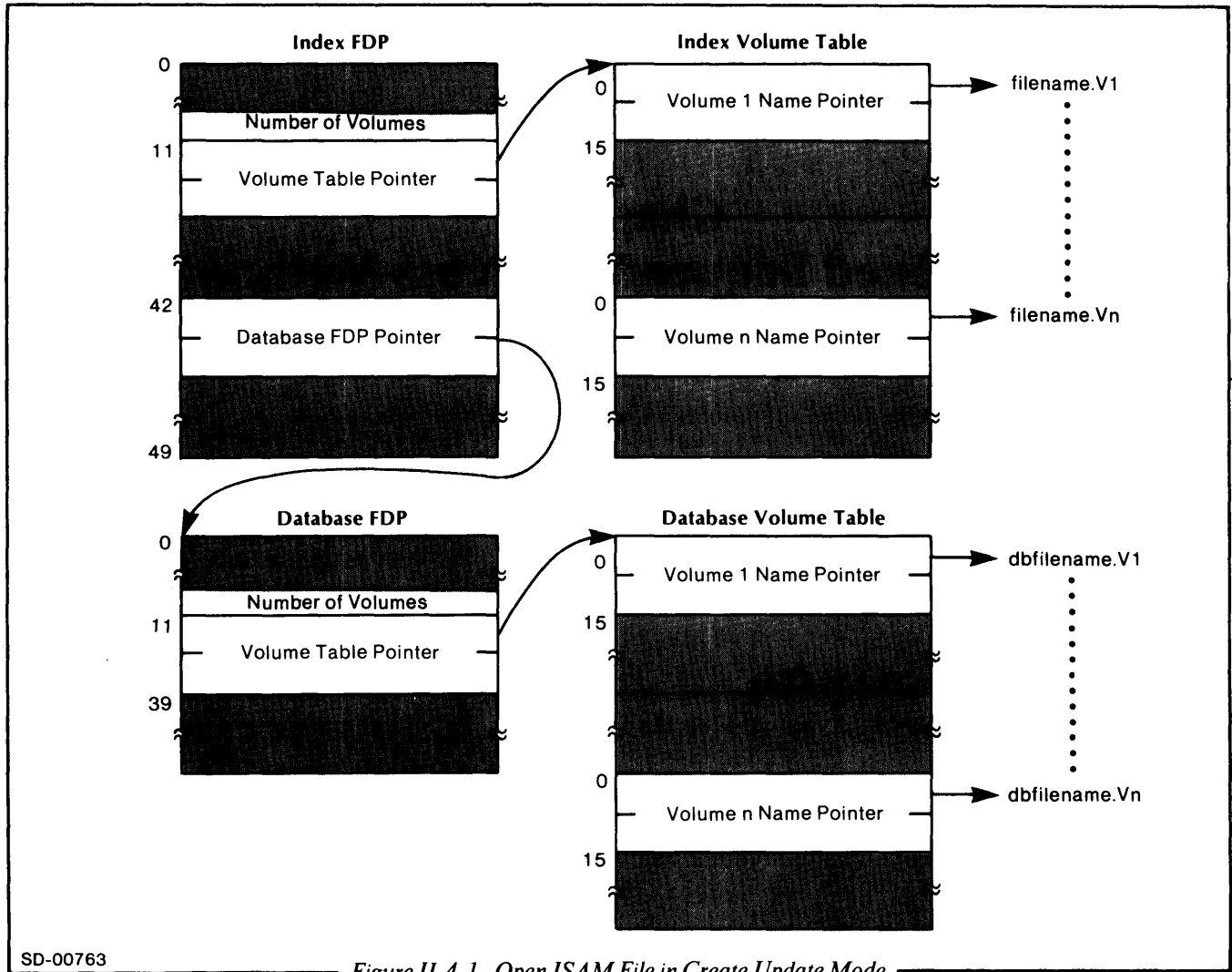
1. Set up the following:
 - A. For the Index:
 - Index FDP (see Tables II-4-1, II-6-1, and BLDFDP in Chapter II-7)
 - Volume names for each Index volume (FDVTP of the FDP)
 - One VDP for each Index volume (see Tables II-4-3, II-6-2, and BLDVDP in Chapter II-7)
 - An Index Volume Table (i.e., concatenate the Index VDPs)

B. For the Database:

- Database FDP (Tables II-4-2, II-6-1, and BLDFDP in Chapter II-7)
- Volume names for each Database volume (FDVTP of each FDP)
- One VDP for each Database volume (Tables II-4-3, II-6-2, and BLDVDP in Chapter II-7)
- A Database Volume Table

2. Put the address of the Index FDP in AC2.
3. Issue a Pre-open (.PINFOS) system call.
4. When INFOS returns:
 - A. Make sure that the address of the Index FDP is in AC2.
 - B. Issue an Open (.OINFOS) system call.
5. When INFOS returns, move the file's pseudo channel number from AC1 to your program area.

Figure II-4-1 shows how the packets and tables relate.



SD-00763

Figure II-4-1. Open ISAM File in Create Update Mode

**Table II-4-1. ISAM Index FDP for Create Update Mode
(See Table II-6-1)**

You must specify the following:

1. ISAM access method (in FDFL1)
2. Variable length records (in FDFL1)
3. Create Update processing mode (in FDFL1)
4. Subindexing not allowed (in FDIFL)
5. The address of the Volume Table (FDVTP)
6. The address of the database FDP (FDDBP)
7. Initial node size (FDINS)
8. Maximum key length (FDMKL)

You may specify the following:

System default values for these options are:

- | | | |
|----|--|--|
| 1. | The number of I/O buffers for this opening (FDBUF) | 2 buffers, recorded as a changeable part of the file's PFS |
| 2. | Page size (FDBLK) | 512 bytes |
| 3. | Number of volumes (FDNVD) | 1 volume |
| 4. | Read-After-Write verification (in FDFL1) | No verification for this opening |
| 5. | Space management (in FDFL2) | No space management |
| 6. | Disable hierarchical replacement (in FDFL2) | Hierarchically modulated LRU buffer management used for this opening |

**Table II-4-2. ISAM Database FDP for Create Update Mode
(See Table II-6-1)**

You must specify the following:	
1. RAM access method (in FDFL1)	
2. Variable length records (in FDFL1)	
3. Create Update processing mode (in FDFL1)	
4. The address of the Volume Table (FDVTP)	
You may specify the following:	System default values for these options are:
1. Page size (FDBLK)	512 bytes
2. Number of I/O buffers for this opening (FDBUF)	1 buffer, recorded as a changeable part of the file's PFS
3. Number of volumes (FDNVD)	1 volume
4. Read-After-Write verification (in FDFL1)	No verification for this opening
5. Space management (in FDFL2)	No space management

**Table II-4-3. VDPs for ISAM Index and Database Files Opened in the Create Update Mode
(See Table II-6-2)**

You must specify the address of each volume's name (VDVNP)	
You may specify the following:	System default values for these options are:
1. Volume size (VDVSZ)	65,535 blocks if random allocation; if contiguous allocation is specified, you must also specify the number of blocks to be allocated.
2. Contiguous allocation (in VDIVC)	Random allocation
3. Disable file initialization (in VDIVC)	If contiguous allocation, each block is null filled
4. Pad character (VDPAD)	Null (binary 0)

Licensed Material - Property of Data General Corporation

Steps for Opening in the Update Mode without a Database Runtime FDP

1. Set up the following:

A. For the Index:

- Index FDP (Tables II-4-4, II-6-1, and BLDFDP in Chapter II-7)
- Index volume name (FDVTP of the FDP)
- A VDP for the Index volume (Tables II-4-5, II-6-2, and BLDVDP in Chapter II-7)
- Index volume table

B. For the Database:

- The Database filename (FDDBP of the index's FDP)

2. Put the address of the Index FDP in AC2.

3. Issue a Pre-open (.PINFOS) system call.

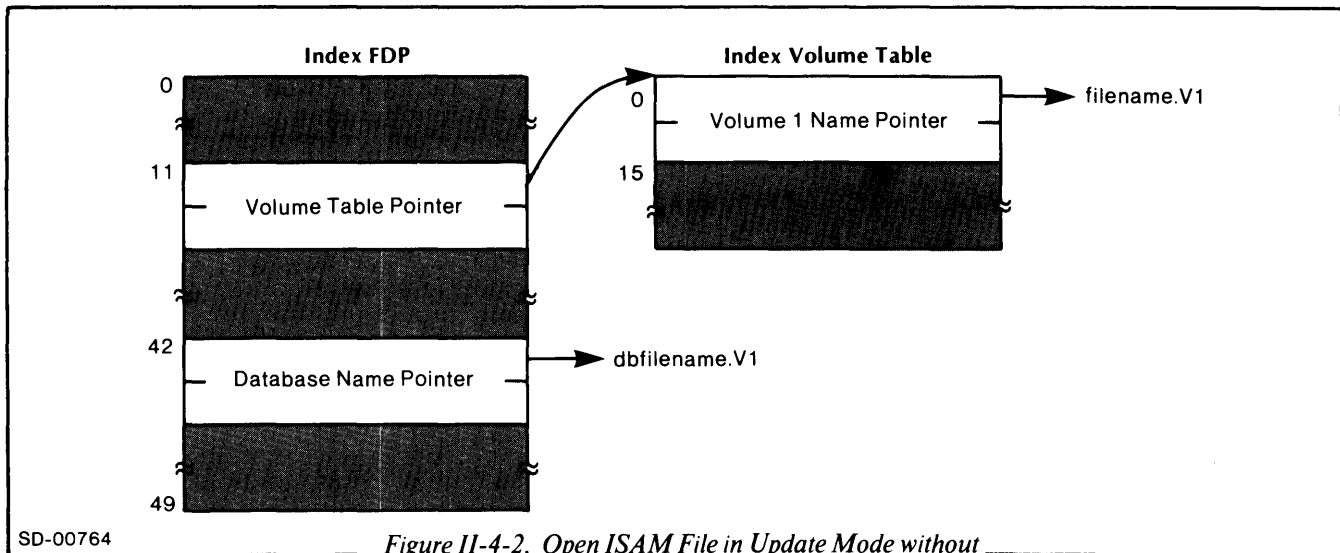
4. When INFOS returns:

A. Make sure that the address of the Index FDP is in AC2.

B. Issue an Open (.OINFOS) system call.

5. When INFOS returns, move the file's pseudo channel number from AC1 to your program area.

Figure II-4-2 shows how the packets and tables relate.



SD-00764

Figure II-4-2. Open ISAM File in Update Mode without Database Runtime FDP

Table II-4-4. ISAM Index FDP for Update Mode without Database Runtime FDP
(See Table II-6-1)

You *must* specify the following:

1. ISAM access method (in FDFL1)
2. Variable length records (in FDFL1)
3. Update processing mode (in FDFL1)
4. The address of the Volume Table (FDVTP)
5. The address of the database file name (FDDBP)

You *may* specify the following:

Default values for these options are:

- | | | |
|----|--|--|
| 1. | The number of I/O buffers for this opening (FDBUF) | Contents of the index and database PFSs are used |
| 2. | Read-After-Write verification (in FDFL1) | No verification for this opening |
| 3. | Disable hierarchical replacement (in FDFL2) | Hierarchically modulated LRU buffer management used for this opening |

Table II-4-5. VDP for ISAM Index Opened in the Update Mode

You *must* specify the address of the name of the first volume of the index in FDVTP

The contents of the .VL files for the index and database are used in the Update processing mode.

Licensed Material - Property of Data General Corporation

Steps for Opening in the Update Mode with a Database Runtime FDP

1. Set up the following:

A. For the Index:

- Index FDP (Tables II-4-6, II-6-1, and BLDFDP in Chapter II-7)
- Index volume name (FDVTP of the FDP)
- Index VDP (Tables II-4-8, II-6-2, and BLDVDP in Chapter II-7)
- Index Volume Table

B. For the Database:

- Database FDP (Tables II-4-7, II-6-1, and BLDFDP)
- Database volume name (FDVTP of the FDP)
- Database VDP (Tables II-4-8, II-6-2, and BLDVDP)
- Database Volume Table

2. Put address of Index FDP in AC2.

3. Issue a Pre-open (.PINFOS) system call.

4. When INFOS returns:

A. Make sure that the address of the Index FDP is in AC2.

B. Issue an Open (.OINFOS) system call.

5. When INFOS returns, move the file's pseudo channel number from AC1 to your program area.

Figure II-4-3 shows how the packets and tables relate.

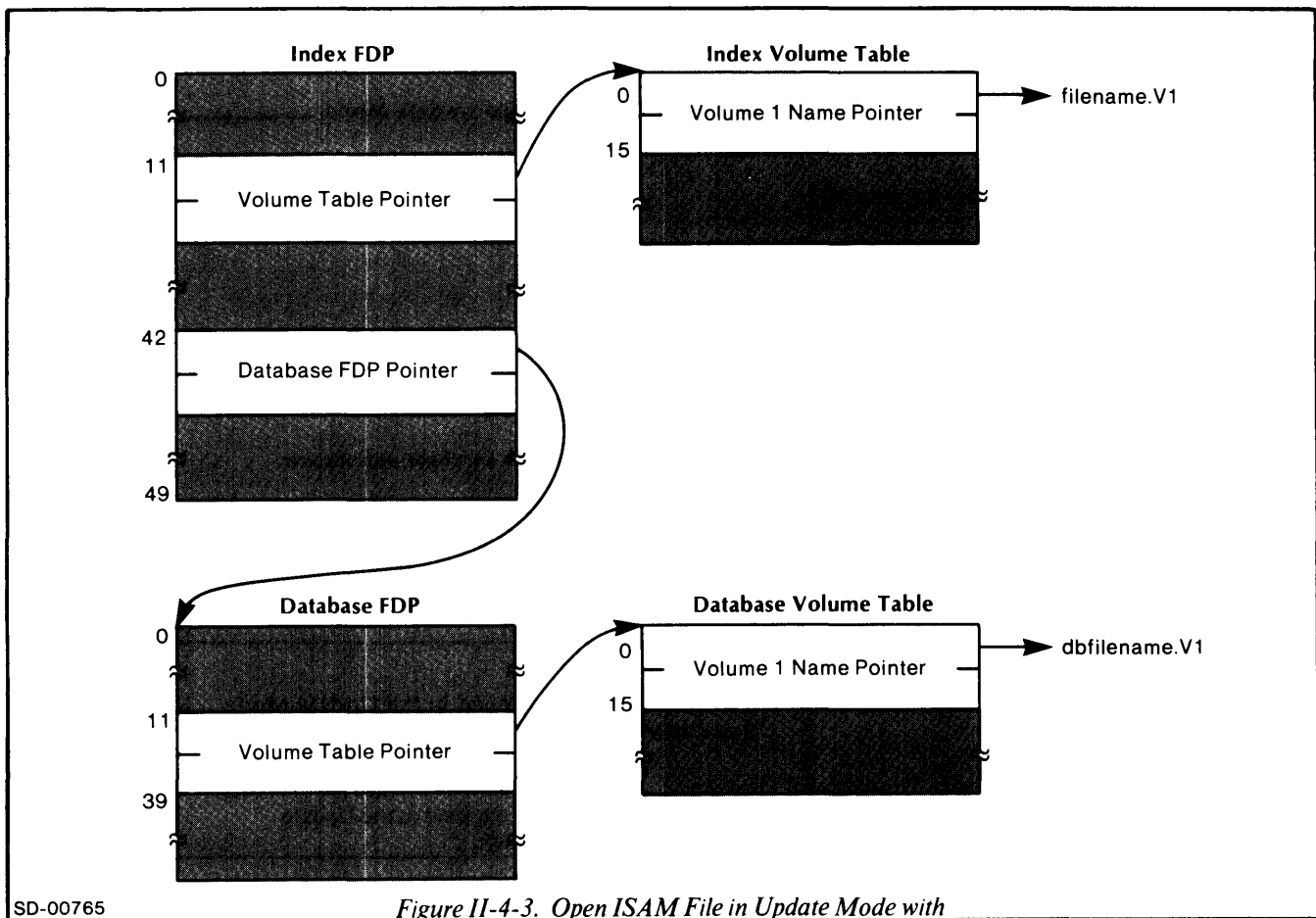


Figure II-4-3. Open ISAM File in Update Mode with Database Runtime FDP.

**Table II-4-6. ISAM Index FDP for Update Mode with Database Runtime FDP
(See Table II-6-1)**

You must specify the following:

1. ISAM access method (in FDFL1)
2. Variable length records (in FDFL1)
3. Update processing mode (in FDFL1)
4. The address of the Volume Table (FDVTP)
5. That a database FDP is present (in FDFL2)
6. The address of the database FDP (FDDBP)

You may specify the following:

Default values for these options are:

- | | |
|---|--|
| <ol style="list-style-type: none"> 1. The number of I/O buffers to be used for this opening (FDBUF) 2. Read-After-Write verification (in FDFL1) 3. Disable hierarchical replacement (in FDFL2) | <p>The contents of the Index PFS are used</p> <p>No verification for this opening</p> <p>Hierarchically modulated LRU buffer management is used for this opening</p> |
|---|--|

**Table II-4-7. ISAM Database Runtime FDP for Update Mode
(See Table II-6-1)**

You must specify the following:

1. RAM access method (in FDFL1)
2. Variable length records (in FDFL1)
3. Update processing mode (in FDFL1)
4. The address of the Volume Table (in FDVTP)

You may specify the following:

Default values for these options are:

- | | |
|---|--|
| <ol style="list-style-type: none"> 1. The number of I/O buffers to be used for this opening (FDBUF) 2. Read-After-Write verification (in FDFL1) | <p>The contents of the data base PFS are used for this opening</p> <p>No verification for this opening</p> |
|---|--|

**Table II-4-8. VDPs for ISAM Index and Database Opened in the Update Mode
(See Table II-6-2)**

You must specify the address of the name of the first volume of each file in VDVNP

The contents of the .VL files for the index and database are used in the Update processing mode.

Processing ISAM Files

You can issue processing requests as soon as you open your ISAM file. For each request you issue, however, you must supply an Extended Processing Packet (see Table II-6-4 and BLDPP in Chapter II-7), and specify either Keyed or Relative access in PRCCW of that packet.

Keyed Access

Whenever you want to use keyed access, you must provide a Key Table. An ISAM Key Table consists of a single Key Definition Packet, followed immediately by either a zero length key or a null word, as shown in Figure II-4-4.

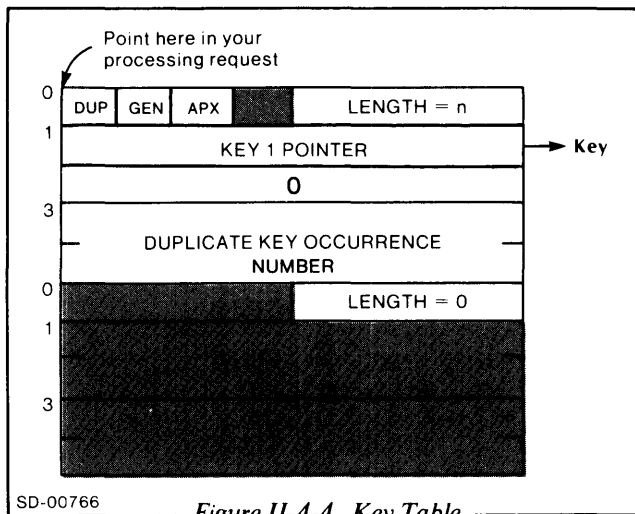


Figure II-4-4. Key Table

Relative Access

As you might think, you don't need a key table to do Relative Access, since you won't be using keys. With each Relative processing request, you need only specify a direction of motion relative to the last established current position. In ISAM, you can move forward, backward, up, down, down and forward, or not at all (static). The available ISAM processing requests are as follows:

- Read, Write, Rewrite
- Delete, Delete Subindex, Reinstate
- Retrieve Status, Retrieve Key, Retrieve High Key
- Close

Note that for all of the following processing requests, the system will let you know:

- if it encounters a physical end-of-file while processing, or
- if the Read-After-Write verification cycle failed on a previous write or rewrite request.

Refer to the macro call descriptions in Chapter II-7 to build the indicated packets:

Packet	Macro Call
Extended Processing	BLDPP
Key Definition	BLDKDP
Subindex Definition	BLDSDP

ISAM Read Processing Request

1. Specify an access technique in PRCCW of the Extended Processing Packet (Table II-6-4).
 - A. For Keyed Access, you must build a Key Table (i.e., a Key Definition Packet) with one key and put a pointer to it in PRKTP.
 - B. For Relative Access, you must specify a direction of motion in PRCCW.

On each access into the Index, the system returns this information:

- Whether the key accessed is a duplicate,
 - Whether the key accessed has been logically deleted,
 - The length of the key accessed.
2. If you want to read a record by using an approximate or generic key, specify the desired option in KDTYP of the Key Definition Packet.
 3. If you want to establish a new current position, specify Set Current Position in PRCCW.

If you don't set a new current position, upon the successful completion of this request, the INFOS system returns to the last established current position.

4. If you want the data record returned, specify the address of your data area in PRDAT.

If you don't want the data record returned, specify Suppress Database in PRCCW.

5. You may specify the length of the record you want returned (in PRLen). The system will let you know if the actual record is longer than your request. If the system does return a record, it will let you know its length, its address, and whether it has been logically deleted.
6. If you want to lock or unlock a record, specify:

Local lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC in PRCCW),

or

Global lock or unlock (PFLOC/PFUNL in PRSTA and CCGLB in PRCCW),

or

Both local and global lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC and CCGLB in PRCCW).

Common Error Messages for a Read Request

Code	Description
200	ILLEGAL FUNCTION
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
231	RAM ACCESS OUTSIDE FILE
243	RDOS OPEN ERROR
244	VOLUME ALREADY EXISTS
254	DEVICE NOT SUPPORTED
257	COMPARE ERROR (ISAM)
260	RESOLUTION ERROR (ISAM)
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
266	SUBINDEX NOT DEFINED
267	END OF SUBINDEX
274	ILLEGAL COMMAND CONTROL
276	KEY POSITIONING ERROR
400	DATA BASE REC NOT PRESENT
403	DATA RECORD LOCKED
413	INDEX ENTRY LOCKED

ISAM Write Processing Request

1. You must specify Keyed Access in PRCCW of the Extended Processing Packet (Table II-6-4), build a Key Table with one key (i.e., a Key Definition Packet), and put a pointer to the Key Table in PRKTP of the EPP.
2. You may Set Current Position if you want to establish a new one by setting CCSCP in PRCCW. If you don't, the system will return you to the last established current position when it successfully completes this request.
3. You must specify the address of your data area in PRDAT and the length of the record to be written in PRLN if you want to link the index entry you are creating to a data record. If you don't want to link the new index entry to a data record, specify Suppress Database in PRCCW.
4. You may specify Write Immediate in PRSTA if you want the new index entry and data record written to the file as soon as this request is completed. If you do not specify Write Immediate, the INFOS system will not transfer the contents of the buffer used for this request until it needs the buffer for a subsequent request.
5. You may lock or unlock a record by specifying:

Local lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC in PRCCW),

or

Global lock or unlock (PFLOC/PFUNL in PRSTA and CCGLB in PRCCW),

or

Both local and global lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC and CCGLB in PRCCW).

Common Error Messages for a Write Request

Code	Description
200	ILLEGAL FUNCTION
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
230	NO MORE DISK SPACE
231	RAM ACCESS OUTSIDE FILE
234	RESIDUAL DISK ERROR
252	NO NODE SPACE
255	OUTPUT END VOLUME ERROR
257	COMPARE ERROR (ISAM)
260	RESOLUTION ERROR
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
266	SUBINDEX NOT DEFINED
272	ILLEGAL KEY LENGTH
273	INVALID ENTRY NUMBER
274	ILLEGAL COMMAND CONTROL
275	KEY ALREADY EXISTS
276	KEY POSITIONING ERROR
277	INVALID RECORD LENGTH
414	NO WRITE WITHOUT KEY

ISAM Rewrite Processing Request

1. You must specify an access technique in PRCCW of the Extended Processing Packet (Table II-6-4).
 - A. For Keyed access, you must build a Key Table with one key (i.e., a Key Definition Packet) and put a pointer to it in PRKTP.
 - B. For Relative access, you must specify a direction of motion in PRCCW.
2. Specify Set Current Position in PRCCW if you want to establish a new one. If you don't, the system will return to the last established current position.
3.
 - A. If you are rewriting an *existing* data record, specify the address of your data area in PRDAT and the length of the record you're now writing in PRLN.
 - B. If you are writing a *new* data record, specify the address of your data area in PRDAT and the length of the record you're writing in PRLN.
4. If you want to lock or unlock a record for rewriting, specify:

Local lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC in PRCCW),

or

Global lock or unlock (PFLOC/PFUNL in PRSTA and CCGLB in PRCCW),

or

Both local and global lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC and CCGLB in PRCCW).

Common Error Messages for a Rewrite Request

Code	Description
200	ILLEGAL FUNCTION
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
231	ACCESS OUTSIDE FILE
234	RESIDUAL DISK ERROR
252	NO NODE SPACE
260	RESOLUTION ERROR
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
266	SUBINDEX NOT DEFINED
267	END OF SUBINDEX
274	ILLEGAL COMMAND CONTROL
276	KEY POSITIONING ERROR
400	DATA BASE REC NOT PRESENT
403	DATA RECORD LOCKED
413	INDEX ENTRY LOCKED

ISAM Delete Processing Request

NOTE: This request allows you to either physically delete an index entry and its corresponding data record, or simply mark either one as logically deleted. However, to *physically* delete an index entry and its associated data record, you must use *keyed* access. Thus:

1. Specify an access technique in PRCCW of the Extended Processing Packet (Table II-6-4).
 - A. For Keyed access, you must build a Key Table with one key (i.e., a Key Definition Packet), and put a pointer to it in PRKTP.
 - B. For Relative access, you must specify a direction of motion in PRCCW.
2. A. To mark an *index entry* as logically deleted, specify *Local Logical Delete* (set CCLOC and CCLOG in PRCCW).
 - B. To mark a *data record* as logically deleted, specify *Global Logical Delete* (set CCGLB and CCLOG in PRCCW).
3. Specify Set Current Position in PRCCW if you want to establish a new one. If you don't, the system will return to the last established position when it completes this request.

For a Physical Delete, current position will either be on:

- A. The next higher index entry from the one deleted (if any), or
- B. The initial system-set current position.

Common Error Messages for a Delete Request

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
230	NO MORE DISK SPACE
231	ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
267	END OF SUBINDEX
270	DELETE POSITIONING ERROR
272	ILLEGAL KEY LENGTH
274	ILLEGAL COMMAND CONTROL
275	KEY ALREADY EXISTS
276	KEY POSITIONING ERROR
403	DATA RECORD LOCKED
411	SUBINDEX HAS SUBINDEX; DELETE SUBINDEX ERROR
412	ATTEMPT TO DELETE ENTRY WITHOUT KEYED ACCESS
413	INDEX ENTRY LOCKED

ISAM Delete Subindex Processing Request

1. You must specify *upward* relative motion in PRCCW of the Extended Processing Packet.

Common Error Messages for a Delete Subindex Request

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
230	NO MORE DISK SPACE
231	RAM ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
265	SUBINDICES NOT ALLOWED
266	SUBINDEX NOT DEFINED
267	END OF SUBINDEX
274	ILLEGAL COMMAND CONTROL
276	KEY POSITIONING ERROR

ISAM Reinstate Processing Request

1. Specify an access technique in PRCCW of the Extended Processing Packet (Table II-6-4).
 - A. For Keyed access, you must build a Key Table with one key (i.e., a Key Definition Packet) and put a pointer to it in PRKTP.
 - B. For Relative access, you must specify a direction of motion in PRCCW.
2. A. To reinstate an index entry, specify Local (CCLOC in PRCCW).
 - B. To reinstate a data record, specify Global (CCGLB in PRCCW).
 - C. To reinstate both a data record and an index entry, specify both Local and Global in a single request (i.e., set both CCLOC and CCGLB in PRCCW).

Common Error Messages for a Reinstate Request

Code	Description
213	UNRESOLVED SOURCE CONFLICT
230	NO MORE DISK SPACE
231	ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
272	ILLEGAL KEY LENGTH
274	ILLEGAL COMMAND CONTROL
275	KEY ALREADY EXISTS
276	KEY POSITIONING ERROR
400	DATA BASE REC NOT PRESENT
403	DATA RECORD LOCKED

ISAM Retrieve Status Processing Request

1. Specify an access technique in PRCCW of the Extended Processing Packet (Table II-6-4).
 - A. For Keyed access, you must build a Key Table with one key (i.e., a Key Definition Packet), and put a pointer to it in PRKTP.
 - B. For Relative access, you must specify a direction of motion in PRCCW.
2. Specify Set Current Position in PRCCW if you want to establish a new one. If you don't, the system will return to the last established position when it successfully completes this task.
3. The INFOS system will return the following status:
 - A. The record length, whether or not it's marked as logically deleted, and whether it's locked.
 - B. The key length, whether it's marked as logically deleted, and its occurrence number if it's a duplicate.

Common Error Messages for a Retrieve Status Request

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
231	RAM ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
267	END OF SUBINDEX
274	ILLEGAL COMMAND CONTROL
276	KEY POSITIONING ERROR
400	DATA BASE REC NOT PRESENT
403	DATA RECORD LOCKED

ISAM Retrieve Key and Retrieve High Key Processing Requests

1. Specify the address of the area to which the system is to return the key in PRDAT and provide a Key Definition Packet to receive a duplicate key occurrence number (if any). If you didn't allow duplicate keys, place -1 in the second word of the Key Table pointer.
2. Specify an access technique in PRCCW.
 - A. For Keyed access, you must build a Key Table with one key (i.e., a Key Definition Packet), and put a pointer to it in PRKTP.
 - B. For Relative access, you must specify a direction of motion in PRCCW.
3. Specify Set Current Position in PRCCW if you want to establish a new one, otherwise the system will return to the last established current position.
4. The system will return the key, its length, and its occurrence number if it's a duplicate.

Common Error Messages for Retrieve Key and Retrieve High Key Requests

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
231	ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
267	END OF SUBINDEX
274	ILLEGAL COMMAND CONTROL
276	KEY POSITIONING ERROR

End of Chapter

Chapter 5

Data Base Access Method (DBAM) Files

This chapter discusses how to open a DBAM file in various circumstances and how to use the DBAM processing requests. Similar to the rest of Section Two, this chapter is procedure-oriented. You can find further details about the information described here in Chapter 5 of Section One of this manual.

Opening DBAM Files

You can open a DBAM file in five different ways:

- In the Create Update mode
- In the Update mode *without* a Database runtime FDP
- In the Update mode *with* a Database runtime FDP
- When you're creating a new DBAM Index *without* a Database runtime FDP
- When you're creating a new DBAM Index *with* a Database runtime FDP

Let's look at each of these situations individually.

Steps for Opening in the Create Update Mode

1. Set up the following:

A. For the Index:

- Index FDP (see Tables II-5-1, II-6-1, and BLDFDP in Chapter II-7)
- Volume names for each Index volume (FDVTP of the FDP)
- One VDP for each Index volume (Tables II-5-3, II-6-2, and BLDVDP in Chapter II-7)
- An Index Volume Table (i.e., concatenate the Index VDPs)

B. For the Database:

- Database FDP (Tables II-5-2, II-6-1, and BLDFDP)
- Volume names for each Database volume (FDVTP of the FDP)
- One VDP for each Database volume (Tables II-5-3, II-6-2, and BLDVDP)
- A Database Volume Table

2. Put the address of the Index FDP in AC2.

3. Issue a Pre-open (.PINFOS) system call.

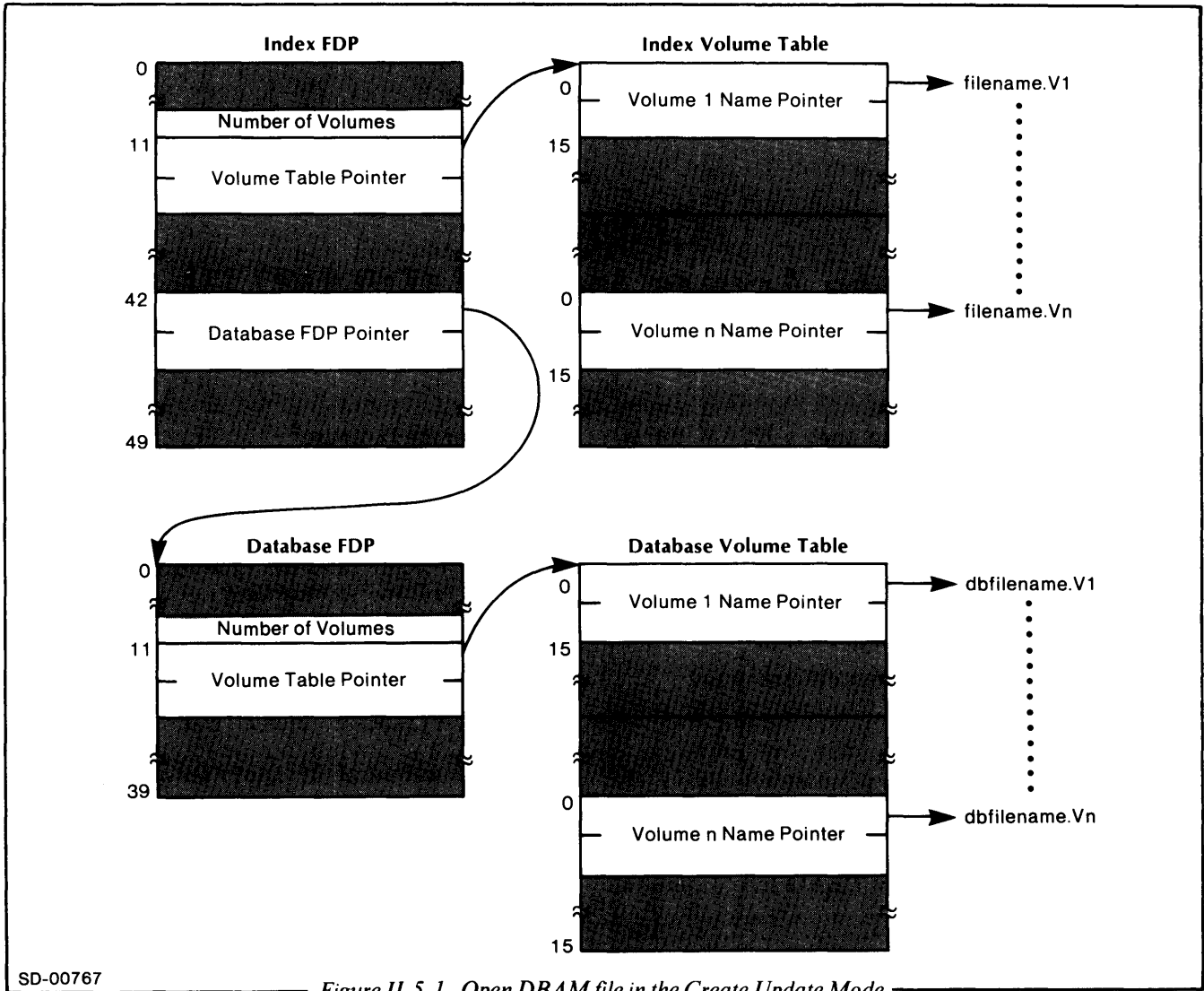
4. When INFOS returns:

A. Make sure that the address of the Index FDP is in AC2.

B. Issue an Open (.OINFOS) system call.

5. When INFOS returns, move the file's pseudo channel number from AC1 to your program area.

Figure II-5-1 shows how the packets and tables relate.



SD-00767

Figure II-5-1. Open DBAM file in the Create Update Mode

Table II-5-1. DBAM Index FDP for Create Update Mode
(See Table II-6-1)

You *must* specify the following:

1. DBAM access method (in FDFL1)
2. Variable length records (in FDFL1)
3. Create Update processing mode (in FDFL1)
4. The number of index levels (FDNIL)
5. The address of the Volume Table (FDVTP)
6. The address of the Database FDP (FDDBP)
7. Initial node size (FDINS)
8. Maximum key length (FDMKL)

You *may* specify the following:

System default values for these options are:

- | | | |
|-----|--|---|
| 1. | The number of I/O buffers for this opening (FDBUF) | 2 buffers, recorded as a changeable part of the file's PFS |
| 2. | Page size (FDBLK) | 512 bytes |
| 3. | Number of volumes (FDNVD) | 1 volume |
| 4. | Partial record length (FDPRL) | No partial record (0 length) |
| 5. | Root node merit factor (FDRMF) | Merit factor of 0 |
| 6. | Read-After-Write verification (in FDFL1) | No verification for this opening |
| 7. | Space management (in FDFL2) | No space management |
| 8. | Optimized distribution (in FDFL2) | No optimized distribution |
| 9. | Disable hierarchical replacement (in FDFL2) | Hierarchically modulated LRU buffer management used for this opening |
| 10. | Key compression (in FDIFL) | Keys in this subindex not compressed |
| 11. | High priority node (in FDIFL) | Root node priority based on its subindex level |
| 12. | Temporary subindex (in FDIFL) | Permanent subindex |
| 13. | Permanent data records (in FDIFL) | Data records linked to index entries in this subindex can be physically deleted |

**Table II-5-2. DBAM Database FDP for Create Update Mode
(See Table II-6-1)**

You must specify the following:	
1. RAM access method (in FDFL1)	
2. Variable length records (in FDFL1)	
3. Create Update processing mode (in FDFL1)	
4. The address of the Volume Table (FDVTP)	
You may specify the following:	System default values for these options are:
1. Page size (FDBLK)	512 bytes
2. Number of I/O buffers for this opening (FDBUF)	1 buffer, recorded as a changeable part of the file's PFS
3. Number of volumes (FDNVD)	1 volume
4. Read-After-Write verification. (in FDFL1)	No verification for this opening
5. Space management (in FDFL2)	No space management
6. Optimized distribution (in FDFL2)	No optimized distribution

Table II-5-3. VDPs for Index and Database Files Opened in the Create Update Mode (See Table II-6-2)

You must specify the address of each volume's name in VDVNP.	
You may specify the following:	System default values for these options are:
1. Volume size (VDVSZ)	65,535 blocks (Make sure that you have at least this much available disk space if you choose the default.)
2. Contiguous allocation (in VDIVC)	Random allocation
3. Disable file initialization (in VDIVC)	If contiguous allocation, each block is null filled
4. Volume merit factor (VDVMF)	0 merit factor
5. Pad character (VDPAD)	Null (binary 0)

Licensed Material - Property of Data General Corporation

Steps for Opening in the Update Mode without a Database Runtime FDP

1. Set up the following:
 - A. For the Index:
 - Index FDP (Tables II-5-4, II-6-1, and BLDFDP in Chapter II-7)
 - Index volume name (FDVTP of the FDP)
 - A VDP for the Index volume (Tables II-5-5, II-6-2, and BLDVDP in Chapter II-7)
 - Index volume table
 - B. For the Database:
 - The Database filename (FDDBP of the index's FDP)
2. Put the address of the Index FDP in AC2.
3. Issue a Pre-open (.PINFOS) system call.
4. When INFOS returns:
 - A. Make sure that the address of the Index FDP is in AC2.
 - B. Issue an Open (.OINFOS) system call.
5. When INFOS returns, move the file's pseudo channel number from AC1 to your program area.

Figure II-5-2 shows how the packets and tables relate.

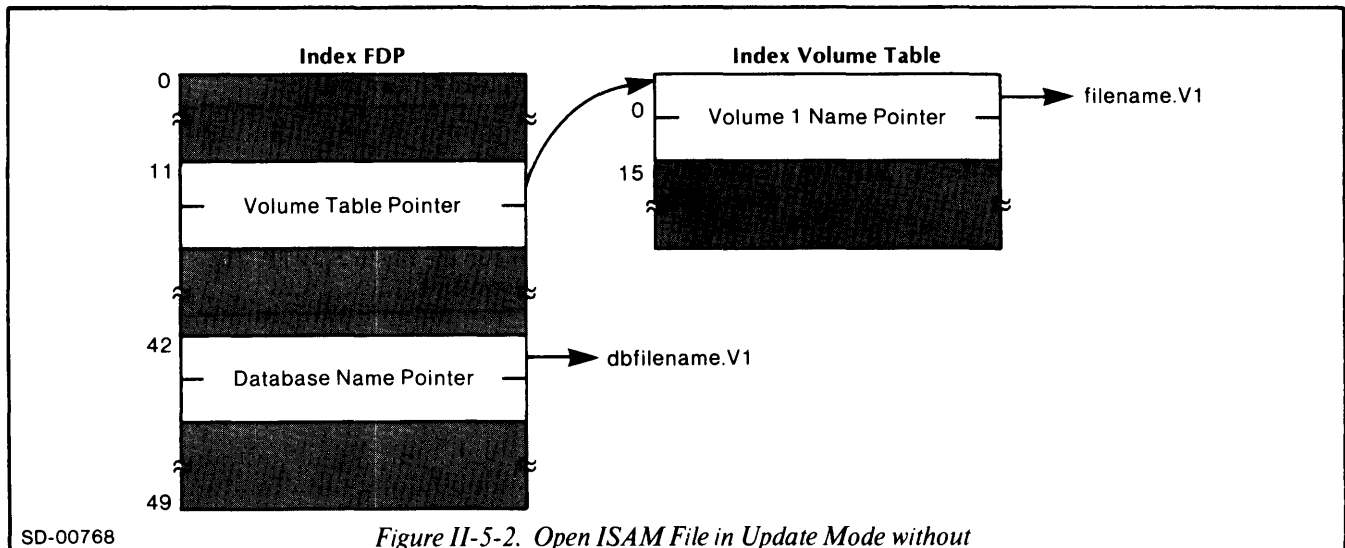


Figure II-5-2. Open ISAM File in Update Mode without Database Runtime FDP

**Table II-5-4. DBAM Index FDP for Update Mode without Database Runtime FDP
(See Table II-6-1)**

You must specify the following:

1. DBAM access method (in FDFL1)
2. Variable length records (in FDFL1)
3. Update processing mode (in FDFL1)
4. The address of the Volume Table (FDVTP)
5. The address of the database filename (FDDBP)

You may specify the following:

Default values for these options are:

- | | | |
|----|--|--|
| 1. | The number of I/O buffers for this opening (FDBUF) | Contents of index and database PFSs are used |
| 2. | Read-After-Write verification (in FDFL1) | No verification for this opening |
| 3. | Disable hierarchical replacement (in FDFL2) | Hierarchically modulated LRU buffer management used for this opening |

**Table II-5-5. VDP for DBAM Index Opened in the Update Mode
(See Table II-6-2)**

You must specify the address of the name of the first volume in the file in FDVTP. The system will use the contents of the PFS for the remaining information.

Licensed Material - Property of Data General Corporation

Steps for Opening in the Update Mode with a Database Runtime FDP

1. Set up the following:
 - A. For the Index:
 - Index FDP (Tables II-5-6, II-6-1, and BLDFDP in Chapter II-7)
 - Index volume name (FDVTP of the FDP)
 - Index VDP (Tables II-5-8, II-6-2, and BLDVDP in Chapter II-7)
 - Index Volume Table
 - B. For the Database:
 - Database FDP (Tables II-5-7, II-6-1, and BLDFDP)
 - Database volume name (FDVTP of the FDP)
 - Database VDP (Tables II-5-8, II-6-2, and BLDVDP)
 - Database Volume Table
2. Put address of Index FDP in AC2.
3. Issue a Pre-open (.PINFOS) system call.
4. When INFOS returns:
 - A. Make sure that the address of the Index FDP is in AC2.
 - B. Issue an Open (.OINFOS) system call.
5. When INFOS returns, move the file's pseudo channel number from AC1 to your program area.

Figure II-5-3 shows how the packets and tables relate.

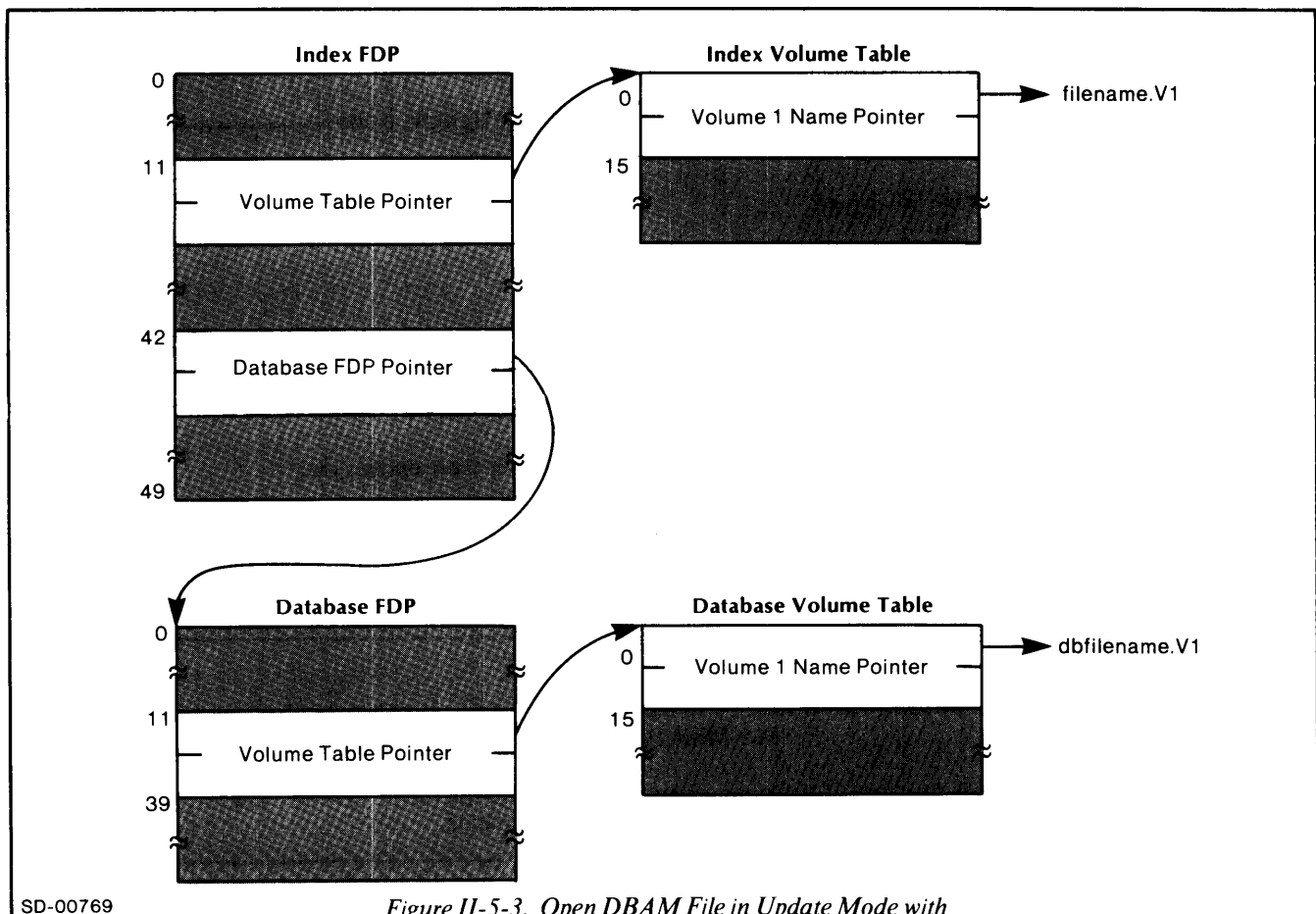


Figure II-5-3. Open DBAM File in Update Mode with Database Runtime FDP

Table II-5-6. DBAM Index FDP for Update Mode with Database Runtime FDP
(See Table II-6-1)

You must specify the following:	
1. DBAM access method (in FDFL1)	
2. Variable length records (in FDFL1)	
3. Update processing mode (in FDFL1)	
4. The address of the Volume Table (FDVTP)	
5. That a database FDP is present (in FDFL2)	
6. The address of the database FDP (FDDBP)	
You may specify the following:	Default values for these options are:
1. The number of I/O buffers for this opening (FDBUF)	The contents of the index PFS are used
2. Read-After-Write verification (in FDFL1)	No verification for this opening
3. Disable hierarchical replacement (in FDFL2)	Hierarchically modulated LRU buffer management is used for this opening

Table II-5-7. DBAM Database Runtime FDP for Update Mode
(See Table II-6-1)

You must specify the following:	
1. RAM access method (in FDFL1)	
2. Variable length records (in FDFL1)	
3. Update processing mode (in FDFL1)	
4. The address of the Volume Table (FDVTP)	
You may specify the following:	Default values for these options are:
1. The number of I/O buffers to be used for this opening (FDBUF)	The contents of the database PFS are used for this opening
2. Read-After-Write verification (in FDFL1)	No verification for this opening

Table II-5-8. VDPs for DBAM Index and Database Opened in the Update Mode
(See Table II-6-2)

You must specify the address of the name of the first volume in the file in VDVNP. The system will use the contents of the PFS for the remaining information.

Licensed Material - Property of Data General Corporation

Steps for Creating a New DBAM Index without a Database Runtime FDP

1. Set up the following:

A. For the Index:

- New Index FDP (Tables II-5-9, II-6-1, and BLDFDP in Chapter II-7)
- New Index volume names (FDVTP of the FDP)
- One VDP for each new Index volume (Tables II-5-10, II-6-2, and BLDVDP in Chapter II-7)
- New Index Volume Table

B. For the Database:

- Database filename (FDDBP of the index's FDP)

2. Put the address of the new Index FDP in AC2.

3. Issue a Pre-open (.PINFOS) system call.

4. When INFOS returns:

A. Make sure that the address of the new Index FDP is in AC2.

B. Issue an Open (.OINFOS) system call.

5. When INFOS returns, move the file's pseudo channel number from AC1 to your program area.

Figure II-5-4 shows how the packets and tables relate.

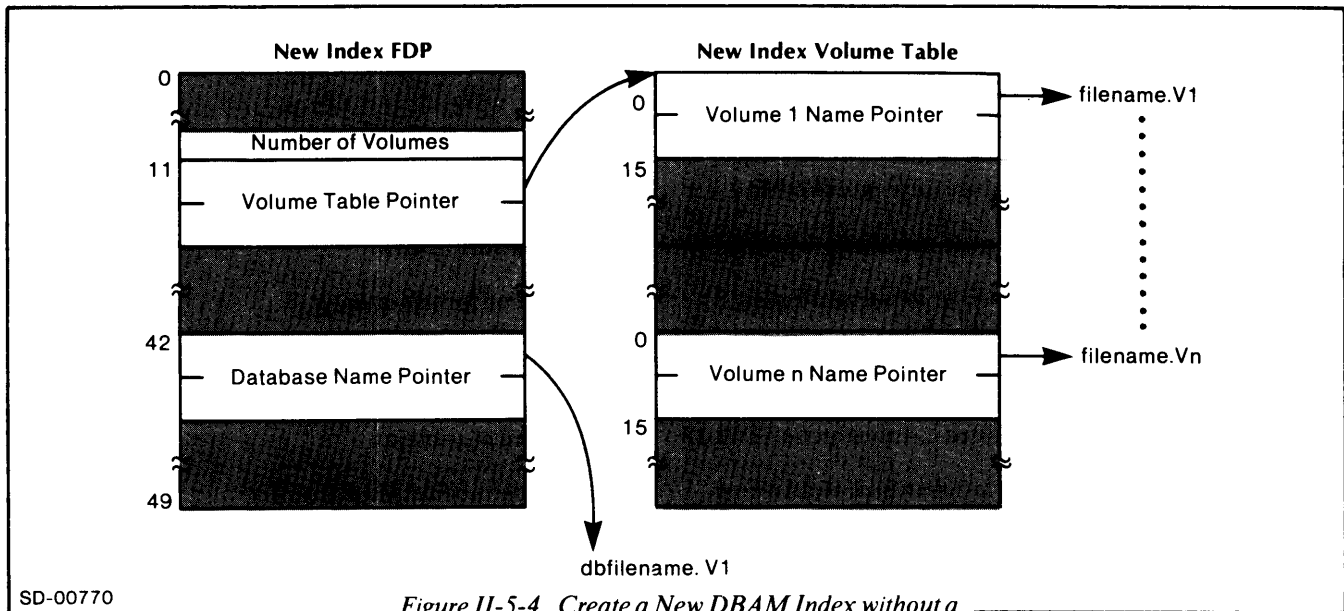


Figure II-5-4. Create a New DBAM Index without a Database Runtime FDP

SD-00770

**Table II-5-9. FDP for New DBAM Index without Database Runtime FDP
(See Table II-6-1)**

You must specify the following:

1. DBAM access method (in FDFL1)
2. Variable length records (in FDFL1)
3. Create Update processing mode (in FDFL1)
4. That you are inverting a file (in FDFL1)
5. The number of index levels (FDNIL)
6. The address of the Volume Table (FDVTP)
7. The address of the database filename (FDDBP)
8. The initial node size (FDINS)
9. The maximum key length (FDMKL)

You may specify the following:

System default values for these options are:

- | | | |
|-----|--|---|
| 1. | The number of I/O buffers for this opening (FDBUF) | 2 buffers, recorded as a changeable part of the file's PFS |
| 2. | Page size (FDBLK) | 512 bytes |
| 3. | Number of volumes (FDNVD) | 1 volume |
| 4. | Partial record length (FDPRL) | No partial records (0 length) |
| 5. | Root node merit factor (FDRMF) | No optimized distribution |
| 6. | Read-After-Write verification (in FDFL1) | No verification for this opening |
| 7. | Space management (in FDFL2) | No space management |
| 8. | Optimized distribution (in FDFL2) | No optimized distribution |
| 9. | Disable hierarchical replacement. (in FDFL2) | Hierarchically modulated LRU buffer management used for this opening |
| 10. | Key compression (in FDIFL) | Keys in this subindex not compressed |
| 11. | High priority node (in FDIFL) | Root node priority based on its subindex level |
| 12. | Temporary subindex (in FDIFL) | Permanent subindex |
| 13. | Permanent data records (in FDIFL) | Data records linked to index entries in this subindex can be physically deleted |

**Table II-5-10. VDP for New DBAM Index
(See Table II-6-2)**

You must specify the address of each volume's name in VDVNP.

You may specify the following:

System default values for these options are:

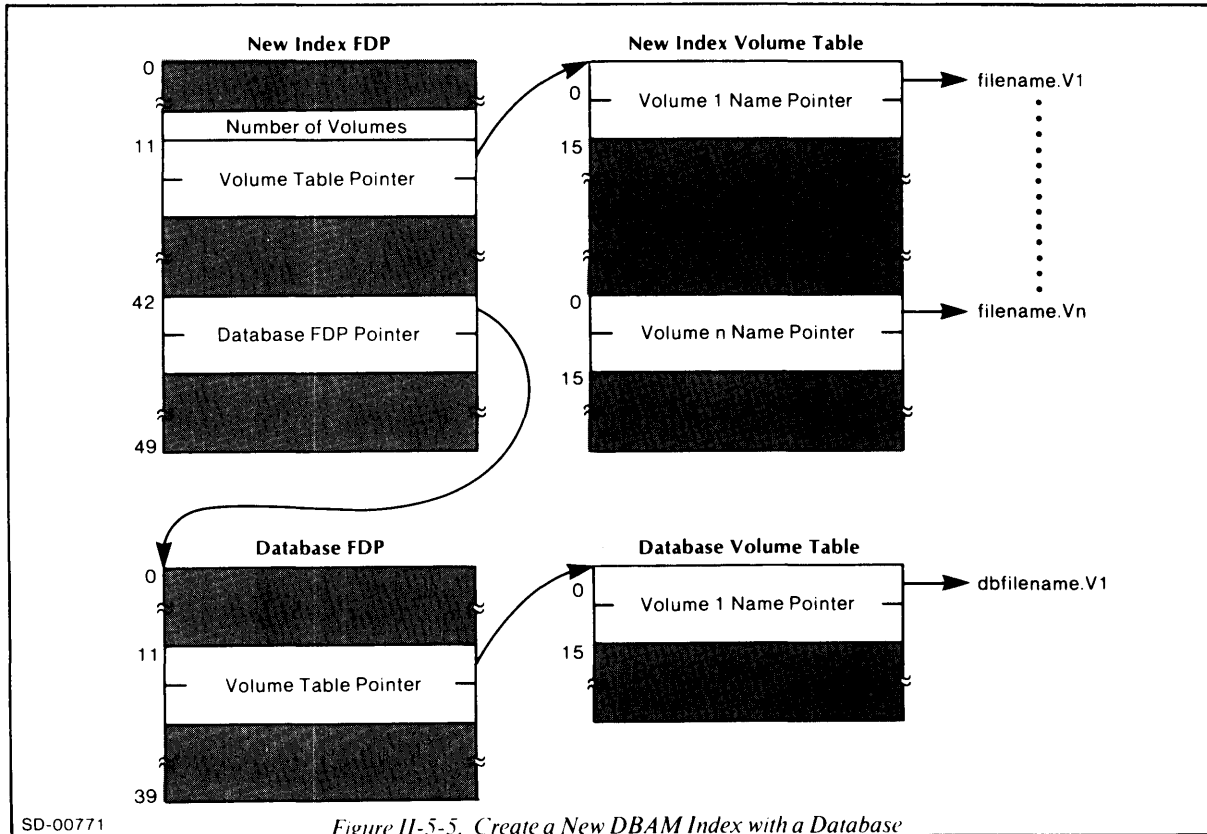
- | | | |
|----|--|--|
| 1. | Volume size (VDVSZ) | 65,535 blocks (Make sure that you have at least this much available disk space if you choose the default.) |
| 2. | Contiguous allocation (in VDIVC) | Random allocation |
| 3. | Disable file initialization (in VDIVC) | If contiguous allocation, each block is null filled |
| 4. | Volume merit factor (VDVMF) | 0 merit factor |
| 5. | Pad character (VDPAD) | Null (binary 0) |

Steps for Creating a New DBAM Index with a Database Runtime FDP

1. Set up the following:
 - A. For the Index:
 - New Index FDP (Tables II-5-11, II-6-1, and BLDFDP in Chapter II-7)
 - New Index volume names (FDVTP of the FDP)
 - One VDP for each new Index volume (Tables II-5-12, II-6-2, BLDVDP in Chapter II-7)
 - New Index Volume Table
 - B. For the Database:
 - New Database runtime FDP for the new Index (Tables II-5-13, II-6-1 and BLDFDP)
 - Database volume name (FDVTP of the FDP)
 - Database VDP (Tables II-5-14, II-6-2, and BLDVDP)
 - Database Volume Table

2. Put the address of the new Index FDP in AC2.
3. Issue a Pre-open (.PINFOS) system call.
4. When INFOS returns:
 - A. Make sure that the address of the new Index FDP is in AC2.
 - B. Issue an Open (.OINFOS) system call.
5. When INFOS returns, move the file's pseudo channel number from AC1 to your program area.

Figure II-5-5 shows how the packets and tables relate.



SD-00771

Figure II-5-5. Create a New DBAM Index with a Database Runtime FDP

**Table II-5-11. FDP for New DBAM Index with Database Runtime FDP
(See Table II-6-1)**

You must specify the following:

1. DBAM access method (in FDFL1)
2. Variable length records (in FDFL1)
3. Create Update processing mode (in FDFL1)
4. That you are inverting a file (in FDFL1)
5. The number of index levels (FDNIL)
6. The address of the Volume Table (FDVTP)
7. That a database FDP is present (in FDFL2)
8. The address of the database FDP (FDDBP)
9. The initial node size (FDINS)
10. The maximum key length (FDMKL)

You may specify the following:

System default values for these options are:

- | | | |
|-----|--|---|
| 1. | The number of I/O buffers for this opening (FDBUF) | 2 buffers, recorded as a changeable part of the file's PFS |
| 2. | Page size (FDBLK) | 512 bytes |
| 3. | Number of volumes (FDNVD) | 1 volume |
| 4. | Partial record length (FDPRL) | No partial records (0 length) |
| 5. | Root node merit factor (FDRMF) | No optimized distribution |
| 6. | Read-After-Write verification (in FDFL1) | No verification for this opening |
| 7. | Space management (in FDFL2) | No space management |
| 8. | Optimized distribution (in FDFL2) | No optimized distribution |
| 9. | Disable hierarchical replacement (in FDFL2) | Hierarchically modulated LRU buffer management used for this opening |
| 10. | Key compression (in FDIFL) | Keys in this subindex not compressed |
| 11. | High priority node (in FDIFL) | Root node priority based on its subindex level |
| 12. | Temporary subindex (in FDIFL) | Permanent subindex |
| 13. | Permanent data records (in FDIFL) | Data records linked to index entries in this subindex can be physically deleted |

Table II-5-12. VDP for New DBAM Index
(See Table II-6-2)

You must specify the address of each volume's name in VDVNP.	
You may specify the following:	System default values for these options are:
1. Volume size (VDVSZ)	65,535 blocks (Make sure that you have at least this much available disk space if you choose the default.)
2. Contiguous allocation (in VDIVC)	Random allocation
3. Disable file initialization (in VDIVC)	If contiguous allocation, each block is null filled
4. Volume merit factor (VDVMF)	0 merit factor
5. Pad character (VDPAD)	Null (binary 0)

Table II-5-13. DBAM Database Runtime FDP for New Index
(See Table II-6-1)

You must specify the following:	
1. RAM access method (in FDFL1)	
2. Variable length records (in FDFL1)	
3. Update processing mode (in FDFL1)	
4. The address of the Volume Table (FDVTP)	
You may specify the following:	Default values for these options are:
1. The number of I/O buffers to be used for this opening (FDBUF)	Number recorded in the database PFS
2. Read-After-Write verification (in FDFL1)	No verification for this opening

Table II-5-14. VDP for DBAM Database Opened in the Update Mode
(See Table II-6-2)

You must specify the address of the name of the first volume in the file in VDVNP. The system will use the contents of the PFS for the remaining information.
--

Processing DBAM Files

You can issue DBAM processing requests as soon as your file is open. For each request you issue, however, you must build an Extended Processing Packet (see Table II-6-4 and BLDPP in Chapter II-7), and specify either Keyed, Relative, or combined Keyed and Relative access in PRCCW of that packet. Furthermore, note that, for all processing requests, the INFOS system will notify you:

- if it encounters a physical end-of-file while processing, or
- if the Read-After-Write verification cycle failed on a previous write or rewrite request.

Keyed Access

You must provide a Key Table for each Keyed access request. A Key Table for a DBAM request is merely a contiguous sequence of Key Definition Packets, and it contains one definition for each subindex level through which you want the INFOS system to pass. Each time you use Keyed access, you must specify (in the FDP) the address of the first word of your Key Table. (See Figure II-6-6 for further details on Key Tables.)

Relative Access

Obviously, you don't need a Key Table for Relative access because you're not using keys. In Relative access, motion is relative to the last established current position, and, in a DBAM file, you can move in any of eight unique directions: forward, backward, up, down, up and forward, down and forward, up and backward, and static (no move).

Combined Keyed and Relative Access

You can combine Keyed and Relative access in most DBAM processing requests. And, when you do, you must specify a direction of motion *and* provide a Key Table. Furthermore, you can only request upward, downward, or static movement. The system will perform the Relative movement first, then apply the contents of the Key Table from there. Therefore, you need fewer key definitions in your Key Table for a combined access processing request than you do for plain Keyed access. For example, in Figure II-5-6, imagine your current position is on "Payroll" in the subindex level one, and you want to read the record associated with the key "Zappa" in subindex level two. Just specify downward motion (to bring you down to level two,) and provide a key table with just one key definition (for "Zappa").

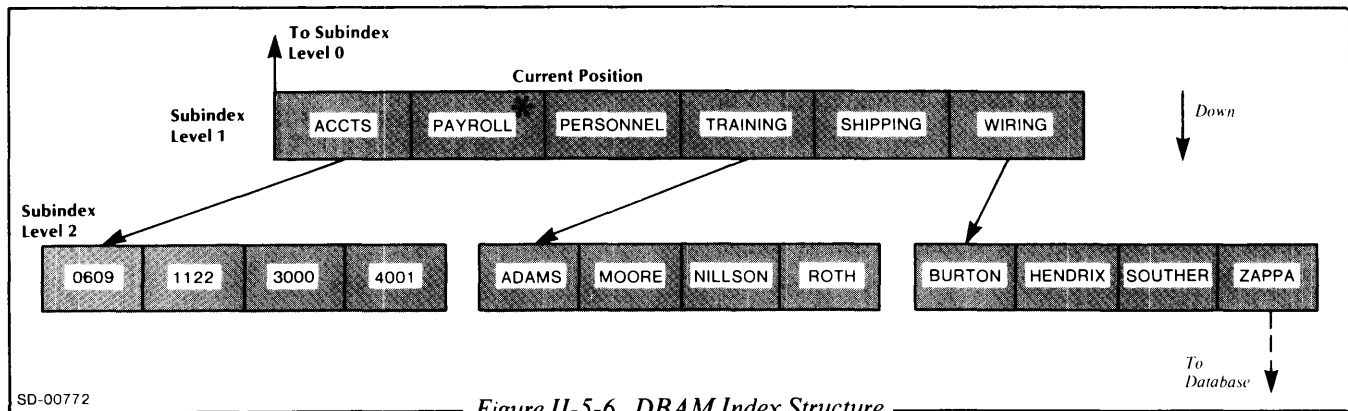


Figure II-5-6. DBAM Index Structure

DBAM Read Processing Request

1. Specify an access technique in PRCCW of the Extended Processing Packet (Table II-6-4).
 - A. For Keyed access, build a Key Table with one key for each subindex level to be accessed (see BLDKDP in Chapter II-7 to build Key Definition Packets) and put a pointer to its first word in PRKTP.
 - B. For Relative access, specify a direction of movement in PRCCW.
 - C. For combined access, provide a Key Table, a pointer to it in PRKTP, and a direction of motion in PRCCW.
2. If you want to read a record by an approximate, generic, or duplicate key, specify your desired option in KDTP of the Key Definition Packet. If you want to read a duplicate key, place its occurrence number in KDDKO.
3. Specify Set Current Position (in PRCCW) if you want to establish a new one; otherwise, the system will return to the last established current position.
4. Specify the address of your data area in PRDAT if you want the system to return the data record; otherwise specify Suppress Database in PRCCW.
5. Specify (in PRLen) the length of the record to be returned.

The system will tell you the record's length, its address, whether or not it has been logically deleted, and whether the actual record is longer than the length you specified.

6. Specify the address of your partial record area in PRPRA if you want the partial record returned; otherwise, specify Suppress Partial Record in PRCCW.

The system will tell you (in PRSRL) the length of any partial records returned.

7. If you want to lock or unlock, specify:

Local lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC in PRCCW),

or

Global lock or unlock (PFLOC/PFUNL in PRSTA and CCGLB in PRCCW),

or

Both local and global lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC and CCGLB in PRCCW).

8. On each access into the Index, the system will return:
 - A. the subindex level number of the key accessed (in PRSIL of the Processing Packet),
 - B. whether or not the key accessed is a duplicate (in PRSRS), and
 - C. whether or not the key accessed is logically deleted (in PRSRS).

Common Error Messages for a DBAM Read Request

Code	Description
200	ILLEGAL FUNCTION
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
231	RAM ACCESS OUTSIDE FILE
243	OPERATING SYSTEM OPEN ERROR
244	VOLUME ALREADY EXISTS
254	DEVICE NOT SUPPORTED
257	COMPARE ERROR (ISAM)
260	RESOLUTION ERROR (ISAM)
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
266	SUBINDEX NOT DEFINED
267	END OF SUBINDEX
274	ILLEGAL COMMAND CONTROL
276	KEY POSITIONING ERROR
400	DATA BASE REC NOT PRESENT
403	DATA RECORD LOCKED
413	INDEX ENTRY LOCKED

Licensed Material - Property of Data General Corporation

DBAM Write Processing Request

1. Specify an access technique in PRCCW of the Extended Processing Packet (Table II-6-4):
 - A. For Keyed access, provide a Key Table containing one key for each subindex level to be accessed and put a pointer to its first word in PRKTP. The system will write the last key in the Table to the appropriate subindex level, and it will return that level number to PRSIL of the Processing Packet.
 - B. For combined Keyed and Relative access, specify a direction of motion in PRCCW, provide a Key Table, and put its address in PRKTP.
 - C. If you're writing a duplicate key, specify that you're doing so by setting KTDUP in KDTYP of the Key Definition Packet. The system will assign an occurrence number for the duplicate and return that number in KDDKO.

You may *not* use simple Relative access to write to a DBAM file.

2. Specify Set Current Position in PRCCW if you want to establish a new one; otherwise the system will return you to the last established position.
3. A. Specify the address of your data area (in PRDAT) and the length of the record to be written (in PRLLEN) if you want to link your new Index entry to a *new* data record; otherwise, specify Suppress Database in PRCCW.
 - B. Specify the Record Merit Factor in PRRMF if you specified Optimized Distribution in FDFL2 of the Database FDP.
 - C. If you want to link the index entry you are creating to an *existing* data record, specify the address of that record in PRDAT and specify Inverting in PRCCW. (To get the address of an existing data record, access it with a Read, Write, or Rewrite request. The system always returns the data record address for these operations in PRDFB.)
4. A. Specify the address of your partial record area in PRPRA if the subindex in which you're creating the index entry uses partial records; otherwise specify Suppress Partial Record in PRCCW.
 - B. If you write a partial record, the INFOS system will return its length in PRSRL.

5. Specify Write Immediate in PRSTA if you want to write the index entry and the data record to the file when this request is completed; otherwise, the system will not transfer the contents of the buffers used for this request until it needs them for a subsequent request.
6. If you want to lock or unlock, specify:

Local lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC in PRCCW),
or
Global lock or unlock (PFLOC/PFUNL in PRSTA and CCGLB in PRCCW),
or
Both Local and Global lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC and CCGLB in PRCCW).
7. On each access into the index, the system will return the following in the Processing Packet:
 - A. the subindex level of the key accessed (in PRSIL),
 - B. whether or not the key accessed is a duplicate (in PRSRS), and
 - C. whether or not the key accessed is logically deleted (in PRSRS).

Common Error Messages for a DBAM Write Request

Code	Description
200	ILLEGAL FUNCTION
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
230	NO MORE DISK SPACE
231	RAM ACCESS OUTSIDE FILE
234	RESIDUAL DISK ERROR
252	NO NODE SPACE
255	OUTPUT END VOLUME ERROR
257	COMPARE ERROR (ISAM)
260	RESOLUTION ERROR (ISAM)
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
266	SUBINDEX NOT DEFINED
272	ILLEGAL KEY LENGTH
273	INVALID ENTRY NUMBER
274	ILLEGAL COMMAND CONTROL
275	KEY ALREADY EXISTS
276	KEY POSITIONING ERROR
277	INVALID RECORD LENGTH
414	NO WRITE WITHOUT KEY

DBAM Rewrite Processing Request

1. Specify an access technique in PRCCW of the Extended Processing Packet (Table II-6-4).
 - A. For Keyed access, provide a Key Table containing one key for each subindex level to be accessed and put a pointer to its first word in PRKTP.
 - B. For Relative access, specify a direction of motion in PRCCW.
 - C. For combined Keyed and Relative access, provide a Key Table, put a pointer to it in PRKTP, and specify a direction of motion in PRCCW.

On each access into the Index, the system will return the subindex level number of the key accessed to PRSIL.

2. If you are rewriting a duplicate key, set KTDUP in KDTYP of the Key Definition Packet and specify the occurrence number in KDDKO.
3. Specify Set Current Position in PRCCW if you want to establish a new one; otherwise the system will return to the last established position.
4.
 - A. If you are rewriting an *existing* data record, specify the address of your data area in PRDAT, and the length of the new record you're writing in PRLen. In addition, you may specify a new Record Merit Factor in PRRMF.
 - B. If you are not rewriting an existing data record and you do not want the record returned, specify Suppress Database in PRCCW.
 - C. If you are writing a *new* data record, specify the address of your data area in PRDAT and the length of the new record in PRLen. Also, specify a Merit Factor for the new record in PRRMF if you specified Optimized Distribution in FDFL2 of the Database FDP.
5.
 - A. If you have partial records in the subindex entry in which you are rewriting this index entry, specify either the address of your partial record area (in PRPRA), or Suppress Partial Record (in PRCCW).
 - B. If you rewrite a partial record, the INFOS system will return its length in PRSRL.

6. To link the Index entry accessed for this request to an *existing* data record, specify Inverting in PRCCW and the address of the existing record in PRDAT. However, an Index entry accessed for Inverting cannot already have a data record link. (To get the address of an existing data record, access it with a Read, Write, or Rewrite request. The system always returns the data record address for these operations to PRDFB.)
7. On each access into the index, the system will return the following in the Processing Packet:
 - A. The subindex level of the key accessed (in PRSIL),
 - B. Whether or not the key accessed is a duplicate (in PRSRS), and
 - C. Whether or not the key accessed is logically deleted (in PRSRS).
8. If you want to use the Write Immediate feature, set PFWIF in PRSTA.
9. To lock or unlock a record or an Index entry, specify:

Local lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC in PRCCW),

or

Global lock or unlock (PFLOC/PFUNL in PRSTA and CCGLB in PRCCW),

or

Both local and global lock or unlock (PFLOC/PFUNL in PRSTA and CCLOC and CCGLB in PRCCW).

Common Error Messages for a DBAM Rewrite Request

Code	Description
200	ILLEGAL FUNCTION
210	FILE NOT OPEN
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
231	ACCESS OUTSIDE FILE
234	RESIDUAL DISK ERROR
252	NO NODE SPACE
260	RESOLUTION ERROR (ISAM)
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
266	SUBINDEX NOT DEFINED
267	END OF SUBINDEX
274	ILLEGAL COMMAND CONTROL
276	KEY POSITIONING ERROR
400	DATA BASE REC NOT PRESENT
403	DATA RECORD LOCKED
413	INDEX ENTRY LOCKED

DBAM Define Subindex Processing Request

1. Build a Subindex Definition Packet (see Figure II-6-7 and BLSDP in Chapter 7) and specify its address in PRSID of the Extended Processing Packet (Table II-6-4).
2. In PRCCW, specify an access technique to reach the index entry under which you're defining the subindex.
 - A. For Keyed access, provide a Key Table containing one key for each subindex level to be accessed and put a pointer to its first word in PRKTP.
 - B. For Relative access, specify a direction of motion in PRCCW.
 - C. For combined Keyed and Relative access, provide a Key Table, put its address in PRKTP, and specify a direction of motion in PRCCW.
3. Specify Local lock if you want to lock the accessed index entry (i.e., set PFLOC in PRSTA and CCLOC in PRCCW).
4. Specify Set Current Position if you want to establish a new one by setting CCSCP in PRCCW; otherwise, the system will return to the last established position.

Common Error Messages for a DBAM Define Subindex Request

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
230	NO MORE DISK SPACE
231	RAM ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
265	SUBINDICES NOT ALLOWED
267	END OF SUBINDEX
272	ILLEGAL KEY LENGTH
274	ILLEGAL COMMAND CONTROL
275	KEY ALREADY EXISTS
276	KEY POSITIONING ERROR
406	SUBINDEX LINK COUNT OVERFLOW
407	ALREADY LINKED TO SUBINDEX
410	SUBINDEX LEVEL OVERFLOW

DBAM Link Subindex Processing Request

NOTE: For this request, you must position to *two* existing index entries: the source (which already has a defined subindex under it), and the destination (which must *not* have a subindex defined under it prior to this request). The INFOS system will retrieve the source key and copy its subindex link information to the destination.

1. Build a Link Subindex Processing Packet (see Table II-6-9 and BLDLSP in Chapter II-7).
2. Specify a technique to access *each* index entry (source *and* destination) in PRDCC and PRSCC of the Link Packet.
 - A. For Keyed access, provide a Key Table containing one key for each subindex level to be accessed, and put the address of its first word in PRDCC and PRSCC.
 - B. For Relative access, specify a direction of motion in PRDCC and PRSCC.
 - C. For combined Keyed and Relative access, provide Key Tables, their addresses, *and* a direction of motion.
3. Specify Set Current Position (CCSCP) on either the source (i.e., in word PRSCC) or the destination key (i.e., in word PRDCC) if you want to establish a new current position; otherwise the system will return to the last established position. Note, however, that if you Set Current Position on *both* keys, the new position will be on the destination key when the system successfully completes this request.
4. Specify Local lock or Local unlock if you want to lock or unlock either (or both) the source and destination keys (i.e., set PFLOC or PFUNL in PRSTA and CCLOC in either PRSCC or PRDCC).
5. If you link to an index entry which has been logically deleted, the system will set SRLLD in the Processing Packet.

Common Error Messages for a Link Subindex Request

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
230	NO MORE DISK SPACE
231	ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
265	SUBINDICES NOT ALLOWED
266	SUBINDEX NOT DEFINED
267	END OF SUBINDEX
272	ILLEGAL KEY LENGTH
274	ILLEGAL COMMAND CONTROL
275	KEY ALREADY EXISTS
276	KEY POSITIONING ERROR
403	DATA RECORD LOCKED
406	SUBINDEX LINK COUNT OVERFLOW
407	ALREADY LINKED TO SUBINDEX

DBAM Delete Processing Request

This request allows you to physically delete an index entry and its corresponding data record, or to mark an index entry or a data record (or both) as logically deleted. However, to *physically* delete an index entry and its data record, you must access the index entry via Keyed access (i.e., set CCKEY in PRCCW of the Extended Processing Packet, and provide a Key Table and its address).

1. Specify an access technique in PRCCW.
 - A. For Keyed access, provide a Key Table containing one key for each level of subindex to be accessed and put its address in PRKTP.
 - B. For Relative access, specify a direction of motion in PRCCW.
 - C. For combined Keyed and Relative access, provide a Key Table, put its address in PRKTP, and specify a direction of motion in PRCCW.

If you provide a Key Table, the system will delete (or mark as logically deleted) the last key in the table.

2. A. To mark an *index* entry as logically deleted, specify *Local* Logical Delete (set CCLOC and CCLOG in PRCCW).
- B. To mark a *data record* as logically deleted, specify *Global* Logical Delete (set CCGLB and CCLOG in PRCCW).
- C. You may mark both the index entry and the data record as logically deleted in a single request by setting CCLOC, CCGLB, and CCLOG.

NOTE: For multiple-inverted records, a physical delete will only delete one index entry and decrease the pointer count for the data record by one. You can physically delete a data record only when the count goes to zero (i.e., when you have physically deleted all the subindexes which point to the record).

3. Specify Set Current Position in PRCCW if you want to establish a new one; otherwise the system will return you to the last established position.
 - A. For a logical delete, Set Current Position will return you to the accessed index entry.
 - B. For a physical delete, Set Current Position will return you to either the next higher index entry after the one deleted (if any), or the index entry that owns the subindex containing the deleted entry, or (if you are in subindex level 0) to the system-set initial current position.
4. If you want to delete a duplicate key, you must set KTDUP in KDTYP of the Key Definition Packet and put the key's occurrence number in KDDKO.

Common Error Messages for a DBAM Delete Request

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
226	NO SUCH VOLUME
230	NO MORE DISK SPACE
231	RAM ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
267	END OF SUBINDEX
270	DELETE POSITIONING ERROR
272	ILLEGAL KEY LENGTH
274	ILLEGAL COMMAND CONTROL
275	KEY ALREADY EXISTS
276	KEY POSITIONING ERROR
403	DATA RECORD LOCKED
411	SUBINDEX HAS SUBINDEX; DELETE SUBINDEX ERROR
412	ATTEMPT TO DELETE ENTRY WITHOUT KEYED ACCESS
413	INDEX ENTRY LOCKED

DBAM Delete Subindex Processing Request

1. Specify an access technique in PRCCW of the Extended Processing Packet (Table II-6-4).
 - A. For Keyed access, provide a Key Table containing one key for each subindex level to be accessed and put its address in PRKTP.
 - B. For Relative access, specify a direction of motion in PRCCW.
 - C. For combined Keyed and Relative access, provide a Key Table, put its address in PRKTP, and specify a direction of motion in PRCCW.
2. The INFOS system will delete the subindex under the last key in the Key Table provided, or that under the key accessed by Relative motion.
3. Specify Set Current Position in PRCCW to establish a position on the subindex entry which owns the subindex you're deleting; otherwise, the system will return to the last established position.

NOTE: You may issue this command as you desire, but the system will not physically delete the subindex until its use count goes to zero. That is, the subindex will remain as long as at least one other subindex path uses that subindex.

Common Error Messages for a Delete Subindex Request

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
230	NO MORE DISK SPACE
231	RAM ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
265	SUBINDICES NOT ALLOWED
266	SUBINDEX NOT DEFINED
267	END OF SUBINDEX
274	ILLEGAL COMMAND CONTROL
276	KEY POSITIONING ERROR

DBAM Reinstatement Processing Request

1. Specify an access technique in PRCCW of the Extended Processing Packet (Table II-6-4).
 - A. For Keyed access, provide a Key Table with one key for each subindex level to be accessed and put its address in PRKTP.
 - B. For Relative access, specify a direction of motion in PRCCW.

- C. For combined Keyed and Relative access, provide a Key Table, put its address in PRKTP, and specify a direction of motion in PRCCW.
2.
 - A. Specify Local if you want to reinstate the index entry (i.e., set CCLOC in PRCCW).
 - B. Specify Global if you want to reinstate the data record (i.e., set CCGLB in PRCCW).
 - C. You may reinstate both the index entry and the data record in a single request; that is, you may set both CCLOC and CCGLB in PRCCW.
 3. Specify Set Current Position in PRCCW to establish a position either on the index entry you're reinstating (if you also set CCLOC in PRCCW), or on the index entry which points to the data record you're reinstating (if you set CCGLB in PRCCW).

Common Error Messages for a DBAM Reinstatement Request

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
230	NO MORE DISK SPACE
231	RAM ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
272	ILLEGAL KEY LENGTH
274	ILLEGAL COMMAND CONTROL
275	KEY ALREADY EXISTS
276	KEY POSITIONING ERROR
400	DATA BASE REC NOT PRESENT
403	DATA RECORD LOCKED

DBAM Retrieve Key and Retrieve High Key Processing Requests

1. In the Extended Processing Packet (Table II-6-4), specify the address of the area to which you want the system to return the key in PRDAT, and provide the address of a Key Table in PRKTP to receive a duplicate key occurrence number (if you allowed duplicates). If you didn't allow duplicates, place a -1 in KDDKO of the Key Table.
2. Specify an access technique in PRCCW.
 - A. For Keyed access, provide a Key Table containing one key for each subindex level to be accessed and put its address in PRKTP.
 - B. For Relative access, specify a direction of motion in PRCCW.
 - C. For combined Keyed and Relative access, provide a Key Table, put its address in PRKTP, and specify a direction of motion in PRCCW.

3. Specify Set Current Position in PRCCW to establish a position on the index entry accessed by this request; otherwise, the system will return to the last established position.
4. The INFOS system will return the key to your data area, its length to PRSRL, and its occurrence number if it's a duplicate to PRSID.

Common Error Messages for the DBAM Retrieve Key and Retrieve High Key Request

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
231	RAM ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
267	END OF SUBINDEX
274	ILLEGAL COMMAND CONTROL
276	KEY POSITIONING ERROR

DBAM Retrieve Subindex Definition Processing Request

1. Specify (in PRSID) the address of the area where you want the system to return the subindex definition.
2. Specify an access technique in PRCCW.
 - A. For Keyed access, provide a Key Table containing one key for each subindex level to be accessed and put its address in PRKTP.
 - B. For Relative access, specify a direction of motion in PRCCW.
 - C. For combined Keyed and Relative access, provide a Key Table, put its address in PRKTP, and specify a direction of motion in PRCCW.
3. Specify Set Current Position in PRCCW to establish a position on the index entry accessed by this request; otherwise, the system will return you to the last established current position.

Common Error Messages for a Retrieve Subindex Definition Request

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
230	NO MORE DISK SPACE
231	RAM ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
265	SUBINDICES NOT ALLOWED
267	END OF SUBINDEX
272	ILLEGAL KEY LENGTH
274	ILLEGAL COMMAND CONTROL
275	KEY ALREADY EXISTS
276	KEY POSITIONING ERROR
406	SUBINDEX LINK COUNT OVERFLOW
407	ALREADY LINKED TO SUBINDEX
410	SUBINDEX LEVEL OVERFLOW

Licensed Material - Property of Data General Corporation

DBAM Retrieve Status Processing Request

1. Specify an access technique in PRCCW of the Extended Processing Packet (Table II-6-4).
 - A. For Keyed access, provide a Key Table containing one key for each subindex level to be accessed and put its address in PRKTP.
 - B. For Relative access, specify a direction of motion in PRCCW.
 - C. For combined Keyed and Relative access, provide a Key Table, puts its address in PRKTP, and specify a direction of motion in PRCCW.
2. Specify Set Current Position in PRCCW to establish a position on the index entry accessed by this request; otherwise, the system will return you to the last established position.
3. The system will return the following information:
 - A. The record length (in PRLen), whether or not it's marked as logically deleted, and whether or not it's locked (both in PRSRS).
 - B. The key length (in PRSRL), whether it has been logically deleted (in PRSRS), and its occurrence number if it's a duplicate (in KDDKO of the Key Definition Packet).

Common Error Messages for a DBAM Retrieve Status Request

Code	Description
213	UNRESOLVED RESOURCE CONFLICT
231	RAM ACCESS OUTSIDE FILE
261	ILLEGAL REL MOTION
262	INVALID NODE ADDRESS
263	INVALID CURRENT ENTRY
267	END OF SUBINDEX
274	ILLEGAL COMMAND CONTROL
276	KEY POSITIONING ERROR
400	DATA BASE REC NOT PRESENT
403	DATA RECORD LOCKED

Chapter 6 Packet Formats

In the preceding chapters in this section, we told you that you must set up FDPs and VDPs and supply processing packets to make the INFOS system do what you want it to. In this chapter we get down to the nuts and bolts of packets. We will show you what each of the various types of packets looks like, and we will describe each specification within each packet.

General Packet Information and Conventions

Each INFOS system packet contains two types of parameters: those which occupy one or more full words, and those which occupy one or more bits *within* a given word. For full word specifications, you must either enter the appropriate value or default the parameter by specifying -1 for the entire word. For single-bit specifications, a field value of 1 indicates the presence of that parameter, and a value of 0 (the system default value) indicates the absence of it. For example, if you specify FIUBR for an FDP, you will set one bit of one word within that packet to 1, indicating that you have Unblocked records. If you default this parameter (or make no specification for it), the system will assume that you don't have unblocked records -- in other words, that your records are blocked.

We will present the data for the various packets as a series of charts containing the following information:

- The Field Name for each parameter (under "Field Name");
- The feature or option which that parameter specifies (under "Description");

- The field or bit name (i.e., the mnemonic) which you specify to set up that parameter (under "Specified As");
- The field value which will indicate the presence of that parameter (under "Field Value"); and
- What it will mean if you use the default value (under "Default Means").

In the last column ("Default Means"), the word "Required" indicates that you cannot default that parameter. Also, each chart contains a series of footnotes to explain factors you should be aware of when using certain parameters. We advise you to read and heed these footnotes.

Finally, we refer you to Part One of this manual for further details on any of the features and options available in any of the packets.

File Definition Packet (FDP)

As you know by now, you must set up an FDP to describe each file which your program opens. Figure II-6-1 shows the location of every possible parameter in an FDP. You will not use *every* parameter for any one access method, so we will summarize the applicable parameters for each access method after we describe the packet.

File Definition Packet

FIELD NAME	DECIMAL OFFSET	
	0	
FDFL1	3	UBR AM1 AM2 FT1 FT2 RAW OVR EXF PM1 PM2 RER INV
FDFL2	4	SPM ORD DHR FDP
FDBLK	5	BLOCK OR PAGE SIZE
FDBUF	6	NUMBER OF BUFFERS
FDLEN (FDNIL)	7	RECORD LENGTH OR NUMBER OF INDEX LEVELS
FDNVD	10	NUMBER OF VOLUME TABLE ENTRIES
FDVTP	11	VOLUME TABLE POINTER
FDUIT	13	USER INPUT TRANSLATION TABLE POINTER
FDUOT	15	USER OUTPUT TRANSLATION TABLE POINTER
FDTCF	17	LT1 LT2 LT3 SFT OT1 OT2 OT3 OT4 LL1 LL2 LL3 IT1 IT2 IT3 IT4
FDDSD	18	DATA SENSITIVE DELIMITER TABLE POINTER
FDFSI	20	FILE SET ID POINTER
FDEXP	23	EXPIRATION DATE
FDSEQ	26	SEQUENCE NUMBER
FDGEN	27	GENERATION NUMBER
FDACC	28	FILE ACCESSIBILITY
FDIDO	29	INITIAL DATA OFFSET
FDSFT	30	SELECTIVE FIELD TRANSLATION TABLE POINTER
FDDBP	42	DATABASE FILE DEFINITION PACKET POINTER OR NAME POINTER
FDINS	44	INITIAL NODE SIZE
FDMKL/FDPRL	46	MAXIMUM KEY LENGTH PARTIAL RECORD LENGTH
FDRMF	48	ROOT NODE MERIT FACTOR
FDIFL	49	PKC NSI HPN TSI PRM

SD-00159A

Figure II-6-1

Table II-6-1. File Definition Packet (FDP)

Field Name/Description	Specified As	Field Value	Default Means
FDL1	3	UBR AM1 AM2 FT1 FT2 RAW OVR EXF PM1 PM2 RER INV	
File Definition Flags			
Unblocked Records	F1UBR	1	Blocked Records
Access Method	F1AM1/2		Required
RAM	F1RAM	00	
SAM	F1SAM	01	
ISAM	F1ISM	10	
DBAM	F1DBM	10	
Record Format	F1FT1/2		Required
Undefined	F1UND	00	
Variable	F1VAR	01	
Fixed	F1FIX	10	
Data Sensitive	F1SEN	11	
Read-After-Write verification	F1RAW	1	No verification
Overwrite	F1OVR	1	No overwriting
Exclusive File ¹	F1EXF	1	No exclusive use of file
Processing Mode	F1PM1/2		Required
Input	F1INP	00	
Update	F1UPD	01	
Output	F1OUT	10	
Create Update	F1CRU	11	
Rewrite Mode ²	F1RER	1	No rewriting or release allowed
Inverting	F1INV	1	See Comment 1
FDL2	4	SPM ORD DHR FDP	
Space Management ³	F2SPM	1	No Space Management
Optimized ⁴ Record Distribution	F2ORD	1	No optimized distribution
Disable Hierarchical Replacement ⁵	F2DHR	1	Buffers managed on Hierarchically Modulated LRU basis (See Comment 4)
Database FDP Present ⁶	F2FDP	1	See Comment 1
FDBLK	5	BLOCK OR PAGE SIZE	
Block or Page Size	FDBLK	Size of blocks or pages	512 bytes for disks; 80 bytes for all other devices

Footnotes

1. Applicable only to SAM disk files and RAM files.
2. Applicable only to SAM disk files opened in the Update mode.
3. Applicable only to ISAM or DBAM files (index, database, or both) opened in the Create Update mode.
4. Applicable only to ISAM and DBAM files opened in Create Update.
5. Applicable only to ISAM and DBAM indexes.
6. Applicable only to ISAM and DBAM files opened in Update or DBAM indexes opened in Create Update (see also Comment 1 below).

Table II-6-1. File Definition Packet (continued)

Field Name/Description	Specified As	Field Value	Default Means
FDBUF 6 <input type="text" value="NUMBER OF BUFFERS"/> Number of buffers	FDBUF	Number	At creation: 2 for Index files, 1 for all others After creation: creation specifications
FDLEN 7 <input type="text" value="RECORD LENGTH OR NUMBER OF INDEX LEVELS"/> FDNIL Record Length ⁷ Number of Index Levels ⁴	FDLEN FDNIL	Size Number	SAM/RAM files = block size ISAM/DBAM = required in database FDP Required for ISAM/DBAM indexes
FDNVD 10 <input type="text" value="NUMBER OF VOLUME TABLE ENTRIES"/> Number of Volume Table entries	FDNVD	Number	Only one file volume recognized
FDVTP 11 <input type="text" value="VOLUME TABLE POINTER"/> Volume Table Pointer	FDVTP	Address of first word of volume table	Required
FDUIT 13 <input type="text" value="USER INPUT TRANSLATION TABLE POINTER"/> User Input Translation Table Pointer	FDUIT	Address of first byte of Input Translation Table	See Comment 2
FDUOT 15 <input type="text" value="USER OUTPUT TRANSLATION TABLE POINTER"/> User Output Translation Table Pointer	FDUOT	Address of first byte of Output Translation Table	See Comment 2

Footnotes (continued)

7. For Variable and Data Sensitive records, specify your expected maximum record length; for Undefined length records, default this specification; for Fixed length records, specify the exact record size. For tape files opened in the Input mode, specify the same length as you did at file creation.

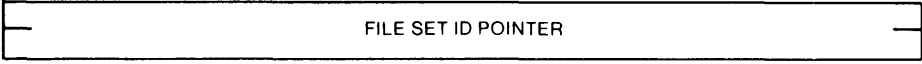

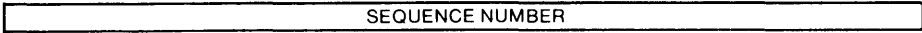
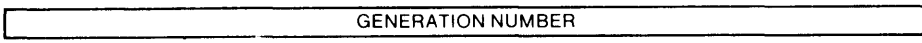

Table II-6-1. File Definition Packet (continued)

Field Name/Description	Specified As	Field Value	Default Means															
FDTCF	17	<table border="1"> <tr> <td>LT1</td><td>LT2</td><td>LT3</td><td>SFT</td><td>OT1</td><td>OT2</td><td>OT3</td><td>OT4</td><td>LL1</td><td>LL2</td><td>LL3</td><td>IT1</td><td>IT2</td><td>IT3</td><td>IT4</td> </tr> </table>	LT1	LT2	LT3	SFT	OT1	OT2	OT3	OT4	LL1	LL2	LL3	IT1	IT2	IT3	IT4	
LT1	LT2	LT3	SFT	OT1	OT2	OT3	OT4	LL1	LL2	LL3	IT1	IT2	IT3	IT4				
Translation and Label Control Flags																		
Label Type ⁸	TCLT1/2/3		ANSI															
ANSI Labels	TCANS	000																
IBM Labels	TCIBM	001																
Selective Field Translation	TCSFT	1	Entire record translated (See Comment 3)															
Output Translation ⁹			No translation															
No code translation	TCNTO	0000																
EBCDIC to ASCII	TCEAO	0001																
ASCII to EBCDIC	TCEAO	0010																
Anything to your code (See Comment 2)	TCUTO	1111																
Label Level	TCLL1/2/3		Required for tape processing															
Level 1	TCLV1	000																
Level 2	TCLV2	010																
Level 3	TCLV3	011																
Input Translation ¹⁰	TCIT1/2/3/4		No translation (See Comment 2)															
No Input translation	TCINIT	0000																
EBCDIC to ASCII	TCEAI	0001																
ASCII to EBCDIC	TCAEI	0010																
Anything to your code	TCUTI	1111																
FDDSD	18	<table border="1"> <tr> <td style="text-align: center;">DATA SENSITIVE DELIMITER TABLE POINTER</td> </tr> </table>		DATA SENSITIVE DELIMITER TABLE POINTER														
DATA SENSITIVE DELIMITER TABLE POINTER																		
Data Sensitive Delimiter ¹¹ Table Pointer	FDDSD	Address of first byte of Data Sensitive Delimiter Table	Delimiter = 0,), or ↓															

Footnotes (continued)

- 8. Applies only to labeled tape files.
- 9. Applies only to SAM or RAM files opened in the Output or Create Update mode.
- 10. Applies only to SAM or RAM files opened in the Input or Update mode.
- 11. Applies only to SAM files where the record delimiter character is *not* 0,), or ↓.

Table II-6-1. File Definition Packet (continued)

Field Name/Description	Specified As	Field Value	Default Means
FDPSI 20 File Set ID Pointer ¹³	FDPSI		Address ¹² of the first byte of file set ID (for ANSI labels) or Volume serial number (for IBM labels). No file set ID in file's HDR1 label
FDEXP 23 Expiration Date ¹³	FDEXP		Date File can be overwritten at any time
FDSEQ 26 Sequence Number ¹³	FDSEQ		Number On Input: No search according to sequence number On Output: Assign next available sequence number to file based on current tape position
FDGEN 27 Generation Number ¹³ (See Appendix B)	FDGEN		Number On Input: No search by generation and version number On Output: File will have generation #1 and version #0
FDACC 28 File Accessibility ¹³	FDACC		Restricted access character No restrictions on file processing

Footnotes (continued)

- 12. The File Set ID or Volume Serial Number must occupy the first six bytes of a seven-byte field, and the last byte must contain the null character.
- 13. Applies only to tape files.

Table II-6-.1 File Definition Packet (continued)

Field Name/Description	Specified As	Field Value	Default Means
FDIDO Initial Data Offset ¹⁴	29 FDIDO	29 INITIAL DATA OFFSET	Number of bytes offset Records begin in first byte of block
FDSFT Selective Field Translation Pointer ¹⁵	FDSFT	30 SELECTIVE FIELD TRANSLATION TABLE POINTER	Address of first byte of Selective Field Translation Table Required if you set TCSFT = 1
FDDBP Database FDP Pointer or Name Pointer	FDDBP	42 DATA BASE FILE DEFINITION PACKET POINTER OR NAME POINTER	Address of first word of Database FDP ¹⁶ See Comment 1 below
FDINS Initial Node Size	FDINS	44 INITIAL NODE SIZE	Initial size (in bytes) of the Root Node for the Index or Subindex you're defining Required for ISAM and DBAM files
FDMKL FDPRL Maximum Key Length Partial Record Length	FDMKL FDPRL	46 MAXIMUM KEY LENGTH PARTIAL RECORD LENGTH	Length of the longest key in Index (ISAM) or Level 0 subindex (DBAM) Length of partial records in subindex level 0 System derived maximum ¹⁸ No partial records allowed
FDRMF Root Node Merit Factor ¹⁷	FDRMF	48 ROOT NODE MERIT FACTOR	Merit Factor Required if you set F2ORD = 1

Footnotes (continued)

- 14. Applies only to SAM disk or tape files opened in the Input mode.
- 15. Applies only to SAM and RAM files.
- 16. For Index files, if F1INV = 1 and F2FDP = 0, then you must specify the address of the first byte of the database filename.
- 17. If you did not set F2ORD = 1 (Optimized Record Distribution), then you must specify 0 for this word.
- 18. The system will derive the default maximum according to this formula:

$$\text{maxkeylength} = \left(\frac{\text{initial node size} - 36}{3} \right) - (10) - (\text{partial record length})$$

or, if you allowed subindexes

$$\text{maxkeylength} = \left(\frac{\text{initial node size} - 36}{3} \right) - (14) - (\text{partial record length})$$

Table II-6-1. File Definition Packet (concluded)

Field Name/Description	Specified As	Field Value	Default Means
FDIFL 49	PKC NSI HPN TSI PRM		
Index Flags			
Perform Key Compression	IXPKC	1	No key compression
No Subindexes	IXNSI	1	Subindexes allowed
High Priority Node (See Comment 4)	IXHPN	1	No high priority status for root node of subindex level 0
Temporary Subindex ¹⁹	IXTSI	1	The system will modify the record use count when you perform a Write, an inverted Write, or a Delete
Permanent Data Records ²⁰	IXPRM	1	Access to record after file closing will be unpredictable

Footnotes (concluded)

19. You should only set this bit if you can ensure that the record will not be deleted or rewritten to a greater length while you are using the temporary subindex.
20. If you open a DBAM index in Create Update and specify F1INV (Inverting) plus IXTSI (Temporary Subindex), then you should also specify IXPRM. This will ensure that you can rewrite the data record to a different length and still be able to access it through temporary subindexes. Note, however, that if you do this, you will never be able to physically delete the data record from the database.

Comments

1. You should use the parameters F1INV (Inverting), F2FDP (Database FDP present), and FDDBP (Database FDP pointer) in combination as follows:
 - When you open an ISAM or DBAM file in the *Create Update* mode, set F1INV and F2FDP to 0 and place the address of the first word of the database FDP in FDDBP.
 - When you open an ISAM or DBAM file in the *Update* mode and you do *not* want a runtime file specification for the database, set F1INV and F2FDP to 0 and place the address of the first byte of the database filename in FDDBP. If you *do* want a runtime file specification for the database, set F2FDP to 1 and place the address of the first word of the database FDP in FDDBP.
 - If you want to create another index for an existing database, open the DBAM index file in the Create Update mode and set F1INV to 1. Then, if you do *not* want a runtime file specification for the database, set F2FDP to 0 and place the address of the first byte of the database filename in FDDBP. If you *do* want a runtime file specification for the database, set F2FDP to 1 and enter the address of the first word of the database FDP in FDDBP.

2. For input and output translation, you must specify parameters as follows:
 - If you specify TCUTI, then you must build a code translation table and specify its address in FDUIT. If you specified TCNIT, TCEAI, or TCAEI, you don't need to build a translation table or specify its address unless you also specified TCUTO.
 - If you specify TCUTO, you need a code translation table, and you must specify its address in FDUOT.

NOTE: You may use the same translation table for input and output; you don't need to build a separate one for each processing function.

3. If you want only selected fields of a record translated, you must specify TCSFT, build a Selective Field Translation Table, and put the address of its first byte in FDSFT. If you do this, furthermore, you must specify that you want translation on input or output by setting the appropriate bits in TCIT1/2/3/4 or TCOT1/2/3/4.
4. If you open a DBAM index file in the Create Update mode and set both F2DHR and IXHPN to 1, a temporary conflict arises because both options modify the Hierarchically modulated LRU technique normally used by the INFOS system. If you specify both parameters, the system will use F2DHR for the current opening, and record IXHPN (with a value of 1) as an unchangeable part of the PFS. Therefore, on subsequent openings, the system will use IXHPN if you don't specify F2HDR. If you *do* specify F2HDR, it will take precedence over IXHPN and the system will use a strict LRU buffer management technique. (See Chapter 6 in Part One for details on LRU and Hierarchical Modulation.)

Licensed Material - Property of Data General Corporation

**INFOS FDP Parameters by
Access Method/Device/and
Processing Mode**

SAM files/Disk/Create Update and Output Modes

Required	Optional
F1AM1/2	F1UBR
F1PM1/2	F1RAW
F1FT1/2	F1EXF
FDVTP	TCOT1/2/3/4
	FDUOT
	TCSFT
	FDSFT
	FDBLK
	FDLEN
	FDBUF
	FDNVD
	FDDSD

SAM files/Labeled Tape/Input mode

Required	Optional
F1AM1/2	TCLT1/2/3
F1PM1/2	TCLL1/2/3
F1FT1/2	TCFSI
FDVTP	FDSEQ
	FDGEN
	FDACC
	TCIT1/2/3/4
	FDUIT
	TCSFT
	FDSFT
	FDIDO
	FDNVD

SAM files/Disk/Input and Update modes

Required	Optional
F1AM1/2	F1RAW
F1PM1/2	F1OVR (Update only)
F1FT1/2	F1EXF
FDVTP	F1RER
	TCIT1/2/3/4
	FDUIT
	TCSFT
	FDSFT
	FDIDO
	FDDSD

SAM files/Unlabeled Tape/Output mode

Required	Optional
F1AM1/2	F1UBR
F1PM1/2	TCOT1/2/3/4
F1FT1/2	FDOUT
FDVTP	TCSFT
	FDSFT
	FDBLK
	FDLEN
	FDBUF
	FDNVD
	FDDSD

SAM files/Labeled Tape/Output mode

Required	Optional
F1AM1/2	F1UBR
F1PM1/2	TCLT1/2/3
F1FT1/2	TCLL1/2/3
FDVTP	TCFSI
	FDEXP
	FDSEQ
	FDGEN
	FDACC
	TCOT1/2/3/4
	FDOUT
	TCSFT
	FDSFT
	FDBLK
	FDLEN
	FDBUF
	FDNVD

SAM files/Unlabeled Tape/Input mode

Required	Optional
F1AM1/2	TCIT1/2/3/4
F1PM1/2	FDUIT
F1FT1/2	TCSFT
FDVTP	FDSFT
	FDIDO
	FDNVD
	FDDSD

SAM files/TTY (Interactive Terminal)/Output mode

Required	Optional
F1AM1/2	TCOT1/2/3/4
F1PM1/2	FDOUT
F1FT1/2	TCSFT
FDVTP	FDSFT
	FDBLK
	FDLEN
	FDBUF
	FDDSD

SAM files/TTY (Interactive Terminal)/Input mode

Required	Optional
F1AM1/2	TCIT1/2/3/4
F1PM1/2	FDUIT
F1FT1/2	TCSFT
FDVTP	FDSFT
	FDBLK
	FDLEN
	FDBUF
	FDDSD

ISAM Index files/Disk/Create Update mode

Required	Optional
F1AM1/2	F1RAW
F1PM1/2	F2SPM
F1FT1/2	F2DHR
FDVTP	FDMKL
IXNSI	FDINS
	FDBLK
	FDBUF
	FDNVD

SAM files/Line Printer/Output mode

Required	Optional
F1AM1/2	TCOT1/2/3/4
F1PM1/2	FDUOT
F1FT1/2	TCSFT
FDVTP	FDSFT
	FDBLK
	FDLEN
	FDBUF
	FDDSD

ISAM Index files/Disk/Update mode

Required	Optional
F1AM1/2	F1RAW
F1PM1/2	F2FDP
F1FT1/2	FDDBP
FDVTP	F2DHR

RAM files/Disk/Output or Create Update mode

Required	Optional
F1AM1/2	F1RAW
F1PM1/2	F1EXF
F1FT1/2	TCOT1/2/3/4
FDVTP	FDUOT
	TCSFT
	FDSFT
	FDBLK
	FDLEN
	FDBUF
	FDNVD

ISAM Database files/Disk/Create Update mode

Required	Optional
F1AM1/2	F1RAW
F1PM1/2	F2SPM
F1FT1/2	FDBLK
FDVTP	FDBUF
	FDNVD

ISAM Database files/Disk/Update mode

Required	Optional
(None)	F1AM1/2
	F1PM1/2
	F1FT1/2
	FDVTP
	F1RAW

RAM files/Disk/Input or Update mode

Required	Optional
F1AM1/2	F1RAW
F1PM1/2	F1EXF
F1FT1/2	TCIT1/2/3/4
FDVTP	FDUIT
	TCSFT
	FDSFT

DBAM Index files/Disk/Create Update mode

Required	Optional	Optional
F1AM1/2	F1RAW	IXNSI
F1PM1/2	F1INV	FDNIL
F1FT1/2	F2FDP	FDPRL
FDVTP	FDDBP	FDMKL
	F2SPM	FDINS
	F2ORD	IXPKC
	FDRMF	IXTSI
	F2DHR	IXPRM
	IXHPN	FDBLK
	FDBUF	FDNVD

Licensed Material - Property of Data General Corporation

DBAM Index files/Disk/Update mode

Required	Optional
F1AM1/2	F1RAW
F1PM1/2	F2FDP
F1FT1/2	FDDBP
FDVTP	F2DHR

DBAM Database files/Disk/Update mode

Required	Optional
(None)	F1AM1/2
	F1PM1/2
	F1FT1/2
	FDVTP
	F1RAW

DBAM Database files/Disk/Create Update mode

Required	Optional
F1AM1/2	F1RAW
F1PM1/2	F2SPM
F1FT1/2	F2ORD
FDVTP	FDBLK
	FDBUF
	FDNVD

Volume Definition Packets (VDP)

Just as you need an FDP to describe each file you open, you also need a VDP to describe each volume of that file. When you open any INFOS disk file in the Output or Create Update mode, you'll need a Volume Table. This is nothing more than a contiguous sequence of the VDPs for the file volumes. Once you have defined the volumes and created your file, you only need to supply the VDP for the first volume when you open the file for updating. (The INFOS system makes a permanent, unchangeable record of volume definitions for disk files and uses it when you open in either the Input or Update mode.)

When you open a tape file in either the Input or Output mode, you must supply a Volume Table with a VDP for each reel of tape. (The system does not permanently store tape file VDPs.)

Files on interactive terminals and line printers are single volume files by definition, and you must build a VDP for them each time you open them for processing.

Figure II-6-2 shows the format of a VDP. Following the packet description is a breakdown of the required and optional parameters for each access method.

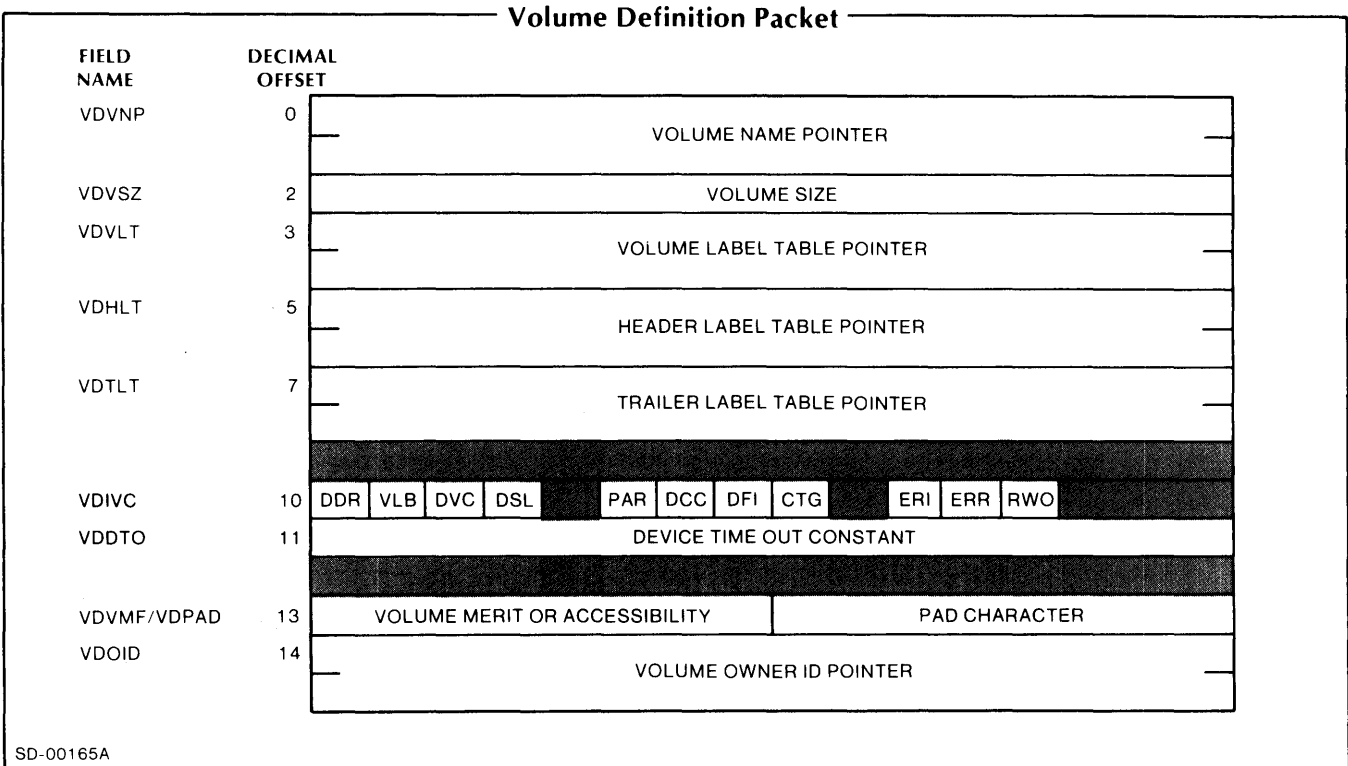







Figure II-6-2

Table II-6-2. Volume Definition Packet (VDP)

Field Name/Description	Specified As	Field Value	Default Means
VDVNP Volume Name Pointer	0 VDVNP		Address of first byte of volume name Required
VDVSZ Volume size ⁶	2 VDVSZ		Number of contiguous blocks allocated Unlimited volume size
VDVLT Volume Label Table Pointer ¹	3 VDVLT		Address of first byte of volume label table No user volume label processed
VDHLT Header Label Table Pointer ¹	5 VDHLT		Address of first byte of Header Label Table No user header label processed
VDTLT Trailer Label Table Pointer	7 VDTLT		Address of first byte of Trailer Label Table No trailer labels processed

Footnotes:

1. Applies only to labeled tape files.
2. Applies only to unlabeled tape files.
3. This option applies only to SAM disk and RAM files opened in the Output or Create Update mode. You should enable conflict checking if more than one user will open a file for updating at the same time, or if a multitask program will open it.
4. If you specify ICERI=1, then you must also specify ICERR=1. You may also specify ICERI=0 and ICERR=1 (i.e., system initialization and runtime release), but if you do so, you must also set PRDSP = 1 in the Processing Packet.
5. Applicable only if you specified Optimized Distribution for a DBAM file opened in the Create Update mode. (See Part One, Chapter 5.)
6. Whenever you open a disk file in the Output or Create Update mode and set ICCTG=1 (indicating Contiguous Allocation), then you must specify the number of contiguous blocks to be allocated. If you set ICCTG=0, you may enter either the maximum number of blocks to be allocated on the volume being defined, or -1. Specifying -1 indicates that the volume is unlimited in size and that the system will allocate up to 65,535 blocks on that volume.

Table II-6-2. Volume Definition Packet (concluded)

Field Name/Description	Specified As	Field Value	Default Means										
VDIVC	10	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="width: 20px;">VLB</td> <td style="width: 20px;">DVC</td> <td style="width: 20px;">DSL</td> <td style="width: 20px;">PAR</td> <td style="width: 20px;">DCC</td> <td style="width: 20px;">DFI</td> <td style="width: 20px;">CTG</td> <td style="width: 20px;">ERI</td> <td style="width: 20px;">ERR</td> <td style="width: 20px;">RWO</td> </tr> </table>	VLB	DVC	DSL	PAR	DCC	DFI	CTG	ERI	ERR	RWO	
VLB	DVC	DSL	PAR	DCC	DFI	CTG	ERI	ERR	RWO				
VDIVC Volume Characteristics													
Variable Length Blocks	ICVLB	1	System transfers Fixed length blocks										
Disable System Labeling ²	ICDSL	1	System will try to read or write labels										
Generate Parity ²	ICPAR	1	System generates odd parity										
Enable Conflict Checking ³	ICECC	1	No conflict checking										
Disable File Initialization	ICDFI	1	System will zero out blocks for the file										
Contiguous Allocation ⁶	ICCTG	1	Random allocation										
Enable Runtime Initialization ¹	ICERI	1	You must initialize volumes via system calls ⁴										
Enable Runtime Release ¹	ICERR	1	You must release volumes via system calls ⁴										
Rewind on Volume Open ¹	ICRWO	1	No rewind on open										
VDDTO	11	DEVICE TIME OUT CONSTANT											
Device Timeout Constant	VDDTO	Number of seconds	System-defined timeout intervals used (See Part One, Chapter 6, or Appendix E)										
VDVMF VDVAC VDPAD	13	<table border="1" style="display: inline-table; border-collapse: collapse;"> <tr> <td style="width: 50%;">VOLUME MERIT OR ACCESSIBILITY</td> <td style="width: 50%;">PAD CHARACTER</td> </tr> </table>		VOLUME MERIT OR ACCESSIBILITY	PAD CHARACTER								
VOLUME MERIT OR ACCESSIBILITY	PAD CHARACTER												
Volume Merit Factor ⁵	VDVMF	Number	Not using Optimized Distribution										
Volume Accessibility ¹ Pad Character	VDVAC VDPAD	Code Number ASCII character	No accessibility code Binary 0 (null)										
VDOID	14	VOLUME OWNER ID POINTER											
Volume Owner ID Pointer ¹	VDOID	Address of first byte of owner ID	No return of ID										

Licensed Material - Property of Data General Corporation
INFOS VDP Parameters by
Access Method/Device/and
Processing Mode

— SAM files/Disk/Create Update and Output modes —

Required	Optional
VDVNP	VDVSZ ICECC ICDFI ICCTG VDDTO VDPAD

— SAM files/Disk/Input and Update modes —

Required	Optional
VDVNP	(None)

— SAM files/Labeled Tape/Output mode —

Required	Optional
VDVNP	VDVLT VDHLT VDTLT ICVLB ICERI ICERR ICRWO VDDTO VDVAC VDPAD VDOID

— SAM files/Labeled Tape/Input mode —

Required	Optional
VDVNP	VDVLT VDHLT VDTLT ICVLB ICERI ICERR ICRWO VDDTO VDVAC VDOID

— SAM files/Unlabeled Tape/Output mode —

Required	Optional
VDVNP ICDSL	ICVLB ICPAR VDDTO VDPAD

— SAM files/Unlabeled Tape/Input mode —

Required	Optional
VDVNP ICDSL	ICVLB VDDTO VDPAD

— SAM files/TTY (Interactive Terminal)/Output mode —

Required	Optional
VDVNP	ICDDR VDDTO VDPAD

— SAM files/TTY (Interactive Terminal)/Input mode —

Required	Optional
VDVNP	ICDDR VDDTO

— SAM files/Line Printer/Output mode —

Required	Optional
VDVNP	ICDDR VDDTO VDPAD

— RAM files/Disk/Output and Create Update modes —

Required	Optional
VDVNP	VDVSZ ICECC ICDFI ICCTG VDDTO

RAM files/Disk/Input or Update modes

Required	Optional
VDVNP	(None)

ISAM Database files/Disk/Update mode

Required	Optional
(None)	VDVNP

ISAM Index files/Disk/Create Update mode

Required	Optional
VDVNP	VDVSZ ICDFI ICCTG VDDTO VDPAD

DBAM Index files/Disk/Create Update mode

Required	Optional
VDVNP	VDVSZ ICDFI ICCTG VDDTO VDVMF VDPAD

ISAM Index files/Disk/Update mode

Required	Optional
VDVNP	(None)

DBAM Index files/Disk/Update mode

Required	Optional
VDVNP	(None)

ISAM Database files/Disk/Create Update mode

Required	Optional
VDVNP	VDVSZ ICDFI ICCTG VDDTO VDPAD

DBAM Database files/Disk/Create Update mode

Required	Optional
VDVNP	VDVSZ ICDFI ICCTG VDDTO VDVMF VDPAD

DBAM Database files/Disk/Update mode

Required	Optional
(None)	VDVNP

Volume Tables

As we mentioned earlier, a Volume Table is a contiguous sequence of one or more VDPs, each of which is 16 words long. The contents of a Volume Table for any given INFOS file depend on the type of device on which the file resides and the processing mode in which you're opening it.

To build a Volume Table for SAM disk, RAM, ISAM, or DBAM files which you are opening in the Output or Create Update mode (i.e., at file creation), you must:

1. Place the VDPs in the Table in the order in which you want the INFOS system to process them. If you are using Optimized Distribution for an index or database file, you must define its volumes in order of decreasing speed. That is, define your most-accessed (or fastest) volume(s) first, and your least-accessed (or slowest) volume(s) last.
2. Specify (in the file's FDP) the number of volumes if the file resides on more than one.
3. In the file's FDP, supply the address of the first word of the first VDP in the Table.

For these files, the INFOS system will automatically build and maintain an internal Volume Definition File, or *X.VL*, where *X* is the name of your file. (In other words, if your file's name is PIGEON, its Volume Definition File will be PIGEON.VL.) When you open a disk file in the Input or Update mode, the system uses the contents of the .VL file as a Volume Table. However, for it to do this, you must:

1. Build a Volume Table consisting of only one VDP, which, in turn, contains only the address of the name of the first file volume you defined when you created the file. (For ISAM and DBAM files, this VDP must contain the address of the first Index volume.)
2. Place the address of the first word of this VDP in the appropriate field of the FDP.

For example, if you open the above-mentioned PIGEON file at runtime, you don't need to redefine each of its volumes. You simply build a new Volume Table which contains only the address of the name of the first volume you defined when you created PIGEON. The system will then refer to its own PIGEON.VL file to obtain the volume definition information it needs to open the file. Naturally, to access all this information, you have to provide (in the runtime FDP) the address of the new, single-entry VDP in the new Volume Table.

For SAM tape, line printer, and interactive terminal files, you must build a Volume Table each time you open the file. To do so:

1. Place one VDP in the Table for each volume of the file, in the order in which you want the system to process them.
2. Specify (in the file's FDP) the number of VDPs in the Volume Table. (Terminal, printer, and unlabeled tape files will usually be single-volume files.)
3. In the file's FDP, supply the address of the first word of the first VDP in the Table.

NOTE: While you must define all the volumes of a file when you create it, the actual physical volumes need not be on-line at file creation. Furthermore, once you have opened a file, the system no longer needs the Volume Table, and you can use the space it occupied for processing.

**General Processing Packet
(SAM and RAM files only)**

You must supply the address of a General Processing Packet whenever you issue a SAM or RAM processing request. This type of packet contains the information

the INFOS system needs to successfully process the call, as well as the means for the system to return general status information to you. Figure II-6-3 shows a full General Processing Packet.

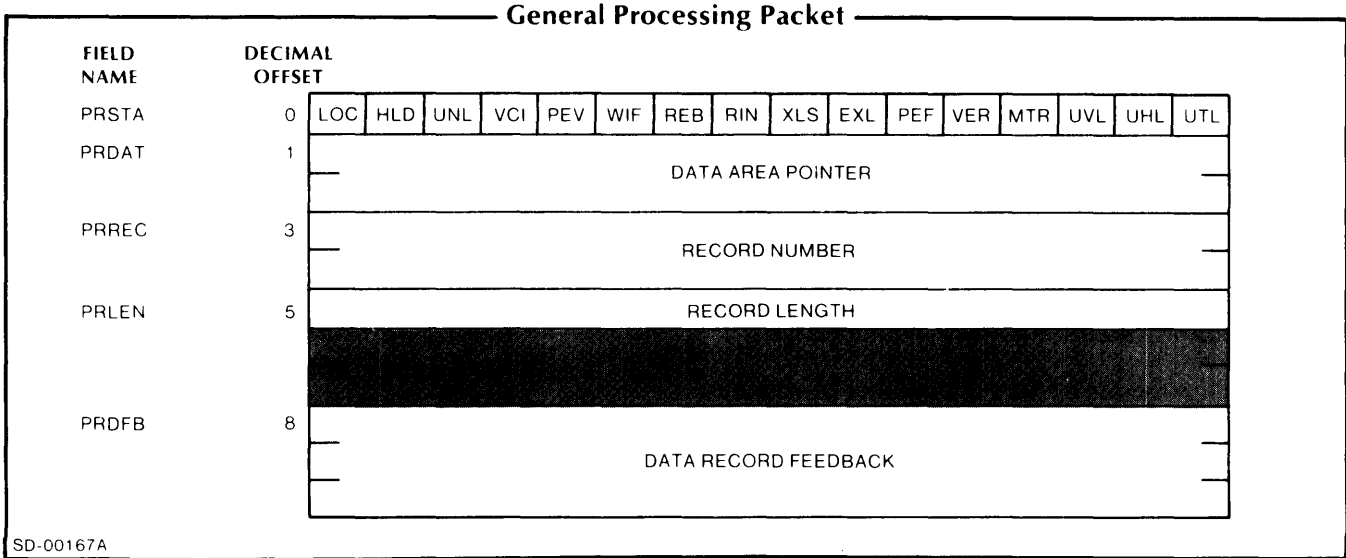

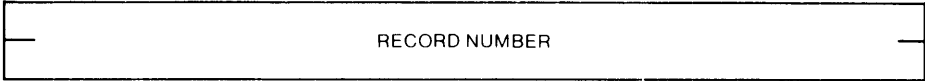
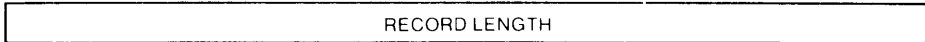



Figure II-6-3

Table II-6-3. General Processing Packet

Field Name/Description	Specified As	Field Value	Default Means
PRSTA	0	LOC HLD UNL VCI PEV WIF REB RIN XLS EXL PEF VER MTR UVL UHL UTL	
Status Flags			
Lock Record	PFLOC	1	No record lock
Hold Record	PFHLD	1	Processing request not held in queue
Unlock Record	PFUNL	1	Do not unlock previously locked record
Volume Change Indicator	PFVCI	1	Not applicable ¹
Physical End-of-Volume ²	PFPEV	1	Not applicable ¹
Write Immediately if Modified ³	PFWIF	1	Buffer not flushed until needed
Record Exceeds Block Size	PFREB	1	Not applicable ¹
Read Inhibit	PFRIN	1	Appropriate block moved to a buffer and record moved to block
Transfer Length Short	PFXLS	1	Not applicable ¹
Excessive Transfer Length ⁴	PFEXL	1	Not applicable ¹
Physical End-of-File	PFPEF	1	Not applicable ¹
Verification Failure ⁵	PFVER	1	Not applicable ¹
Magnetic Tape Control Error ⁶	PFMTR	1	Not applicable ¹
User Volume Label Processed	PFUVL	1	Not applicable ¹
User Header Label Processed	PFUHL	1	Not applicable ¹
User Trailer Label Processed	PFUTL	1	Not applicable ¹

Table II-6-3. General Processing Packet (concluded)

Field Name/Description	Specified As	Field Value	Default Means
PRDAT Data Area Pointer	PRDAT	1 	No data transferred
PRREC Record Number ⁷	PRREC	3 	Required
PRLEN Record Length	PRLEN	5 	Fixed length records
PRDFB Data Record Feedback	PRDFB	8 	Not applicable ¹

Footnotes:

1. This parameter indicates information returned by the system. For example, if the system has to close one volume and open another to find a requested record, it will tell you that it did so by setting the PFVCI bit to 1. Therefore, defaults are not applicable for these bits. A 0 in this position merely indicates no change in that particular parameter's status.
2. The system will only return this status after it has processed the last record in the last block.
3. Only applicable to Write requests in a RAM file. For all other SAM and RAM requests, set this bit to 0.
4. Applies only to SAM tape files. The system will return the actual length of the block transferred in PRLEN.
5. The system will set this flag only if 1) you set F1RAW=1 in the file's FDP and 2) the system could not successfully complete the read-after-write verification cycle in this or a previous processing request. If this bit is set (i.e., if PFVER = 1), it usually indicates a hardware problem.
6. If PFMTR = 1, it means that the system has encountered an I/O error while attempting a magnetic tape control request. Since this may indicate a hardware problem, you should decide whether or not to continue processing.
7. For all SAM requests, specify -1 in the first word of this field.

**Extended Processing Packet
(ISAM and DBAM files only)**

Similar to the General Processing Packet, an Extended Processing Packet contains the information which the INFOS system needs to successfully perform an ISAM

or DBAM processing request. It also provides the means for the system to return general status information. Figure II-6-4 shows a full Extended Processing Packet.

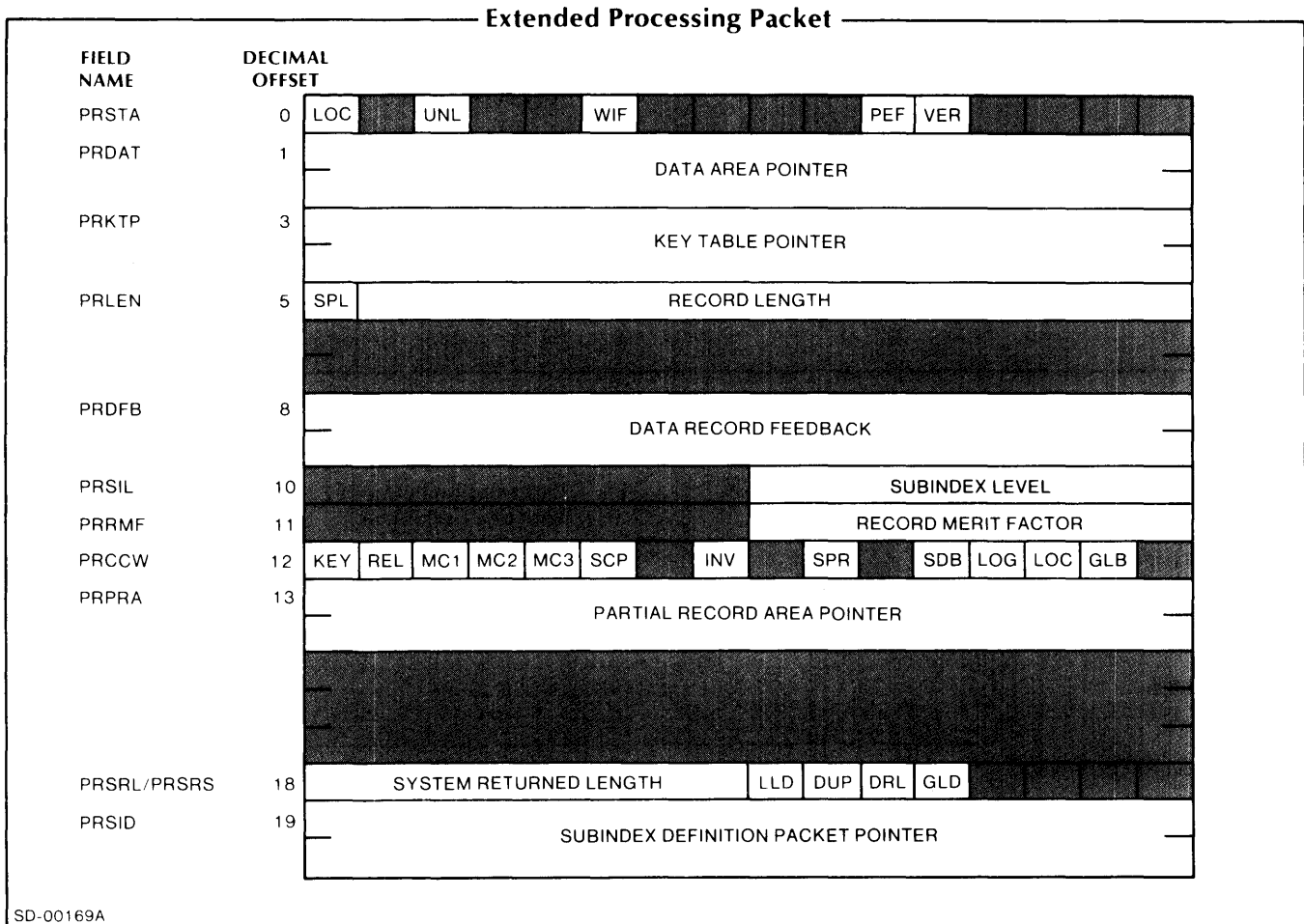


Figure II-6-4

Table II-6-4. Extended Processing Packet

Field Name/Description	Specified As	Field Value	Default Means
PRSTA	0		
Status Flags			
Lock Record ¹	PFLOC	1	No record lock
Unlock Record ¹	PFUNL	1	Record not unlocked
Write Immediately if Modified	PFWIF	1	Buffer not flushed until needed
Physical End-of-File Verification Failure	PFPEF PFVER	1 1	Not applicable ² Not applicable ²
PRDAT	1		
Data Area Pointer ³	PRDAT	Address of first byte of your data area	Required (unless you specify Suppress Database)
PRKTP	3		
Key Table Pointer ⁴	PRKTP	Address of first word of Key Table	Relative Motion
PRLEN	5		
Specified Length ⁵ Record Length	PRSPL PRLEN	1 For Read: Length of record transferred For Write or Rewrite: Length of record to be (re)written	Transfer entire record Not applicable ⁶ Required (unless you specify Suppress Database)

Footnotes:

1. You can Lock or Unlock an index entry, a data record, or both by specifying CCLOC=1 and/or CCGLB=1 at the same time you specify PFLOC or PFUNL = 1.
2. This parameter indicates information returned by the system. That is, if this field is filled, it means that the system has encountered the status indicated. Therefore, defaults are not applicable. A 0 in this bit merely indicates no change in that parameter's status.
3. Note that if either CCSDB=1 (Suppress Database) or CCINV=1 (Write Inverted), the system will ignore the contents of PRDAT.
4. You must also specify Keyed Access in CCKEY (word PRCCW) or the system will ignore the contents of PRKTP.
5. You may set PRSPL = 1 and specify in PRLEN the number of bytes you want the system to transfer for a Read request. Upon completion of the Read, the System will return the number of bytes transferred to PRLEN.
6. The system will return this information as indicated. Therefore, just leave these words blank.

Table II-6-4. Extended Processing Packet (continued)

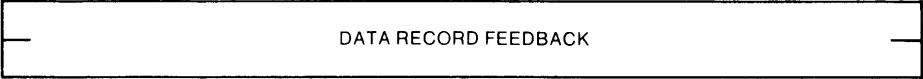



Field Name/Description	Specified As	Field Value	Default Means
PRDFB Data Record Feedback	8 PRDFB		Not applicable ⁶
PRSIL Subindex Level	10 PRSIL		Not applicable ⁶
PRRMF Record Merit Factor	11 PRRMF		Required when you specify Optimized Record Distribution
PRCCW Command Control Flags	12		
Keyed Access ⁷ Relative to Current Position	CCKEY CCREL	1 1	Relative access Keyed Access only
Motion Control Field	CCMC1/2/3		Required for Relative or Combined Access
Forward	CCFWD	000	
Backward	CCBAK	001	
Down	CCDWN	010	
Down and Forward	CCDWF	011	
Up and Forward	CCUFW	100	
Up and Backward	CCUBK	101	
Up	CCUP	110	
Static	CCSTA	111	
Set Current Position	CCSCP	1	No new current position established
Write Inverted ⁸	CCINV	1	No new index entry written
Suppress Partial Record	CCSPR	1	Partial record read or written
Suppress Database	CCSDB	1	Database record read or written
Logical Delete ⁹	CCLOG	1	No logical delete
Local Selector ¹⁰	CCLOC	1	No operation performed on index entry
Global Selector ¹⁰	CCGLB	1	No operation performed on data record

Table II-6-4. Extended Processing Packet (concluded)

Field Name/Description	Specified As	Field Value	Default Means									
<p>PRPRA 13</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 5px auto;">PARTIAL RECORD AREA POINTER</div> <p>Partial Record Area Pointer</p>	PRPRA	Address of first byte of your Partial Record Area	Required (unless you specify Suppress Partial Records)									
<p>PRSRL/ PRSRS 18</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 5px auto;"> <table style="border-collapse: collapse;"> <tr> <td style="border: 1px solid black; padding: 2px;">SYSTEM RETURNED LENGTH</td> <td style="border: 1px solid black; padding: 2px;">LLD</td> <td style="border: 1px solid black; padding: 2px;">DUP</td> <td style="border: 1px solid black; padding: 2px;">DRL</td> <td style="border: 1px solid black; padding: 2px;">GLD</td> <td style="background-color: black; width: 20px; height: 15px;"></td> <td style="background-color: black; width: 20px; height: 15px;"></td> <td style="background-color: black; width: 20px; height: 15px;"></td> <td style="background-color: black; width: 20px; height: 15px;"></td> </tr> </table> </div> <p>System Returned Length</p> <p>System Returned Status</p> <p>Local Logical Delete</p> <p>Duplicate Key¹¹</p> <p>Data Record Locked¹²</p> <p>Global Logical Delete</p>	SYSTEM RETURNED LENGTH	LLD	DUP	DRL	GLD					<p>PRSRL</p> <p>PRSRS</p> <p>SRLLD</p> <p>SRDUP</p> <p>SRDRL</p> <p>SRGLD</p>	<p>Length (in bytes) of partial record accessed</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p>	<p>Not applicable⁶</p> <p>Not applicable²</p> <p>Not applicable²</p> <p>Not applicable²</p> <p>Not applicable²</p>
SYSTEM RETURNED LENGTH	LLD	DUP	DRL	GLD								
<p>PRSID 19</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 5px auto;">SUBINDEX DEFINITION PACKET POINTER</div> <p>Subindex Definition Packet Pointer</p>	PRSID	Address of first word of Subindex Definition Packet for intended new subindex	No new subindex being defined									

Footnotes (concluded)

7. You must set this flag to 1 for all ISAM and DBAM Write or Physical Delete operations.
8. On a Write or Rewrite request, set CCINV = 1 and specify the address of the existing database record in PRDAT.
9. Available for index entries or database records, or both, depending on how you use CCLOC and CCGLB.
10. Note that you can set either CCLOG = 1 or CCGLB = 1, or both (to operate on both the index entry and its associated data record).
11. The system returns duplicate keys' occurrence numbers in offset KDDKO of the Key Table.
12. The system will only set this flag for a Retrieve Status request; all other requests that attempt to access a locked data record will receive error code 403 (data record locked).

Key Definition Packet

Each Key Definition Packet describes a single key, and to use Keyed or combined access, you must concatenate the Key Definition Packets for each request into a Key Table. We'll discuss the Key Table in the next section of this chapter, but first let's look at the components of Key Definition Packets, as illustrated in Figure II-6-5.

Note that, as usual, the default for single-bit parameters is 0, but you cannot default any of the *full word* parameters in this packet.

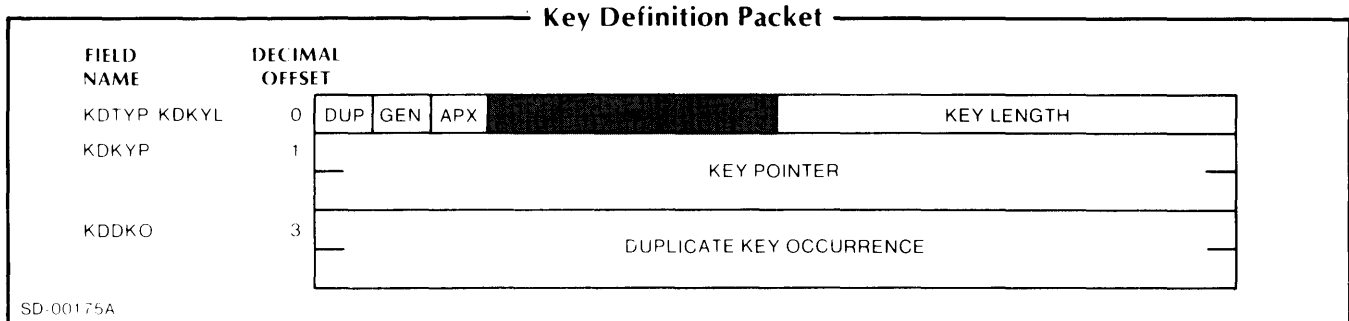


Figure II-6-5

Table II-6-5. Key Definition Packet

Field Name	Description	Specified As	Field Value	Default Means
KDTYP	Key Type Flags	KDTYP		
	Duplicate Keys ¹	KTDUP	1	No duplicate keys read or written
KDKYL	Generic Key ²	KTGEN	1	No generic search
	Approximate Key ²	KTAPX	1	No approximate search
KDKYP	Key Length	KDKYL	Length of key described (between 1 and 255 bytes)	Required
KDKYP	Key Pointer	KDKYP	Address of first byte of key described	Required
KDDKO	Duplicate Key Occurrence	KDDKO	For Write: occurrence number written For all other operations: occurrence number of key processed	Occurrence number of 0 used. (Occurrence number is returned here on a Write request)

Footnotes:

1. If you set KTDUP=1 for a Write request, then specify 0 in each of the four bytes in KDDKO. If you set KTDUP=1 for any other requests, you may specify in KDDKO either the occurrence number of the duplicate key or 0 (for the original key).
2. You must set this bit to 0 for all write operations.

Key Tables

Whenever you use Keyed access in ISAM or DBAM, you must supply the address of the first word of a Key Table. This table is nothing more than a contiguous sequence of one or more Key Definition Packets. The INFOS system applies each key packet at the subindex level indicated by the packet's relative position in the table. That is, it applies the first entry in the table at subindex level 0, the second key at subindex level 1, and so forth. Therefore, the last nonzero key in the table must be the entry that you want the system to

access in order to process your current request. Figure II-6-6 shows a sample Key Table whose entries point to keys at the subindex levels indicated.

Note that a Key Table for an ISAM request will contain only one Key Definition Packet (because there is only one subindex level in an ISAM index structure), but a Key Table for a DBAM request may contain many packets.

Finally, you terminate each Key Table by specifying a zero length key.

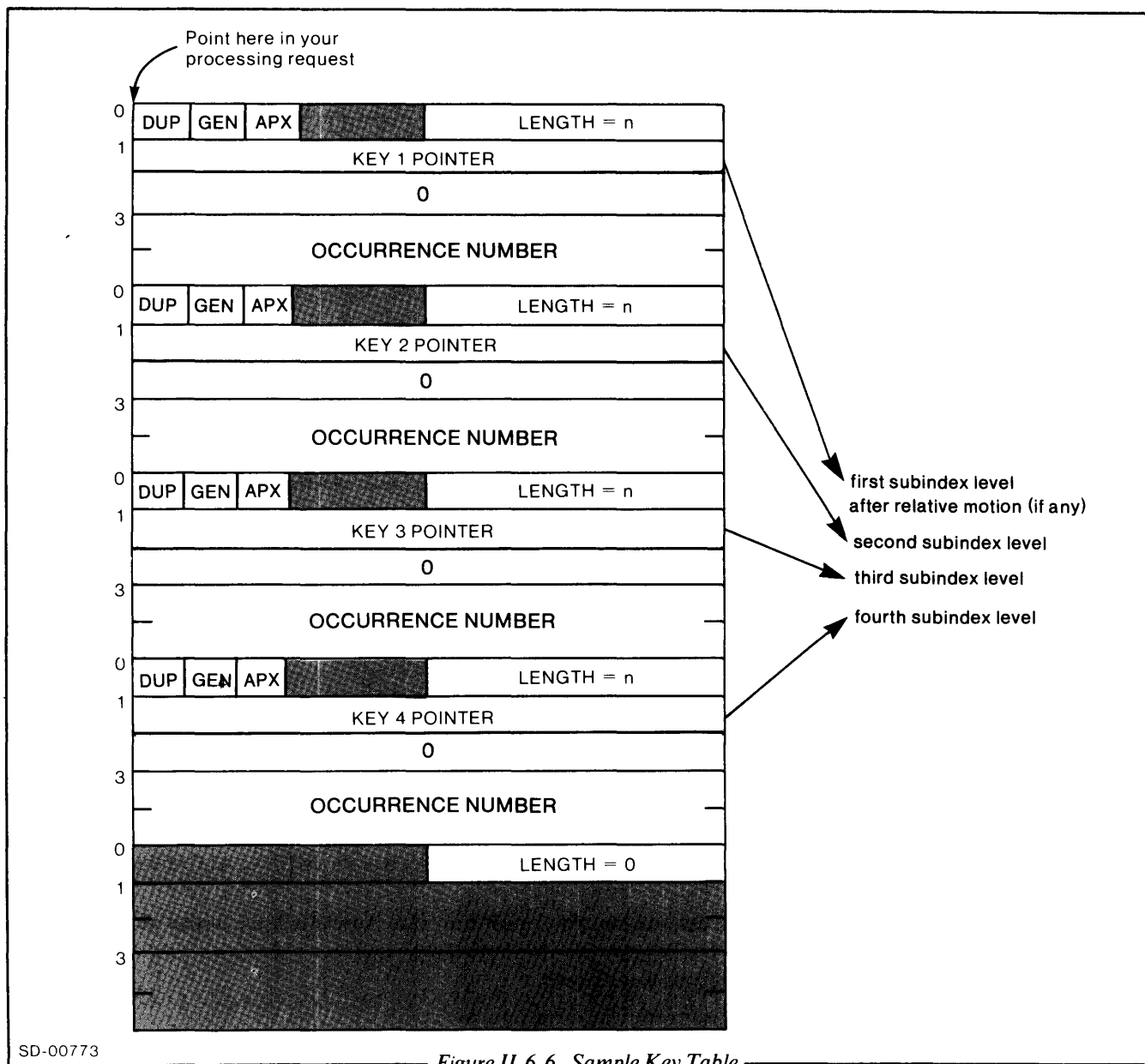


Figure II-6-6. Sample Key Table

Subindex Definition Packet

The Subindex Definition Packet describes the subindex you wish to create. You need to build one of these packets each time you issue a Define Subindex

request, and the Extended Processing Packet for that request must contain the address of the first byte of the Subindex Definition Packet. Figure II-6-7 illustrates the components of a Subindex Definition Packet.

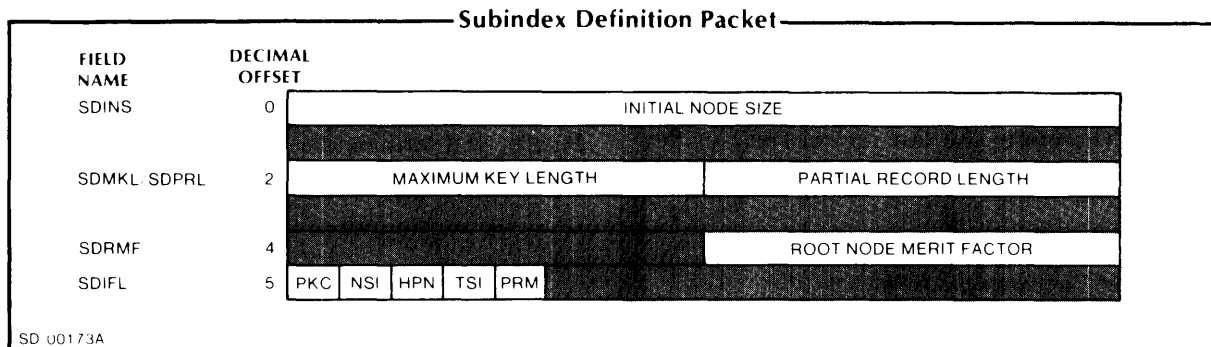


Figure II-6-7

Table II-6-7. Subindex Definition Packet

Field Name	Description	Specified As	Field Value	Default Means
SDINS	Initial Node Size ¹	SDINS	Initial size (in bytes) of the Root Node for the subindex you're defining	Root node size of 506 bytes
SDMKL	Maximum Key Length	SDMKL	Length of longest key in new subindex	System derived maximum key length ⁴
SDPRL	Partial Record Length	SDPRL	Length of longest partial record	No partial records allowed
SDRMF	Root Node Merit Factor ²	SDRMF	Merit factor of Root Node of new subindex	Not using Optimized Distribution
SDIFL	Index Flags			
	Perform Key Compression	IXPKC	1	No key compression
	No Subindexes	IXNSI	1	Subindexes allowed under one being defined
	High Priority Node	IXHPN	1	No high priority for Root Node of this subindex
Temporary Subindex ³	IXTSI	1	This is a permanent subindex	
Permanent Data Records ³	IXPRM	1	Data records linked only to this subindex disappear at file close	

Footnotes

1. You can calculate this size by using either the formulas explained in Appendix B or the INDEXCALC utility, described in the *INFOS Utilities User's Manual*.
2. You must specify a merit factor if you are using Optimized Distribution. (See Appendix B for further details on merit factors.)
3. You may only set this bit when you're defining subindex level 0.
4. The system will derive the maximum key length according to the formula:

$$\text{maxkeylength} = \left(\frac{\text{initial node size} - 36}{3} \right) - (10) - (\text{partial record length})$$

or, if you allowed subindexes:

$$\text{maxkeylength} = \left(\frac{\text{initial node size} - 36}{3} \right) - (14) - (\text{partial record length})$$

Point Processing Packet

You will only need a Point Processing Packet when you issue a Point request for a SAM disk file. Figure II-6-8 illustrates this packet's components.

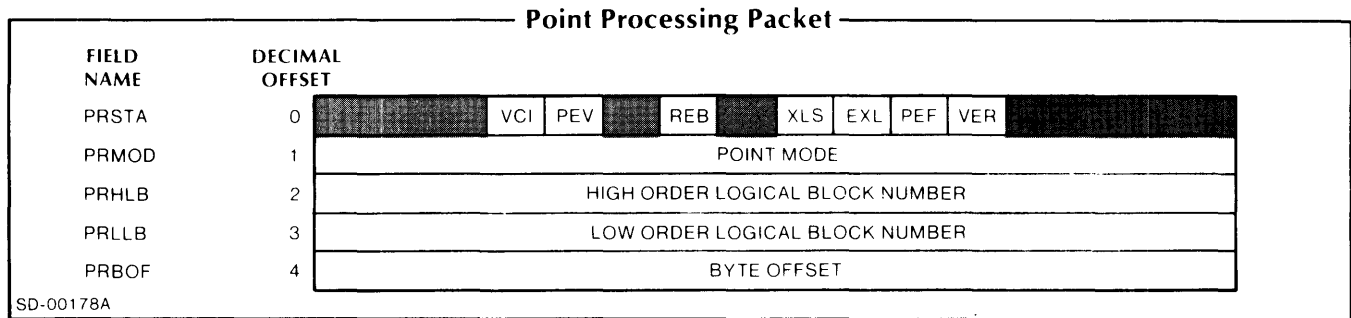


Figure II-6-8

Table II-6-8. Point Processing Packet

Field Name	Description	Specified As	Field Value	Default Means
PRSTA	Status Flags			Not applicable ¹
	Volume Change Indicator	PFVCI	1	
	Physical End-of-Volume	PFPEV	1	
	Record Exceeds Block Size	PFREB	1	
	Transfer Length Short	PFXLS	1	
	Excessive Transfer Length ²	PFEXL	1	
	Physical End-of-File	PFPEF	1	
	Verification Failure ³	PFVER	1	
PRMOD	Point Mode			End of file
	Relocate to End of File	PMEOF	0	
	or Relocate to Record Specified	PMLBN	1	
PRHLB	High Order Logical Block Number	PRHLB	32-bit number of the block containing the desired record	Required when you specify PMLBN
PRLLB	Low Order Logical Block Number	PRLLB		
PRBOF	Byte Offset	PRBOF	Starting location of record in block specified in PRHLB and PRLLB	No offset from beginning of block

Footnotes:

1. These parameters indicate information returned by the system. For example, if the system has to close one volume and open another to find a requested record, it will set PFVCI=1. Therefore, defaults are not applicable for these parameters. A 0 in any of these positions merely indicates no change in that parameter's status.
2. The system will return the actual length of the block transferred in PRLBN of the FDP.
3. The system will set this flag only if: 1) you set F1RAW=1 in the FDP, and 2) the system could not successfully complete the Read-After-Write verification cycle in this or a previous request. If this bit is set (i.e., if PFVER = 1), it usually indicates a hardware failure.

Link Subindex Processing Packet

You only need to build a Link Subindex Processing Packet when you want to use a Link Subindex request. This packet describes the key you wish to link to a given, previously-linked subindex. The "source key"

is defined as the index entry which already exists and which owns the subindex you want to share. The "destination key" is the one you wish to link to the source key's subindex. It cannot own a subindex prior to this request. Figure II-6-9 illustrates this packet's components.

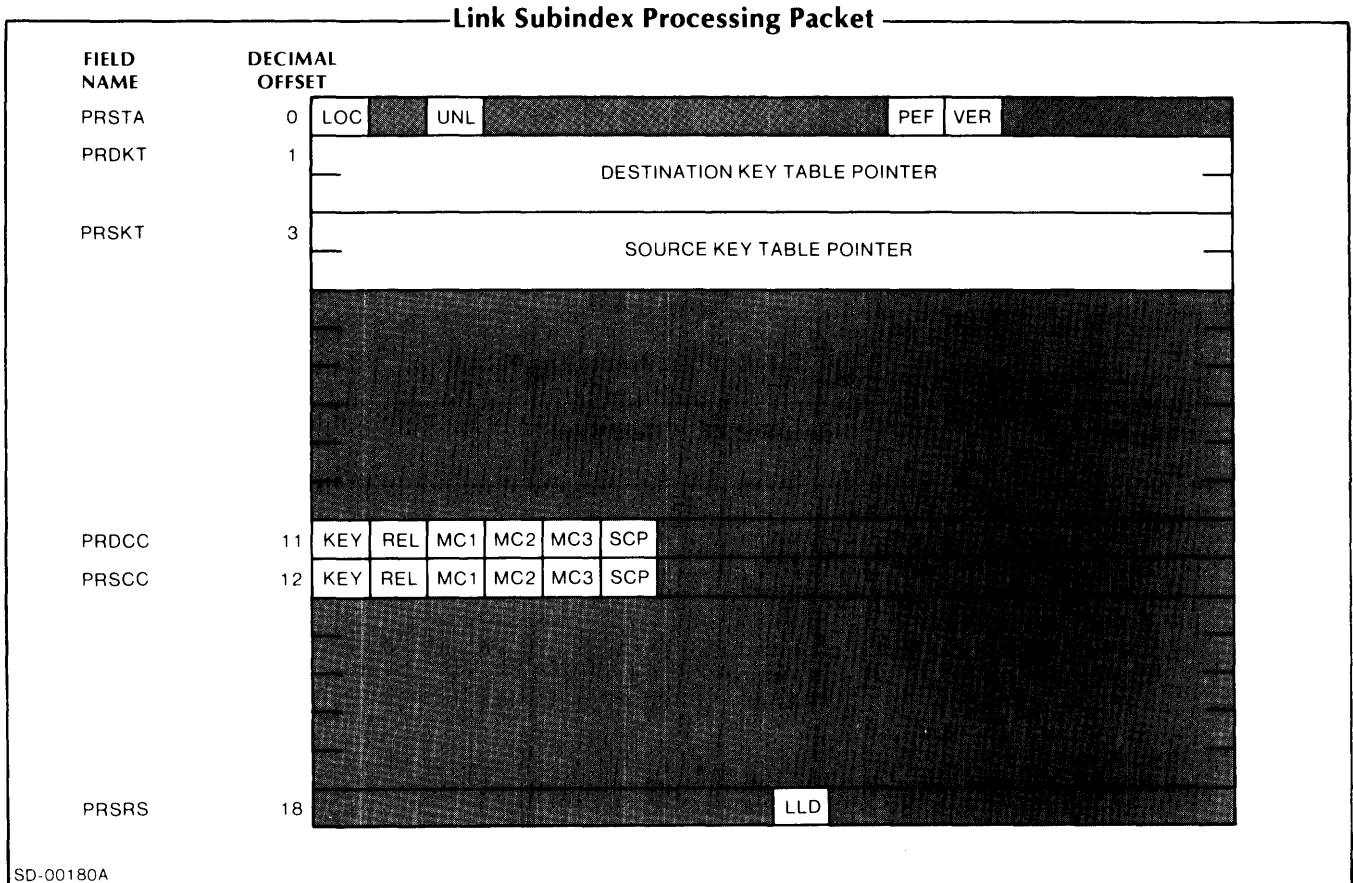


Figure II-6-9

Table II-6-9. Link Subindex Processing Packet

Field Name/Description	Specified As	Field Value	Default Means
PRSTA	0		
Status Flags			
Lock ¹	PFLOC	1	Entry not locked
Unlock ¹	PFUNL	1	Entry not unlocked until file closed
Physical End-of-File ²	PFPEF	1	Physical end of file not reached
Verification Failure ²	PFVER	1	Read-After-Write verification complete
PRDKT	1		
Destination Key Table Pointer	PRDKT	Address of first word of Destination Key Table	Relative motion used to access destination key
PRSKT	3		
Source Key Table Pointer	PRSKT	Address of first word of Source Key Table	Relative motion used to access source key

Footnotes:

1. You can lock or unlock either the source or destination key, or both. However, if you do choose to lock, you must also set CCLOC = 1 in PRDCC or PRSCC (or both). You may also lock (or unlock) the source and destination entirely in the same request.
2. The system will set this status bit only when applicable.

Table II-6-9. Link Subindex Processing Packet (concluded)

Field Name/Description	Specified As	Field Value	Default Means														
PRDCC and PRSCC	11 12	<table border="1"> <tr> <td>KEY</td> <td>REL</td> <td>MC1</td> <td>MC2</td> <td>MC3</td> <td>SCP</td> <td></td> </tr> <tr> <td>KEY</td> <td>REL</td> <td>MC1</td> <td>MC2</td> <td>MC3</td> <td>SCP</td> <td></td> </tr> </table>	KEY	REL	MC1	MC2	MC3	SCP		KEY	REL	MC1	MC2	MC3	SCP		
KEY	REL	MC1	MC2	MC3	SCP												
KEY	REL	MC1	MC2	MC3	SCP												
Command Control Words																	
Keyed Access	CCKEY	1	Relative motion used														
Relative to Current Position ³	CCREL	1	Keyed access used														
Motion Control Field ³	CCMC1/2/3		Required for Relative motion														
Forward	CCFWD	000															
Backward	CCBAK	001															
Down	CCDWN	010															
Down and Forward	CCDFW	011															
Up and Forward	CCUFW	100															
Up and Backward	CCUBK	101															
Up	CCUP	110															
Static	CCSTA	111															
Set Current Position ⁴	CCSCP	1	Current position stays on source key														
Local Selector ⁵	CCLOC	1	No lock or unlock														
PRSRs	18	<table border="1"> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>LLD</td> <td></td> </tr> </table>							LLD								
						LLD											
Status Flag																	
Local Logical Delete	SRLLD	1	Not applicable ²														

Footnotes (concluded)

- If you set CCREL = 1, you must also indicate a direction in the motion control field (CCMC1/2/3).
- If you set CCSCP = 1 in both the source and destination control words, your current position will be on the destination key at the completion of the request.
- If you set CCLOC = 1 in either Command Control Word, you must also set either PFLOC = 1 or PFUNL = 1.

Volume Initialization Packet

You need to build a Volume Initialization Packet for each volume of a labeled tape file which you initialize via the .IINFOS system call. (See Appendix A for volume initialization procedures.) Figure II-6-10 illustrates this packet's components.

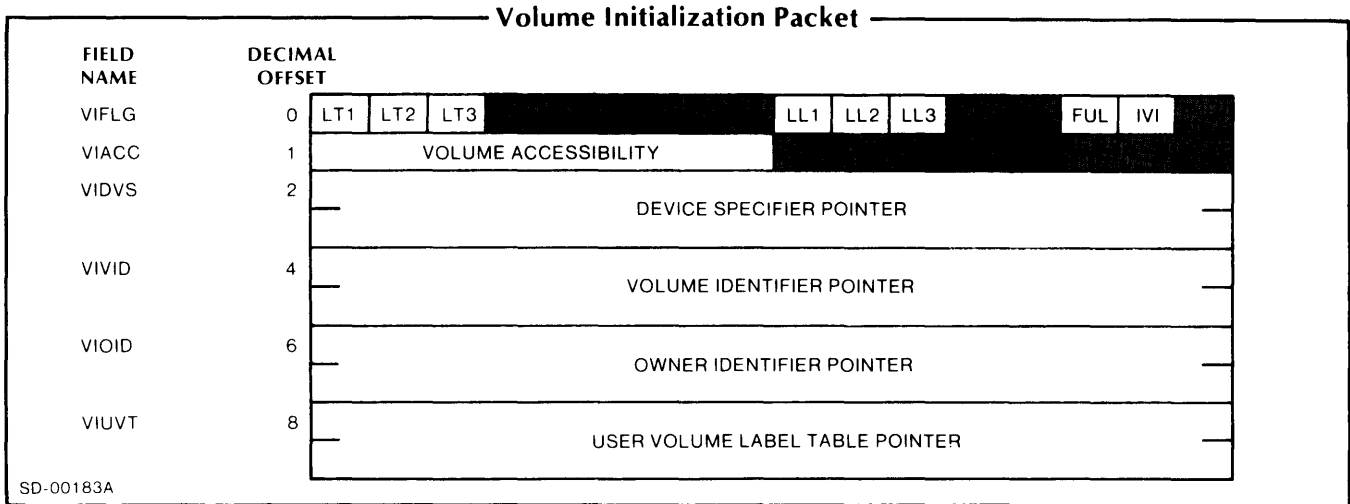


Figure II-6-10



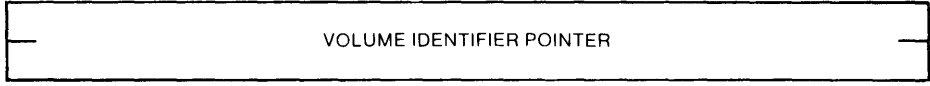
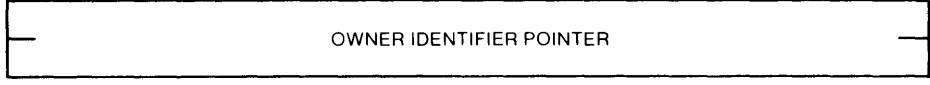

Table II-6-10. Volume Initialization Packet

Field Name/Description	Specified As	Field Value	Default Means
VIFLG	0	LT1 LT2 LT3 ██████████ LL1 LL2 LL3 ██████████ FUL IVI	
Volume Initialization Flags			
Label Type	VFLT1/2/3		ANSI Labels
ANSI Labels	TCANS	000	
IBM Labels	TCIBM	001	
Label Level	VFLL1/2/3		Required
Level 1	TCLV1	001	
Level 2	TCLV2	010	
Level 3	TCLV3	011	
Full Initialization	VFFUL	1	Partial initialization
Ignore User Volume Identifier ¹	VFIVI	1	See footnote 1

Footnotes:

1. If VFFUL = 1, the system ignores this bit. If VFFUL = 0 and VFIVI = 0, the system compares the volume identifier with the Vol1 label. If they match, the system equates the device specifier with the volume identifier so that you can refer to the volume by its volume name, rather than by its device name. In other words, you can call the volume mounted on MT0 "PAYROLL" instead of "MT0". If the device specifier and the volume identifier don't match, the system takes the .IINFOS call error return. If VFFUL = 0 and VFIVI = 1, the system simply equates the volume identifier in the Vol1 label to the device specifier without comparing them.

Table II-6-10. Volume Initialization Packet (concluded)

Field Name/Description	Specified As	Field Value	Default Means
VIACC Volume Accessibility ²	VIACC	1 	Required when VFFUL = 1
VIDVS Device Specifier Pointer ³	VIDVS	2 	Required
VIVID Volume Identifier Pointer	VIVID	4 	Partial initialization
VOID Owner Identifier Pointer ⁴	VOID	6 	Owner ID not returned or written
VIUVT User Volume Label Table Pointer ⁵	VIUVT	8 	No User Volume Labels written or returned

Footnotes (continued)

2. This parameter exists solely to allow you to use tapes interchangeably with other systems. The INFOS system does not restrict access based on the contents of this field.
3. The device specifier refers to the number of the tape drive on which this volume is mounted and can be in the range MT0 to MT7. You must set up the device specifier in memory such that its last character is null.
4. For ANSI labels, the owner identifier memory area can be up to 14 bytes long; for IBM labels, it can be up to 10 bytes long. In both cases, you must terminate the identifier with a null character. On a full initialization, the system will write the identifier to the Vol1 label; on a partial, the system will return the identifier recorded in the Vol1 label.
5. Applicable only to files with ANSI level 3 labels. For all other label types and levels, specify -1 in the first word of this field.

Magnetic Tape Control Processing Packet

You need to build this packet only when you want to issue a Mag Tape Control request. This packet allows you to position the tape during processing. In it, you *must* specify a control function code (i.e., a mnemonic

which specifies the function you wish to perform). If you choose Read, Write, Space Forward, or Space Backwards, you must also supply the number of words or records, and (if you are transferring data) the address of the area in memory to or from which you want the data transferred. Figure II-6-11 illustrates this packet's parts.

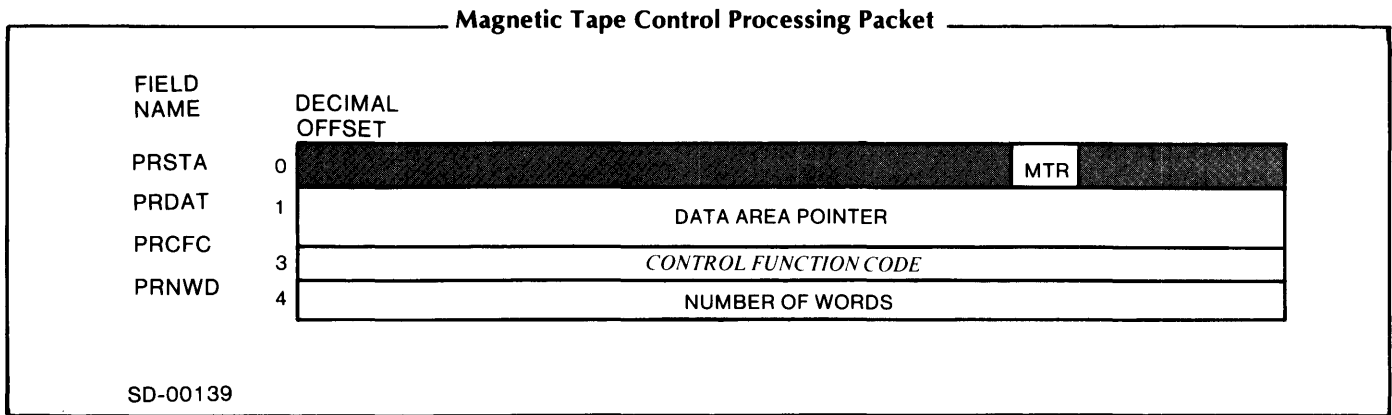


Figure II-6-11

Table II-6-11. Magnetic Tape Control Processing Packet

Field Name/Description	Specified As	Field Value	Default Means
PRSTA Status flags I/O Error ¹	PRSTA PFMTR	1	No error has occurred
PRDAT Data Area Pointer	PRDAT	Address of first word of your data area	No data being transferred (required for all data transfers)
PRCFC Control Function Codes Space forward file Space backward file Read a record Write a record Write an end-of-file Rewind Space forward records Space backward records Erase Mag Tape Status Registers ¹	PRCFC MCSFF MCSBF MCRD MCWRT MCWRF MCREW MCSFR MCSBR MCERS		Required
PRNWD Number of Words	PRNWD	Number of words or records (on an error, system returns here the actual transfer length or number of records spaced over)	Required if you set MCRD, MCWRT, MCSFR, or MCSBR

Footnote

1. The system sets this flag only if an I/O error occurs while processing a request. You must decide whether to continue processing or not.

Now that you know what all the different packets look like, read Chapter 7 to find out how to build them using the Macroassembler.

End of Chapter

Chapter 7

How to Use the Macroassembler with the INFOS System

The tape which contains your basic INFOS system also contains a set of Macros which function as the Assembly language interface. You can access these Macros to build the packets you need simply by including the appropriate macro calls in your source code.

Keyword Parameters

For each call, there is a list of Keyword Parameters which you must define immediately before you issue the call and, in most cases, a list of parameters associated with the call itself. You must define some of the Keyword Parameters, but many also have a default value. Note that you do not need to include those Keyword Parameters for which you're using the default value in the list of assignments you make just before the call. That is, you can either include a specific value for a parameter, or, if you're using the default value, you can omit the assignment.

Furthermore, you must include a Keyword Parameter list just before every macro call which you include in your source code. The only exception to this rule is when you want to use the same values and defaults for a call which you are using more than once in your program. In this case, you should just include **RETAIN = Y** in the Keyword Parameter list for the first occurrence of the call. This will tell the system to retain those values until you code **RETAIN = N**. For example, if you want to build several identical FDPs in the course of your program (that is, you want to use the same parameter values for each one), just include **RETAIN = Y** in the parameter list you code for the first **BLDFDP** (Build FDP) macro call. Then, the next time you want to build an FDP, you only need to use the call **BLDFDP**. The system will automatically use the parameter values you specified earlier.

Finally, note that the default for Keyword Parameters with a YES/NO response is NO.

You specify the parameters associated with the macro calls themselves differently from the Keyword Parameters. You must include in the call every element in the parameter list. If you do not wish to specify a value for an optional parameter, you must replace the parameter with the word **DEFAULT** in the proper sequence in the call.

In the lists that follow, note that slashes between parameter values for Keywords (e.g., **Y/N/DEFAULT**) indicate that you must select one of the given values. Also, we have included in parentheses the name of each Keyword as it appears in the charts in the previous chapter so that you may refer to them for default values or whatever. Finally, the meanings of the values that you may assign to Keyword Parameters are fairly easy to interpret, but here they are anyway for your convenience:

General

Specify	When you mean:
(N)	numeric of up to full word
(byte)	numeric of up to one byte or one character (8-bit ASCII)
NO/N	no
YES/Y	yes
DEFAULT	If you wish to use the default value, this keyword parameter need not appear in an equivalence line.
USER	User-supplied option desired.
RETAIN	If answered "YES", the system will retain the current Keyword values for that call until the end of the program, or until it encounters "RETAIN = NO".

Access Method

Specify	When you mean:
SAM	Sequential Access Method
RAM	Random Access Method
ISAM	Indexed Sequential Access Method
DBAM	Data Base Access Method

Formatting Records

Specify	When you mean:
F	Fixed length
V	Variable length
U	Undefined length
D	Data-sensitive

I/O Mode

Specify	When you mean:
I	Input
O	Output
UPD	Update
C	Create update

Label Types

Specify	When you mean:
ANSI	ANSI
IBM	IBM

Translation Specifiers

Specify	When you mean:
NONE	No translation
ASCII	ASCII
EBCDIC	EBCDIC

Finally, be careful when you're specifying Keywords. Even the interface code's extensive error-checking will not catch misspelled Keywords; these merely become unreferenced equates.

Macro Functions

BLDFDP

To construct an INFOS File Definition Packet.

Access Method:

Used for all access methods.

Keyword Parameters:

Assignment Line		Actual Parameter
UNBLOCKED	= Y/N/DEFAULT	(F1UBR)
ACCESS	= SAM/RAM/ISAM/ DBAM/DBASE*	(F1AM1-2)
FORMAT	= F/V/U/D	(F1FT1-2)
VERIFY	= Y/N/DEFAULT	(F1RAW)
OVERWRITE	= Y/N/DEFAULT	(F1OVR)
EXCLUSIVE	= Y/N/DEFAULT	(F1EXF)
MODE	= I/O/UPD/C	(F1PM1-2)
REWRITE	= Y/N/DEFAULT	(F1RER)
INVERT	= Y/N/DEFAULT	(F1INV)
ONLYINDEX	= Y/N/DEFAULT	(F2OTI)
TEMPINDEX	= Y/N/DEFAULT	(IXTSI)
SPACEMGMT	= Y/N/DEFAULT	(F2SPM)
OPTIMIZE	= Y/N/DEFAULT	(F2ORD)
DBSPEC	= Y/N/DEFAULT	(F2FDP)
BLOCKSIZE	= (n)/DEFAULT	(FDBLK)
BUFFERS	= (n)/DEFAULT	(FDBUF)
RECORDSIZE	= (n)/DEFAULT**	(FDLEN)

* When using BLDFDP for a database file, specify the parameter "ACCESS = DBASE". This alerts the interface diagnostic routines to expect Variable length records.

** When you use BLDFDP for ISAM/DBAM files, use this field to specify the number (n) of index levels, not record size.

Licensed Material - Property of Data General Corporation

Assignment Line		Actual Parameter
VOLUMES	= (n)/DEFAULT	(FDNVD)
FDLBTYP	= ANSI/IBM/ DEFAULT	(TCLT1-3)
FDLBLEV	= (n)/DEFAULT	(TCLL1-3)
SELTRAN	= Y/N/DEFAULT	(TCSFT)
INTRAN	= NONE/USER/ DEFAULT/ASCII/ EBCDIC	(TCIT1-4)
OUTTRAN	= NONE/USER/ DEFAULT/ASCII/ EBCDIC	(TCOT1-4)
EXPMONTH	= (n)/DEFAULT	
EXPDAY	= (n)/DEFAULT	(FDEXP)
EXPYEAR	= (n)/DEFAULT	
SEQUENCE	= (n)/DEFAULT	(FDSEQ)
FDGNRAT	= (n)/DEFAULT	(FDGEN)
FILACC	= (byte)/DEFAULT	(FDACC)
BUFFOFF	= (n)/DEFAULT	(FDIDO)
FDMINOD	= (n)/DEFAULT	(FDINS)
FDMAXKY	= (byte)/DEFAULT	(FDMKL)
FDPRECLEN	= (byte)/DEFAULT	(FDPRL)
FDRTMERIT	= (byte)	(FDRMF)
COMPRESS	= Y/N/DEFAULT	(IXPKC)
SUBINDEX	= Y/N/DEFAULT	(IXNSI)
DISHIEREP	= Y/N/DEFAULT	(F2DHR)
HIPRINOD	= Y/N/DEFAULT	(IXHPN)
PERMREC	= Y/N/DEFAULT	(IXPRM)

Call:

BLDFDP vtp, uit, uot, dsd, fsi, sft, dbp.

where the call parameters are:

Definition	Actual Parameter
vtp	Address of Volume Definition Table (FDVTP)
uit	Optional address of User Input Translate Table (FDUIT)
uot	Optional address of User Output Translate Table (FDUOT)
dsd	Optional address of Data Sensitive Delimiter Table (FDDSD)
fsi	Optional byte address of File Set ID (FDFS ¹ SI)
sft	Optional address of Selective File Translation Control (FDSFT)
dbp	Optional address of Database Definition or Database Name byte address (FDDBP)

Example A:

```
ACCESS = DBAM
FORMAT = V
MODE = C
BLOCKSIZE = 512
BUFFERS = 2
VOLUMES = 1
FDMINOD = 502
FILACC = 0
FDMAXKY = 4
FDPRECLEN = 0
FDRTMERIT = 0
SUBINDEX = Y
BLDFDP VTP, DEFAULT, DEFAULT, DEFAULT,
DEFAULT, DEFAULT, DEFAULT, FDDBP
```

This will generate an Index FDP listing which looks like Figure II-7-1.

```

54 ;
55 ; FILE DEFINITION PACKET
56 ;
57
58 00455'000000 0
59 00456'000000 0
60 00457'000000 0

0006 ASDb
01 00460'044030 FIFL1 ; 3: FDFL1
02 00461'000000 FIFL2 ; 4: FDFL2
03 00462'000512 BLOCKSIZE ; 5: FDBLK
04 00463'000002 BUFFERS ; 6: FDBUF
05 00464'177777 RECORDSIZE ; 7: FDLLEN
06 00465'000000 0
07 00466'000000 0
08 00467'000001 VOLUMES ; 10: FDNVD
09 00470'000645 VIX ; 11: FDVTP
10 00471'000000 0
11 00472'177777 DEFAULT ; 13: FDUIT
12 00473'000000 0
13 00474'177777 DEFAULT ; 15: FDUOT
14 00475'000000 0
15 00476'000000 FITCF ; 17: FDTCF
16 00477'177777 DEFAULT ; 18: FDDSD
17 00500'000000 0
18 00501'177777 DEFAULT ; 20: FDFSI
19 00502'000000 0
20 00503'000000 0
21 00504'177777 EXPDAY ; 23: FDEXP
22 00505'177777 EXPMONTH
23 00506'177777 EXPYEAR
24 00507'177777 SEQUENCE ; 26: FDSEQ
25 00510'177777 FDGNRAT ; 27: FDGENER
26 00511'000000 FILACC*256 ; 28: FDACC
27 00512'177777 BUFFOFF ; 29: FDIDO
28 00513'177777 DEFAULT ; 30: FDSFT
29 00514'000000 0
30 00515'000000 0
31 00516'000000 0
32 00517'000000 0
33 00520'000000 0
34 00521'000000 0
35 00522'000000 0
36 00523'000000 0
37 00524'000000 0
38 00525'177777 DEFAULT ; 40: FDEFT
39 00526'000000 0
40 00527'000537' Fddb ; 42: FDDBP
41 00530'000000 0
42 00531'000502 FDMINOD ; 44: FDMNS
43 00532'000000 0
44 00533'002000 FDMAXKY*256!FDPRECLEN ; 46: FDMKL / FDPRL
45 00534'000000 0
46 00535'000000 FDRMERIT ; 48: FDRMF
47 00536'000000 FIIFL ; 49: FDIFL
48

```

Figure II-7-1. Index File Definition Packet

Licensed Material - Property of Data General Corporation

Each line corresponds to a word in the packet. The first column shows the relocatable location of that word; the second gives the contents of that location (i.e., the contents of each word); the third column describes the contents of each word; and the fourth column gives the decimal offset and the name of each field in the packet. Finally, notice how the macro routines automatically placed -1 (17777 octally) in the parameter words we didn't specify.

Example B:

This will generate an FDP for the Database associated with the Index FDP shown in Example A.

```
ACCESS = DBASE
FORMAT = V
MODE = C
BLOCKSIZE = 512
VOLUMES = 1
BUFFERS = 1
RECORDSIZE = 100
FILACC = 0
```

FDDB: BLDFDP VDB

Figure II-7-2 is what the packet looks like.

```

06          ;
07          ; FILE DEFINITION PACKET
08          ;
09
10 00537'000000      0
11 00540'000000      0
12 00541'000000      0
13 00542'004030      FIFL1          ; 3: FDFL1
14 00543'000000      FIFL2          ; 4: FDFL2
15 00544'000512      BLOCKSIZE       ; 5: FDBLK
16 00545'000001      BUFFERS         ; 6: FDBUF
17 00546'000100      RECORDSIZE      ; 7: FDLEN
18 00547'000000      0
19 00550'000000      0
20 00551'000001      VOLUMES         ; 10: FDNVD
21 00552'000665'     VDB             ; 11: FDVTP
22 00553'000000      0
23          ; 13: FDUIT
24 00554'000000      0
25          ; 15: FDUOT
26 00555'000000      0
27 00556'000000      FITCF          ; 17: FDTCF
28          ; 18: FDDSD
29 00557'000000      0
30          ; 20: FDFSI
31 00560'000000      0
32 00561'000000      0
33 00562'177777      EXPDAY         ; 23: FDEXP
34 00563'177777      EXPMONTH
35 00564'177777      EXPYEAR
36 00565'177777      SEQUENCE       ; 26: FDSEQ
37 00566'177777      FDGNRAT       ; 27: FDGENER
38 00567'000000      FILACC*256    ; 28: FDACC
39 00570'177777      BUFFOFF       ; 29: FDIID
40          ; 30: FDSFT
41 00571'000000      0
42 00572'000000      0
43 00573'000000      0
44 00574'000000      0
45 00575'000000      0
46 00576'000000      0
47 00577'000000      0
48 00600'000000      0
49 00601'000000      0
50

```

Figure II-7-2. Database File Definition Packet

BLDVDP
To construct a Volume Definition Packet

Access Method:

Required for all access methods.

Keyword Parameters:

Assignment Line		Actual Parameter
VOLSIZE	= (n)/DEFAULT	(VDVSZ)
NORESTART	= Y/N/DEFAULT	(ICDDR)
VARYBLOCKS	= Y/N/DEFAULT	(ICVLB)
NOLABELS	= Y/N/DEFAULT	(ICDSL)
PARITY	= Y/N/DEFAULT	(ICPAR)
NOCHECK	= Y/N/DEFAULT	(ICDCC)
CONTIGUOUS	= Y/N/DEFAULT	(ICCTG)
REINIT	= Y/N/DEFAULT	(ICERI)
RUNRELEASE	= Y/N/DEFAULT	(ICERR)
OPENREWIND	= Y/N/DEFAULT	(ICRWO)
TIMEOUT	= (n)	(VDDTO)
VOLMERIT	= (n)/DEFAULT	(VDVMF)
VACCESS	= (n)/DEFAULT	(VDACC)
PADCHAR	= (char)/DEFAULT	(VDPAD)
DUPVOL	= Y/N/DEFAULT	(ICDVC)
NOFILINIT	= Y/N/DEFAULT	(ICDFI)

Call:

BLDVDP vnp, vlt, hlt, tlt, oid

where the call parameters are:

Definition	Actual Parameter
vnp	Byte address of the name string (VDVNP)
vlt	Address of optional User Volume Label Table (labeled magnetic tape only) (VDVLT)
hlt	Address of optional User Header Label Table (labeled magnetic tape only) (VDHLT)
tlt	Address of optional User Trailer Label Table (labeled magnetic tape only) (VDTLT)
oid	Byte address of optional volume owner's ID (labeled magnetic tape only) (VDOID)

If you're not using labeled magnetic tape, write the call as follows:

BLDVDP vnp DEFAULT DEFAULT DEFAULT DEFAULT

Example A:

VDP for an Index File

```
VOLSIZE = 1000
VOLMERIT = 0
PADCHAR = 0
```

VIX: BLDVDP INAME*2,DEFAULT,DEFAULT,DEFAULT DEFAULT

This generates a VDP which looks like Figure II-7-3.

Example B:

VDP for the Database associated with the above Index:

```
VOLSIZE = 1000
VOLMERIT = 0
PADCHAR = 0
```

VDB: BLDVDP DNAME*2,DEFAULT,DEFAULT,DEFAULT DEFAULT

Figure II-7-4 is what the Database VDP looks like.

```

30 ;
31 ; VOLUME DEFINITION PACKET
32 ;
33 00645'000520" INAME*2 ; 0:VDVNP
34 00646'0000000 V V
35 00647'0010000 VGLSIZE ; 2:VDVSZ
36 00650'1777777 DEFAULT ; 3:VDVLT
37 00651'0000000 0
38 00652'1777777 DEFAULT ; 5:VDHLT
39 00653'0000000 0
40 00654'1777777 DEFAULT ; 7:VDTLT
41 00655'0000000 0
42 00656'0000000 0
43 00657'0000000 VPIVC ; 10:VDIVC
44 00660'1777777 TIMEOUT ; 11:VDOTO
45 00661'0000000 0
46 00662'0000000 VOLMERIT*256+PADCHAR ; 13:VDVMF / VDPAD
47 00663'1777777 DEFAULT ; 14:VDODID
48 00664'0000000 0
    
```

Figure II-7-3. Index File Volume Definition Packet

```

56 ;
57 ; VOLUME DEFINITION PACKET
58 ;
59 00665'000520" DNAME*2 ; 0:VDVNP
60 00666'0000000 0

0010 ASDB
01 00667'0010000 VGLSIZE ; 2:VDVSZ
02 00670'1777777 DEFAULT ; 3:VDVLT
03 00671'0000000 0
04 00672'1777777 DEFAULT ; 5:VDHLT
05 00673'0000000 0
06 00674'1777777 DEFAULT ; 7:VDTLT
07 00675'0000000 0
08 00676'0000000 0
09 00677'0000000 VPIVC ; 10:VDIVC
10 00700'1777777 TIMEOUT ; 11:VDOTO
11 00701'0000000 0
12 00702'0000000 VOLMERIT*256+PADCHAR ; 13:VDVMF / VDPAD
13 00703'1777777 DEFAULT ; 14:VDODID
14 00704'0000000 0
    
```

Figure II-7-4. Database Volume Definition Packet

BLDPP

To build a General or Extended Processing Packet

Access Method:

Used for all access methods.

Keyword Parameters:

Assignment Line		Actual Parameter
LOCK	= Y/N/DEFAULT	(PFLOC)
HOLD	= Y/N/DEFAULT	(PFHLD)
UNLOCK	= Y/N/DEFAULT	(PFUNL)
IMMEDIATE	= Y/N/DEFAULT	(PFWIF)
INHIBIT	= Y/N/DEFAULT	(PFRIN)
KEYED	= Y/N/DEFAULT	(CCKEY)
RELATIVE	= Y/N/DEFAULT	(CCREL)
UP	= Y/N/DEFAULT	(CCUP)
DOWN	= Y/N/DEFAULT	(CCDWN)
FORWARD	= Y/N/DEFAULT	(CCFWD)
BACKWARD	= Y/N/DEFAULT	(CCBAK)
STATIC	= Y/N/DEFAULT	(CCSTA)
SETCURR	= Y/N/DEFAULT	(CCSCP)
PRSUPRES	= Y/N/DEFAULT	(CCSPR)
DBSUPRES	= Y/N/DEFAULT	(CCSDB)
RECLN	= (n)	(PRLEN)
TAPDISP	= REWIND/ NOREWIND/ RELREWIND	(PRLEN)*
LIMRECLN	= Y/N/DEFAULT	(PRSPL)**

* This parameter allows you to specify tape disposition. If you do not specify one of these options, the system will automatically rewind the tape. The "RELREWIND" command enables run release as well as tape rewind on FEOV or close.

Call:

BLDPP dat, ktp, pra, sid

where the call parameters are:

Definition	Actual Parameter
dat	Byte address of data record (PRDAT)
ktp	Address of Key Table for ISAM or DBAM; otherwise, DEFAULT (PRKTP)
pra	Byte address of partial record area (DBAM only; otherwise DEFAULT) (PRPRA)
sid	Address of optional Subindex Definition Packet (DBAM only; otherwise DEFAULT) (PRSID)

NOTE: The generation of packet extensions is conditional. That is, the number of positional parameters in the call parameter list governs whether or not the system generates an Extended Processing Packet when you use BLDPP. A BLDPP call with a single parameter (Data Area Pointer) in the parameter list produces a General Processing Packet suitable for processing SAM or RAM files.

You should always code ISAM or DBAM BLDPP processing calls with all four parameters in the list. This will generate an Extended Processing packet. (Of course, you may use the word "DEFAULT" to indicate a parameter for which you do not wish to specify a value.)

Example:

```
KEYED = Y
RELATIVE = Y
DOWN = Y
SETCURR = Y
PP: BLDPP BUFX*2,DEFAULT,DEFAULT,DEFAULT
```

This sets up an Extended Processing Packet which looks like Figure II-7-5.

** In ISAM/DBAM processing this parameter sets bit 0 (PRSPL) of the record length field. This signifies that the length specified is the largest desired input and that the INFOS system should truncate any longer records it finds. If you do not supply this parameter, the system will reset the bit.

```

55          ;
56          ; PROCESSING PACKET
57          ;
58 00620'000000      PPSTA          ; 0: PRSTA
59 00621'000534"    BUFX*2        ; 1: PRDAT
60 00622'000000      0

0009 ASDR
01 00623'177777      DEFAULT        ; 3: PRKTP
02 00624'000000      0
03 00625'000001      PPLEN          ; 5: PRSPL+PRLEN
04 00626'000000      0
05 00627'000000      0
06 00630'000000      0
07 00631'000000      0
08 00632'000000      0
09 00633'000000      0
10 00634'152000      PPCCW          ; 12: PRCCW
11 00635'177777      DEFAULT        ; 13: PRPRA
12 00636'000000      0
13 00637'000000      0
14 00640'000000      0
15 00641'000000      0
16 00642'000000      0
17 00643'177777      DEFAULT        ; 19: PRSID
18 00644'000000      0
19

```

Figure II-7-5. Extended Processing Packet

BLDPNT

To construct a Point Processing Packet

Access Method:

SAM only.

Keyword Parameters:

Assignment Line		Actual Parameter
BYTEOFF	= (n)/DEFAULT	(PRBOF)
EOFMODE	= Y/N/DEFAULT	(PMEOF)
HILOGBLK	= (n)/DEFAULT	(PRHLB)
LOGBLKMODE	= Y/N/DEFAULT	(PMLBN)
LOLOGBLK	= (n)	(PRLLB)

Call:

BLDPNT

You need no calling parameters for this macro call.

Example:

```

BYTEOFF = 25
HILOGBLK = 165704
LOGBLKMODE = Y
LOLOGBLK = 034267
BLDPNT
    
```

This generates a Point Processing Packet which looks like Figure II-7-6.

BLDMTC

To construct a Magnetic Tape Control Processing Packet

Access Method:

SAM magnetic tape files only

Keyword Parameters:

Assignment Line		Actual Parameter
BWDFIL	= Y/N/DEFAULT	(MCSBF)
BWDREC	= Y/N/DEFAULT	(MCSBR)
FWDFIL	= Y/N/DEFAULT	(MCSFF)
FWDREC	= Y/N/DEFAULT	(MCSFR)
MTEOF	= Y/N/DEFAULT	(MCWEF)
MTERASE	= Y/N/DEFAULT	(MCERS)
MTREAD	= Y/N/DEFAULT	(MCRD)
MTREWIND	= Y/N/DEFAULT	(MCREW)
MTWRITE	= Y/N/DEFAULT	(MCWRT)
WORDSIZE	= (n)	(PRNWD)

Call:

BLDMTC dat

where the call parameter is:

Definition	Actual Parameter
dat	Byte address of a data record for either the MTREAD or MTWRITE calls. (PRDAT)

```

00711'000000 0 ; 0: PRSTA
00712'177777 DEFAULT ; 1: FRMOL
00713'165704 HILOGBLK ; 2: PRHLB
00714'034267 LLOGBLK ; 3: PRLLB
00715'000025 BYTEOFF ; 4: PRBOF
    
```

Figure II-7-6. Point Processing Packet

Licensed Material - Property of Data General Corporation

BLDKDP

To construct an INFOS Key Definition Packet.

Call:

BLDKDP key

where the call parameter is:

Access Method:

Used for indexed sequential (ISAM) and database (DBAM) access

Definition

Actual Parameter

key Byte address of the key (KDKYP)

Keyword Parameters:

Assignment Line		Actual Parameter
DUPLICATE	= Y/N/DEFAULT	(KTDUP)
GENERIC	= Y/N/DEFAULT	(KTGEN)
APPROXIMATE	= Y/N/DEFAULT	(KTAPX)
KEYLEN	= (byte)	(KDKYL)
HIOCCUR	= (n)/DEFAULT	(KDDKO)
LOOCCUR	= (n)	

Example:

KEYLEN = 2
BLDKDP

This produces the packet shown in Figure II-7-7.

```

0008 ASDE
01          ;
02          ; KEY DEFINITION PACKET
03          ;
04 00602'000002          KYTYP!KEYLEN.          ; 0: KDTYP / KDKYL
05          ; 1: KDKYP
06 00603'000000          0
07 00604'000000          HIOCCUR          ; 3: KDDKO
08 00605'000000          LOOCCUR

```

Figure II-7-7. Key Definition Packet

BLDSDP

Used to build a Subindex Definition Packet

Access Method:

DBAM only

Keyword Parameters:

Assignment Line		Actual Parameter
MINNODE	= (n)/DEFAULT	(SDINS)
MAXKEY	= (byte)/DEFAULT	(SDMKL)
PARTRECLN	= (byte)/DEFAULT	(SDPRL)
ROOTMERIT	= (n)/DEFAULT	(SDRMF)
COMPRESS	= Y/N/DEFAULT	(IXPKC)
SUBINDEX	= Y/N/DEFAULT	(IXNSI)
TEMPINDEX	= Y/N/DEFAULT	(IXTSI)
HIPRINOD	= Y/N/DEFAULT	(IXHPN)
PERMREC	= Y/N/DEFAULT	(IXPRM)

Call:

BLDSDP

You need no calling parameters for this call.

Example:

```
MINNODE = 506
MAXKEY = 4
PARTRECLN = 0
ROOTMERIT = 0
BLDSDP
```

This will generate the Subindex Definition Packet in Figure II-7-8.

```

35          ;
36          ; SUB-INDEX DEFINITION PACKET
37          ;
38 00612'000506      MINNODE          ; 0: SDMNS
39 00613'000000      0
40 00614'002000      MAXKEY*256+PARTRECLN ; 2: SDMKL / SDPRL
41 00615'000000      0
42 00616'000000      ROUTMERIT         ; 4: SDRMF
43 00617'000000      SIIFL             ; 5: SDIFL

```

Figure II-7-8. Subindex Definition Packet

Licensed Material - Property of Data General Corporation

BLDLSP

To build a Link Subindex Processing Packet

Access Method:

DBAM only

Keyword Parameters:

Assignment Line		Actual Parameter
DSTBACK	= Y/N/DEFAULT	(CCBAK)
DSTDOWN	= Y/N/DEFAULT	(CCDWN)
DSTFWD	= Y/N/DEFAULT	(CCFWD)
DSTKEYED	= Y/N/DEFAULT	(CCKEY)
DSTREL	= Y/N/DEFAULT	(CCREL)
DSTSETC	= Y/N/DEFAULT	(CCSCP)
DSTSTAT	= Y/N/DEFAULT	(CCSTA)
DSTUP	= Y/N/DEFAULT	(CCUP)
SRCBACK	= Y/N/DEFAULT	(CCBAK)
SRCDOWN	= Y/N/DEFAULT	(CCDWN)
SRCFWD	= Y/N/DEFAULT	(CCFWD)

Assignment Line

SRCKEYED	= Y/N/DEFAULT	(CCKEY)
SRCREL	= Y/N/DEFAULT	(CCREL)
SRCSETC	= Y/N/DEFAULT	(CCSCP)
SRCSTAT	= Y/N/DEFAULT	(CCSTA)
SRCUP	= Y/N/DEFAULT	(CCUP)

Actual Parameter

Call:

BLDLSP dkt, skt

where the call parameters are:

Definition	Actual Parameter
dkt	Byte address of Destination Key Table (PRDKT)
skt	Address of Source Key Table for ISAM and DBAM; otherwise DEFAULT (PRSKT)

Example:

DSTFWD = Y
DSTREL = Y
SRCKEYED = Y
SRCSETC = Y
BLDLSP DKT, SKT

This will generate the Link Subindex Processing Packet in Figure II-7-9.

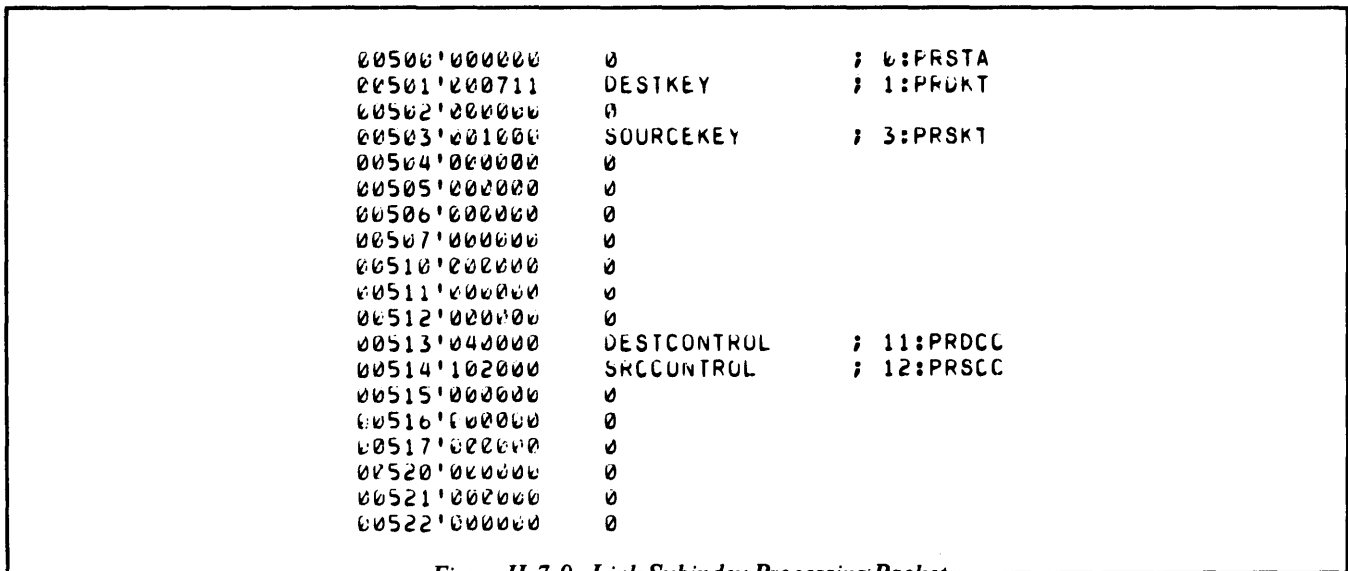


Figure II-7-9. Link Subindex Processing Packet

BLDVIP

To construct a Volume Initialization Packet.

Access Method:

SAM magnetic tape files only

Keyword Parameters:

Assignment Line		Actual Parameter
LABTYP	= ANSI/IBM/DEFAULT	(VFLT1-3)
LABLEV	= (n)/DEFAULT	(VFLL1-3)
FULLINIT	= Y/N/DEFAULT	(VFFUL)
IGNOREID	= Y/N/DEFAULT	(VFIVI)
FACCESS	= (n)/DEFAULT	(VIACC)

Call:

BLDVIP dvs, vid, oid, uvt

where the call parameters are:

Definition	Actual Parameter
dvs	Byte pointer to the device specifier (VIDVS)
vid	Byte address of the volume ID (VIVID)
oid	Byte address of the volume owner's ID (VIOID)
uvt	Address of the User Volume Label Table (VIUVT)

Example:

LABTYP = ANSI
LABLEV = 3
FULLINIT = Y
FACCESS = 0
BLDVIP DVS, VID, OID, UVT

This generates the packet in Figure II-7-10.

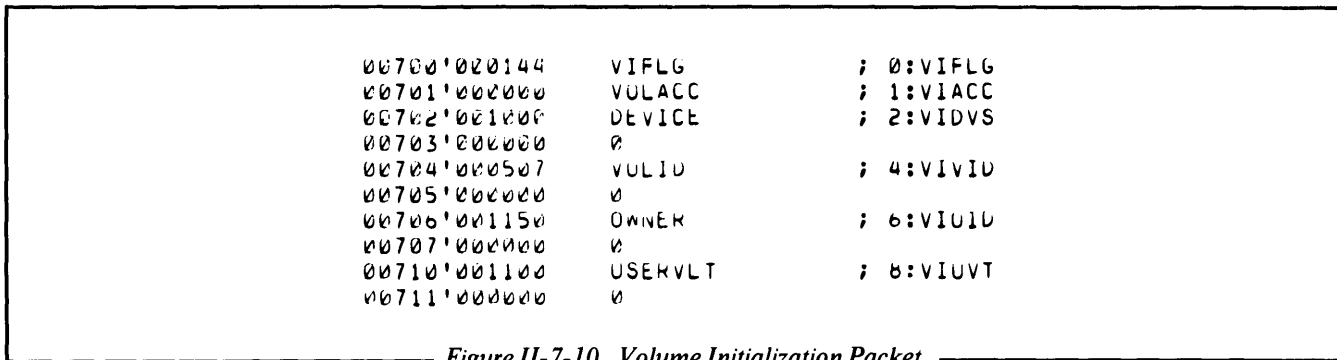


Figure II-7-10. Volume Initialization Packet

Licensed Material - Property of Data General Corporation

BLDULT

To construct a User Label Table

Access Method:

SAM magnetic tape files only

Keyword Parameters:

None

Call:

BLDULT addr1, addr2, ... addrn

where the call parameter is:

addr Byte address for each of the n label areas you want built into the table.

The Assembly Language Interface

In order to use the INFOS macro calls, you have to invoke the Assembly Language interface by including the call **BLDIPKT** on your assembly line before the name of your program:

```
MAC BLDIPKT PROGRAMNAME PROGRAMNAME.RB/B $LPT/L
```

This, however, assumes that you have included **PARIU.SR** in **MAC.PS** (the assembler's symbol table) and that **PROGRAMNAME.RB** is the name you're giving to the assembly output. You may also use the **.RB** pseudo-op within your source code to specify the name of the binary file.

If you did not include **PARIU.SR** in **MAC.PS**, you must invoke the interface by specifying:

```
MAC PARIU/S BLDIPKT PROGRAMNAME  
PROGRAMNAME.RB/B $LPT/L
```

There is one caution here: You should check your listing for diagnostic messages from **BLDIPKT** which appear as "WARNING" or "ERROR" comment lines just prior to a generated packet. Unlike error messages from the assembler itself, these will not appear on the system console. Furthermore, those labeled "ERROR" (rather than "WARNING") may indicate that the system has generated an unsatisfactory packet because of omitted, faulty, or conflicting keywords.

The final part of this chapter contains a listing of the **INFOS** User Parameters for assembly language.

INFOS User Parameters for Assembly Language

```

01
02      ;=====;
03      ;  INFOS USER PARAMETERS  ;
04      ;=====;
05
06      .TITL   PARIU
07
08      ;      FILE DEFINITION PACKET
09
10      000003 .DUSR  FDFL1  =   3          ;INFOS -FILE DEF FLAGS, I
11      000004 .DUSR  FDFL2  =  FDFL1+1    ;INFOS -FILE DEF FLAGS, II
12      000005 .DUSR  FDBLK  =  FDFL2+1    ;INFOS -BLOCK SIZE
13      000006 .DUSR  FDBUF  =  FDBLK+1    ;INFOS -NUMBER OF BUFFERS
14      000007 .DUSR  FDLEN  =  FDBUF+1    ;INFOS -RECORD LENGTH
15      000007 .DUSR  FDNIL  =  FDLEN      ;INFOS -NUMBER OF INDEX LEVELS
16      000012 .DUSR  FDNVD  =  FDLEN+3    ;INFOS -NUM OF VOL TABLE ENTRIES
17      000013 .DUSR  FDVTP  =  FDNVD+1    ;INFOS -VOLUME TABLE POINTER
18      000015 .DUSR  FDUIT  =  FDVTP+2    ;INFOS -USER INPUT TRANS TBL PTR
19      000017 .DUSR  FDUOT  =  FDUIT+2    ;INFOS -USER OUTPUT TRANS TEL PTR
20      000021 .DUSR  FDTCF  =  FDUOT+2    ;INFOS -TRANS & LABEL FLAGS
21      000022 .DUSR  FDCSD  =  FDTCF+1    ;INFOS -DATA SENS DELIM TABLE PTR
22      000024 .DUSR  FDFSI  =  FDCSD+2    ;INFOS -FILE SET ID PTR
23      000027 .DUSR  FDEXP  =  FDFSI+3    ;INFOS -EXPIRATION DATE
24      000032 .DUSR  FDSEQ  =  FDEXP+3    ;INFOS -SEQUENCE NUMBER
25      000033 .DUSR  FDGEN  =  FDSEQ+1    ;INFOS -GENERATION NUMBER
26      000034 .DUSR  FDACC  =  FDGEN+1    ;INFOS -FILE ACCESSABILITY
27      000035 .DUSR  FDIDO  =  FDACC+1    ;INFOS -INITIAL DATA OFFSET
28      000036 .DUSR  FDSFT  =  FDIDO+1    ;INFOS -SEL FIELD TRANS TABLE PTR
29      000050 .DUSR  FDLN1  =  FDSFT+12   ;INFOS -SAM & RAM FDP LENGTH
30
31      000050 .DUSR  FDEFT  =  FDSFT+12   ;INFOS -EXCLUDED FILE TABLE PTR
32      000052 .DUSR  FDDBP  =  FDEFT+2    ;INFOS -DATA BASE FILE DEF PACKET
33      ;INFOS -OR NAME POINTER
34      000054 .DUSR  FDMNS  =  FDDBP+2    ;INFOS -MINIMUM NODE SIZE
35      000056 .DUSR  FDMKL  =  FDMNS+2    ;INFOS -MAX KEY LENGTH (LH BYTE)
36      000056 .DUSR  FDPRL  =  FDMKL     ;INFOS -PART REC LEN (RH BYTE)
37      000060 .DUSR  FDRMF  =  FDPRL+2    ;INFOS -RT ND MERIT FACTOR
38      000061 .DUSR  FDIFL  =  FDRMF+1    ;INFOS -INDEX FLAGS
39      000062 .DUSR  FDLN2  =  FDIFL+1    ;INFOS -ISAM & DBAM FDP LENGTH
40
41      ;      SUBINDEX DEFINITION PACKET
42
43      000000 .DUSR  SDMNS  =   0          ;INFOS -MINIMUM NODE SIZE
44      000002 .DUSR  SDMKL  =  SDMNS+2    ;INFOS -MAX KEY LEN (LH BYTE)
45      000002 .DUSR  SDPRL  =  SDMKL     ;INFOS -PART REC LEN (RH BYTE)
46      000004 .DUSR  SDRMF  =  SDPRL+2    ;INFOS -RT ND MRT FACT (RH BYTE)
47      000005 .DUSR  SDIFL  =  SDRMF+1    ;INFOS -INDEX FLAGS
48      000006 .DUSR  SDLEN  =  SDIFL+1    ;INFOS -SUBINDEX DEF PACKET LENGTH

```

INFOS User Parameters for Assembly Language (continued)

```

01
02
03           ;           FILE DEFINITION FLAGS, I   (FDFL1)
04
05     100000 .DUSR   F10BK   =   180           ;INFOS -UNBLOCKED RECORDS
06     040000 .DUSR   F1AM1   =   181           ;INFOS -ACCESS METHOD FIELD
07     020000 .DUSR   F1AM2   =   182           ;INFOS -
08     010000 .DUSR   F1FT1   =   183           ;INFOS -RECORD FORMAT FIELD
09     004000 .DUSR   F1FT2   =   184           ;INFOS -
10     002000 .DUSR   F1RAW   =   185           ;INFOS -READ AFTER WRITE OVER
11     000200 .DUSR   F1OVR   =   188           ;INFOS -OVERWRITE (APPEND IF 0)
12     000100 .DUSR   F1EXF   =   189           ;INFOS -EXCLUSIVE FILE
13     000020 .DUSR   F1PM1   =   1811          ;INFOS -PROCESSING MODE FIELD
14     000010 .DUSR   F1PM2   =   1812          ;INFOS -
15     000004 .DUSR   F1RER   =   1813          ;INFOS -REWRITE (NORMAL IF 0)
16     000001 .DUSR   F1INV   =   1815          ;INFOS -INVERTING (DBAM)
17
18           ;           ACCESS METHOD SPECIFIERS   (F1AM1,F1AM2)
19
20     060000 .DUSR   F1AMM   =   F1AM1+F1AM2 ;INFOS -FIELD MASK
21     020000 .DUSR   F1SAM   =   F1AM2         ;INFOS -SAM
22     000000 .DUSR   F1RAM   =   0             ;INFOS -RAM
23     040000 .DUSR   F1ISM   =   F1AM1         ;INFOS -ISAM
24     040000 .DUSR   F1DBM   =   F1AM1         ;INFOS -DBAM
25
26           ;           RECORD FORMAT SPECIFIERS   (F1FT1,F1FT2)
27
28     014000 .DUSR   F1FTM   =   F1FT1+F1FT2 ;INFOS -FIELD MASK
29     010000 .DUSR   F1FIX   =   F1FT1         ;INFOS -FIXED LENGTH
30     004000 .DUSR   F1VAR   =   F1FT2         ;INFOS -VARIABLE LENGTH
31     000000 .DUSR   F1UND   =   0             ;INFOS -UNDEFINED LENGTH
32     014000 .DUSR   F1SEN   =   F1FT1+F1FT2 ;INFOS -DATA SENSITIVE
33
34           ;           PROCESSING MODE SPECIFIERS   (F1PM1,F1PM2)
35
36     000030 .DUSR   F1PMM   =   F1PM1+F1PM2 ;INFOS -FIELD MASK
37     000000 .DUSR   F1INP   =   0             ;INFOS -INPUT
38     000020 .DUSR   F1OUT   =   F1PM1         ;INFOS -OUTPUT
39     000010 .DUSR   F1UPD   =   F1PM2         ;INFOS -UPDATE
40     000030 .DUSR   F1CRU   =   F1PM1+F1PM2 ;INFOS -CREATE UPDATE
41
42           ;           FILE DEFINITION FLAGS, II   (FDFL2)
43
44     020000 .DUSR   F20TI   =   182           ;INFOS -OPEN ONLY THIS INDEX
45     002000 .DUSR   F2SPM   =   185           ;INFOS -SPACE MANAGEMENT
46     000100 .DUSR   F2ORD   =   189           ;INFOS -OPTIMIZE REC DISTRIBUTION
47     000040 .DUSR   F2DHR   =   1810          ;INFOS -DISABLE HIERARCHICAL REPLAC
48     000010 .DUSR   F2FDP   =   1812          ;INFOS -DATA BASE FDP PRESENT

```

INFOS User Parameters for Assembly Language (continued)

```

01
02      ;      TRANSLATION AND LABEL CONTROL FLAGS      (FDTCF)
03
04      100000 .DUSR  TCLT1  =  1B0      ;INFOS -LABEL TYPE FIELD
05      040000 .DUSR  TCLT2  =  1B1      ;INFOS -
06      020000 .DUSR  TCLT3  =  1B2      ;INFOS -
07      010000 .DUSR  TCSFT  =  1B3      ;INFOS -SELECTIVE FIELD TRANS
08      004000 .DUSR  TCOT1  =  1B4      ;INFOS -OUTPUT TRANS FIELC
09      002000 .DUSR  TCOT2  =  1B5      ;INFOS -
10      001000 .DUSR  TCOT3  =  1B6      ;INFOS -
11      000400 .DUSR  TCOT4  =  1B7      ;INFOS -
12      000200 .DUSR  TCLL1  =  1B8      ;INFOS -LABEL LEVEL
13      000100 .DUSR  TCLL2  =  1B9      ;INFOS -
14      000040 .DUSR  TCLL3  =  1B10     ;INFOS -
15      000010 .DUSR  TCIT1  =  1B12     ;INFOS -INPUT TRANS FIELD
16      000004 .DUSR  TCIT2  =  1B13     ;INFOS -
17      000002 .DUSR  TCIT3  =  1B14     ;INFOS -
18      000001 .DUSR  TCIT4  =  1B15     ;INFOS -
19
20      ;      LABEL TYPE SPECIFIERS      (TCLT1-TCLT3)
21
22      160000 .DUSR  TCLTM   =  TCLT1+TCLT2+TCLT3
23                                     ;INFOS -FIELD MASK
24      000000 .DUSR  TCANS   =  0        ;INFOS -ANSI STANDARD
25      020000 .DUSR  TCI6M   =  TCLT3     ;INFOS -IBM STANDARD
26
27      ;      LABEL LEVEL SPECIFIERS      (TCLL1-TCLL3)
28
29      000340 .DUSR  TCLLM   =  TCLL1+TCLL2+TCLL3
30                                     ;INFOS -FIELD MASK
31      000040 .DUSR  TCLV1   =  TCLL3     ;INFOS -LEVEL 1
32      000100 .DUSR  TCLV2   =  TCLL2     ;INFOS -LEVEL 2
33      000140 .DUSR  TCLV3   =  TCLL2+TCLL3 ;INFOS -LEVEL 3
34
35      ;      OUTPUT TRANSLATION SPECIFIERS      (TCOT1-TCOT4)
36
37      007400 .DUSR  TCOTM   =  TCOT1+TCOT2+TCOT3+TCOT4
38                                     ;INFOS -FIELD MASK
39      000000 .DUSR  TCNTO   =  0        ;INFOS -NO TRANS ON OUTPUT
40      000400 .DUSR  TCEAO   =  TCOT4     ;INFOS -EBCDIC TO ASCII
41      001000 .DUSR  TCAEO   =  TCOT3     ;INFOS -ASCII TO EBCDIC
42      007400 .DUSR  TCUTO   =  TCOTM     ;INFOS -USER TABLE
43
44      ;      INPUT TRANSLATION SPECIFIERS      (TCIT1-TCIT4)
45
46      000017 .DUSR  TCITM   =  TCIT1+TCIT2+TCIT3+TCIT4
47                                     ;INFOS -FIELD MASK
48      000000 .DUSR  TCNTI   =  0        ;INFOS -NO TRANS ON INPUT
49      000001 .DUSR  TCEAI   =  TCIT4     ;INFOS -EBCDIC TO ASCII
50      000002 .DUSR  TCAEI   =  TCIT3     ;INFOS -ASCII TO EBCDIC
51      000017 .DUSR  TCUTI   =  TCITM     ;INFOS -USER TABLE
52
53      ;      INDEX FLAGS      (FDIFL)
54
55      100000 .DUSR  IXPKC   =  1B0      ;INFOS -PERFORM KEY COMPRESSION
56      040000 .DUSR  IXNSI   =  1B1      ;INFOS -NO SUBINDICES
57      020000 .DUSR  IXHPN   =  1B2      ;INFOS -HIGH PRIORITY NODE
58      010000 .DUSR  IXTSI   =  1B3      ;INFOS -TEMP INDEX (PRIM ICR SUB)
59      004000 .DUSR  IXPRM   =  1B4      ;INFOS -MAKE DATA RECORDS PERMANENT

```

INFOS User Parameters for Assembly Language (continued)

```

01
02      ;      VOLUME INITIALIZATION PACKET
03
04      000000 .DUSR  VIFLG  =  0          ;INFOS -VOL INIT FLAGS
05      000001 .DUSR  VIACC  =  VIFLG+1    ;INFOS -VOL ACCESSABILITY (LH BYTE)
06      000002 .DUSR  VIDVS  =  VIACC+1    ;INFOS -DEV SPECIFIER PTR
07      000004 .DUSR  VIVID  =  VIDVS+2    ;INFOS -VOL ID PTR
08      000006 .DUSR  VIOID  =  VIVID+2    ;INFOS -OWNER ID PTR
09      000010 .DUSR  VIUVT  =  VICID+2    ;INFOS -USER VOL LABEL TAB PTR
10      000012 .DUSR  VILEN  =  VIUVT+2    ;INFOS -VOL INIT PACKET LENGTH
11
12      ;      VOLUME INITIALIZATION FLAGS  (VIFLG)
13
14      100000 .DUSR  VFLT1  =  TCLT1      ;INFOS -LABEL TYPE FLAGS
15      040000 .DUSR  VFLT2  =  TCLT2      ;INFOS -
16      020000 .DUSR  VFLT3  =  TCLT3      ;INFOS -
17      000200 .DUSR  VFLL1  =  TCLL1      ;INFOS -LABEL LEVEL FIELD
18      000100 .DUSR  VFLL2  =  TCLL2      ;INFOS -
19      000040 .DUSR  VFLL3  =  TCLL3      ;INFOS -
20      000004 .DUSR  VFFUL  =  1B13      ;INFOS -FULL INIT
21      000002 .DUSR  VFIVI  =  1B14      ;INFOS -IGNORE VOL ID
22
23      ;
24      ;      NOTE THAT THE LABEL TYPE & LABEL LEVEL SPECIFIERS
25      ;      GIVEN ON THE PREVIOUS PAGE MAY ALSO BE USED FOR
26      ;      THIS FLAG WORD

01
02      ;      VOLUME DEFINITION PACKET
03
04      000000 .DUSR  VDVNP  =  0          ;INFOS -VOLUME NAME POINTER
05      000002 .DUSR  VDVSZ  =  VDVNP+2    ;INFOS -VOLUME SIZE
06      000003 .DUSR  VDVLT  =  VDVSZ+1    ;INFOS -VOLUME LABEL TABLE PTR
07      000005 .DUSR  VDHLT  =  VDVLT+2    ;INFOS -HEADER LABEL TABLE PTR
08      000007 .DUSR  VDTLT  =  VDHLT+2    ;INFOS -TRAILER LABEL TABLE PTR
09      000011 .DUSR  VDPDC  =  VDTLT+2    ;INFOS -PHYS DEV CHARACTERISTICS
10      000012 .DUSR  VDIVC  =  VDPDC+1    ;INFOS -INFOS VOL CHARACTERISTICS
11      000013 .DUSR  VDDTO  =  VDIVC+1    ;INFOS -DEVICE TIME OUT CONSTANT
12      000015 .DUSR  VDVMF  =  VDDTO+2    ;INFOS -VOL MERIT FACTOR(LH BYTE)
13      000015 .DUSR  VDACC  =  VDVMF      ;INFOS -VOL ACCESSABILITY(LH BYTE)
14      000015 .DUSR  VDPAD  =  VDACC      ;INFOS -PAD CHARACTER (RH BYTE)
15      000016 .DUSR  VDGID  =  VDPAD+1    ;INFOS -VOLUME OWNER ID POINTER
16      000020 .DUSR  VDLEN  =  VDOID+2    ;INFOS -VOL DEF PACKET LENGTH
17
18      ;      INFOS VOLUME CHARACTERISTICS  (VDIVC)
19
20      100000 .DUSR  ICDDR  =  1B0         ;INFOS -DISABLE DEVICE RESTART
21      040000 .DUSR  ICVLB  =  1B1         ;INFOS -VARIABLE LENGTH BLOCKS
22      020000 .DUSR  ICDVC  =  1B2         ;INFOS -DUPLICATE VOLUME CONTROL
23      010000 .DUSR  ICDSL  =  1B3         ;INFOS -DISABLE SYSTEM LABELING
24      002000 .DUSR  ICPAR  =  1B5         ;INFOS -GENERATE PARITY
25      001000 .DUSR  ICDC  =  1B6         ;INFOS -DISABLE CONFLICT CHECKING
26      000200 .DUSR  ICCTG  =  1B8         ;INFOS -CONTIGUOUS ALLOCATION
27      000400 .DUSR  ICDFI  =  1B7         ;INFOS -DISABLE FILE INITIALIZATION
28      000040 .DUSR  ICERI  =  1B10        ;INFOS -ENABLE RUN TIME INIT
29      000020 .DUSR  ICERR  =  1B11        ;INFOS -ENABLE RUN TIME RELEASE
30      000010 .DUSR  ICRWO  =  1B12        ;INFOS -REWIND ON VOL OPEN

```

INFOS User Parameters for Assembly Language (continued)

```

01
02          ;      PROCESSING PACKET
03
04      000000 .DUSR  PRSTA  =  0          ;INFOS -STATUS FLAGS
05      000001 .DUSR  PRDAT  =  PRSTA+1    ;INFOS -DATA AREA PCINTER
06      000003 .DUSR  PRREC  =  PRDAT+2    ;INFOS -RECORD NUMBER (RAM)
07      000003 .DUSR  PRKTP  =  PRREC      ;INFOS -KEY TABLE POINTER (ISAM)
08      000005 .DUSR  PRLN   =  PRKTP+2    ;INFOS -RECORD LENGTH
09
10      100000 .DUSR  PRSPL  =  100        ;INFOS -WHEN USED GOES IN PRLN
11                                          ;INFOS -SPECIFIED RECORD LENGTH REQUEST
12                                          ;INFOS -(INPUT TO ISAM OR DBAM READ)
13                                          ;INFOS -RECORD EXCEEDS LENGTH REQUEST EC
14                                          ;INFOS -(RETURNED FROM ISAM OR DBAM REA
15
16      000005 .DUSR  PRDSP  =  PRLN        ;INFOS -MAG TAPE DISPOSITION
17      000010 .DUSR  PRDFB  =  PRLN+3     ;INFOS -DATA RECORD FEEDBACK
18      000012 .DUSR  PRSIL  =  PRDFB+2    ;INFOS -SUB-INDEX LEVEL (RH BYTE)
19      000013 .DUSR  PRLN1  =  PRSIL+1    ;INFOS -SAM & RAM PROC PACKET LEN
20
21      000013 .DUSR  PRRMF  =  PRDFB+3    ;INFOS -RECORD MERIT FACTOR
22      000014 .DUSR  PRCCW  =  PRRMF+1    ;INFOS -COMMAND CONTROL WORD
23      000015 .DUSR  PRPRA  =  PRCCW+1    ;INFOS -PARTIAL RECORD AREA PTR
24      000022 .DUSR  PRSRL  =  PRPRA+5    ;INFOS -RETURNED LEN (LH BYTE)
25      000022 .DUSR  PRSR5  =  PRSRL      ;INFOS -RETURNED STATUS (RH BYTE)
26      000023 .DUSR  PRSID  =  PRSR5+1    ;INFOS -SUBINDEX DEF PACKET PTR
27      000025 .DUSR  PRLN2  =  PRSID+2    ;INFOS -ISAM & DBAM PACKET LEN
28
29
30          ;      PROCESSING PACKET STATUS FLAGS      (PRSTA)
31
32      100000 .DUSR  PFLOC  =  100        ;INFOS -LOCK RECORD
33      040000 .DUSR  PFHLD  =  101        ;INFOS -HOLD REQUEST
34      020000 .DUSR  PFUNL  =  102        ;INFOS -UNLOCK RECORD
35      010000 .DUSR  PFVCI  =  103        ;INFOS -VOLUME CHANGE INDICATOR
36      004000 .DUSR  PFPEV  =  104        ;INFOS -PHYSICAL END OF VOLUME
37      002000 .DUSR  PFWIF  =  105        ;INFOS -WRITE IMMED IF MODIFIED
38      001000 .DUSR  PFREB  =  106        ;INFOS -RECORD EXCEEDS BUF SIZE
39      000400 .DUSR  PFRIN  =  107        ;INFOS -READ INHIBIT ON (RAM)
40      000200 .DUSR  PFXLS  =  108        ;INFOS -XFER LENGTH SHORT
41      000100 .DUSR  PFEXL  =  109        ;INFOS -EXCESSIVE XFER LENGTH
42      000040 .DUSR  PFPEF  =  1010       ;INFOS -PHYSICAL END OF FILE
43      000020 .DUSR  PFVER  =  1011       ;INFOS -VERIFICATION FAILURE
44      000010 .DUSR  PFMTR  =  1012       ;INFOS -MAG TAPE CONTROL ERROR
45      000004 .DUSR  PFUVL  =  1013       ;INFOS -USER VOL LABEL PROCESSED
46      000002 .DUSR  PFUHL  =  1014       ;INFOS -USER HDR LABEL PROCESSED
47      000001 .DUSR  PFUTL  =  1015       ;INFOS -USER TRAILER LAB PROCESSED
48      015340 .DUSR  PFERM  =  PFVCI+PFPEV+PFREB+PFXLS+PFEXL+PFPEF
49      015377 .DUSR  PFERM  =  PFERM+PFVER+PFMTR+PFUVL+PFUHL+PFUTL
50                                          ;INFOS -EXCEPTIONAL RETURN MASK
51
52          ;      INDEX COMMAND CONTROL FLAGS      (PRCCW)
53
54      100000 .DUSR  CCKEY  =  100        ;INFOS -KEYED
55      040000 .DUSR  CCREL  =  101        ;INFOS -RELATIVE TO CUR PCS
56      020000 .DUSR  CCMC1  =  102        ;INFOS -MOTION CONTROL FIELD
57      010000 .DUSR  CCMC2  =  103        ;INFOS -
58      004000 .DUSR  CCMC3  =  104        ;INFOS -
59      002000 .DUSR  CCSCP  =  105        ;INFOS -SET CURRENT POSITION

```


INFOS User Parameters for Assembly Language (continued)

```

01      000100 .DUSR  CCSPR  =   1B9      ;INFOS -SLPPRESS PARTIAL RECORD
02      000020 .DUSR  CCSOB  =   1B11     ;INFOS -SLPPRESS DATA BASE
03      000010 .DUSR  CCLOG  =   1B12     ;INFOS -LOGICAL KEY DELETE
04      000004 .DUSR  CCLOC  =   1B13     ;INFOS -LOCAL LOG DELETE
05      000002 .DUSR  CCGLB  =   1B14     ;INFOS -GLOBAL LOG DELETE
    
```

```

01
02      ;          MOTION CONTROL SPECIFIERS (CCMC1-CCMC3)
03
    
```

```

04      034000 .DUSR  CCMCM  =   CCMC1+CCMC2+CCMC3
05                                     ;INFOS -FIELD MASK
06      000000 .DUSR  CCFWD  =   0        ;INFOS -FORWARD
07      004000 .DUSR  CCBK  =   CCMC3     ;INFOS -BACKWARD
08      010000 .DUSR  CCDWN  =   CCMC2     ;INFOS -DOWN
09      014000 .DUSR  CCDFW  =   CCMC2+CCMC3 ;INFOS -DOWN & FORWARD
10      020000 .DUSR  CCUFW  =   CCMC1     ;INFOS -UP & FORWARD
11      024000 .DUSR  CCUBK  =   CCMC1+CCMC3 ;INFOS -UP & BACKWARD
12      030000 .DUSR  CCUP  =   CCMC1+CCMC2 ;INFOS -UP
13      034000 .DUSR  CCSTA  =   CCMC1+CCMC2+CCMC3
14                                     ;INFOS -STATIC
15
    
```

```

16      ;          SYSTEM RETURNED STATUS FLAGS (PRSRs)
17
    
```

```

18      000200 .DUSR  SRLLD  =   1B8      ;INFOS -LOCAL LOGICAL DELETE
19      000100 .DUSR  SRDUP  =   1B9      ;INFOS -DUPLICATE KEY
20      000040 .DUSR  SRDRL  =   1B10     ;INFOS -DATA RECORD LOCKED
21      000020 .DUSR  SRGLD  =   1B11     ;INFOS -GLOBAL LOGICAL DELETE
22      000220 .DUSR  SRLDM  =   SRLLD+SRGLD ;INFOS -LOGICAL DELETE MASK
23      000360 .DUSR  SRSFM  =   SRLLD+SRDUP+SRDRL+SRGLD
24                                     ;INFOS -STATUS FIELD MASK
    
```

```

01
02      ;          MAG-TAPE CONTROL PROCESSING PACKET
03
    
```

```

04      000003 .DUSR  PRCFC  =   PRDAT+2   ;INFOS -CONTROL FUNCTION CODE
05      000004 .DUSR  PRNWD  =   PRCFC+1   ;INFOS -NUMBER OF WORDS
06
    
```

```

07      ;          MAG-TAPE CONTROL FUNCTION CODES (PRCFC)
08
    
```

```

09      000000 .DUSR  MCSFF  =   0        ;INFOS -SPACE FORWARD FILE
10      000001 .DUSR  MCSBF  =   MCSFF+1  ;INFOS -SPACE BACKWARD FILE
11      000002 .DUSR  MCRD  =   MCSBF+1  ;INFOS -READ
12      000003 .DUSR  MCWRT  =   MCRD+1  ;INFOS -WRITE
13      000004 .DUSR  MCWEF  =   MCWRT+1  ;INFOS -WRITE EOF
14      000005 .DUSR  MCREW  =   MCWEF+1  ;INFOS -REWIND
15      000006 .DUSR  MCSFR  =   MCREW+1  ;INFOS -SPACE FORWARD REC
16      000007 .DUSR  MCSBR  =   MCSFR+1  ;INFOS -SPACE BACKWARD REC
17      000010 .DUSR  MCERS  =   MCSBR+1  ;INFOS -ERASE
18
    
```

```

19      ;          POINT PROCESSING PACKET
20
    
```

```

21      000001 .DUSR  PRMOD  =   PKSTA+1   ;INFOS -INPUT MODE
22      000002 .DUSR  PRHLB  =   PRMOD+1  ;INFOS -HI LOGICAL BLOCK
23      000003 .DUSR  PRLLB  =   PRHLB+1  ;INFOS -LOW LOGICAL BLOCK
24      000004 .DUSR  PRBUF  =   PRLLB+1  ;INFOS -BYTE OFFSET
25
    
```

```

26      ;          POINT INPUT MODE (PRMOD)
27
    
```

```

28      000000 .DUSR  PMEOF  =   0        ;INFOS -POINT TO EOF
29      000001 .DUSR  PMLBN  =   1        ;INFOS -LOGICAL BLOCK NUM
30
    
```

INFOS User Parameters for Assembly Language (continued)

```

31          ;      LINK SUB-INDEX PROCESSING PACKET
32
33      000001 .DUSR  PRCKT  =  PRDAT      ;INFOS -DEST KEY TABLE PTR
34      000003 .DUSR  PRSKT  =  PRKTP      ;INFOS -SOURCE KEY TABLE PTR
35      000013 .DUSR  PRDCC  =  PRRMF      ;INFOS -DEST COMMAND CONTROL
36      000014 .DUSR  PRSCC  =  PRCCW      ;INFOS -SOURCE COMMAND CONTROL

01
02          ;      KEY DEFINITION PACKET
03
04      000000 .DUSR  KDTPY  =  0          ;INFOS -KEY TYPE FLAGS (LF BYTE)
05      000000 .DUSR  KDKYL  =  KDTPY      ;INFOS -KEY LEN (RH BYTE)
06      000001 .DUSR  KDKYP  =  KDKYL+1    ;INFOS -KEY POINTER
07      000003 .DUSR  KDDKO  =  KDKYP+2    ;INFOS -DUP KEY OCCURENCE
08      000005 .DUSR  KDLEN  =  KDDKO+2    ;INFOS -KEY DEF PACKET LENGTH
09
10          ;      KEY TYPE FLAGS (KDTPY)
11
12      100000 .DUSR  KTDUP  =  1B0        ;INFOS -DUPLICATE KEY
13      040000 .DUSR  KTGEN  =  1B1        ;INFOS -GENERIC KEY
14      020000 .DUSR  KTAPX  =  1B2        ;INFOS -APPROX KEY

01
02          ;      INFOS SYSTEM CALLS
03
04      027400 .DUSR  .PINFOS =  57B7      ;INFOS -PRE-OPEN
05      022400 .DUSR  .QINFOS =  45B7      ;INFOS -OPEN
06      022000 .DIO   .INFOS  =  44B7      ;INFOS -PROCESSING CALL
07      021073 .DUSR  .IINFOS =  .SCALL 73 ;INFOS -LABELED INIT FOR MAG TAPE

```

INFOS User Parameters for Assembly Language (continued)

```

08
09
10           ;           .INFOS ARGUMENTS
11
12     000000 .DUSR     .POINT = 0           ;INFOS -PCINT
13     000001 .DUSR     .CNTRL = .POINT+1    ;INFOS -CONTROL
14     000002 .DUSR     .FEUV = .CNTRL+1     ;INFOS -FORCE END OF VOL
15     000003 .DUSR     .ICLOSE = .FEOV+1    ;INFOS -INFOS CLOSE
16     000004 .DUSR     .SETX = .ICLOSE+1    ;INFOS -SET EXCLUSIVE USE
17     000005 .DUSR     .RELX = .SETX+1      ;INFOS -RELEASE EXCLUSIVE USE
18     000006 .DUSR     .TRNCT = .RELX+1     ;INFOS -TRUNCATE BLGCK
19     000007 .DUSR     .IREAD = .TRNCT+1    ;INFOS -INFOS READ
20     000010 .DUSR     .IWRITE = .IREAD+1   ;INFOS -INFOS WRITE
21     000011 .DUSR     .DEFSI = .IWRITE+1   ;INFOS -DEFINE SUB-INDEX
22     000012 .DUSR     .LNKSI = .DEFSI+1    ;INFOS -LINK SUB-INDEX
23     000013 .DUSR     .DELRC = .LNKSI+1    ;INFOS -DELETE RECORD
24     000014 .DUSR     .DELSI = .DELRC+1    ;INFOS -DELETE SUB-INDEX
25     000015 .DUSR     .RETST = .DELSI+1    ;INFOS -RETURN STATUS
26     000016 .DUSR     .RETHK = .RETST+1    ;INFOS -RETURN HIGH KEY
27     000017 .DUSR     .RETKY = .RETHK+1    ;INFOS -RETURN KEY
28     000020 .DUSR     .REINS = .RETKY+1    ;INFOS -REINSTATE REC
29     000021 .DUSR     .RDDIR = .REINS+1    ;INFOS -READ DIRECT
30     000022 .DUSR     .WRDIR = .RDDIR+1    ;INFOS -WRITE DIRECT
31     000023 .DUSR     .RELRC = .WRDIR+1    ;INFOS -RELEASE REC
32     000024 .DUSR     .RELSE = .RELRC+1    ;INFOS -RELEASE BUFFER
33     000025 .DUSR     .REWRT = .RELSE+1    ;INFOS -REWRITE
34     000026 .DUSR     .RETDF = .REWRT+1    ;INFOS -RETURN SUB-INDEX DEF
35     000027 .DUSR     .PRERD = .RETDF+1    ;INFOS -PREREAD

```

INFOS User Parameters for Assembly Language (continued)

01					
02	;		INFOS ERROR CODES		
03					
04	000200 .DUSR	IOILF	=	200	;INFOS -ILLEGAL FUNCTION
05	000201 .DUSR	IOVTI	=	ICILF+1	;INFOS -VARIABLE LENGTH TRANSFER
06					;INFOS -ILLEGAL ON THIS DEVICE
07	000202 .DUSR	IORDO	=	IOVTI+1	;INFOS -REWRITE ON DISK ONLY
08	000203 .DUSR	IGIFD	=	IGRDO+1	;INFOS -ILLEGAL FUNCTION FOR DEV
09	000204 .DUSR	IOOPE	=	IGIFD+1	;INFOS -OPEN PROCESSING ERROR
10	000205 .DUSR	IORFC	=	IOOPE+1	;INFOS -REC FMT & FUNC CONFLICT
11	000206 .DUSR	IOEXF	=	IORFC+1	;INFOS -FILE IN USE
12	000207 .DUSR	IOLOK	=	IOEXF+1	;INFOS -FILE LOCKED
13	000210 .DUSR	ICFNO	=	IOLOK+1	;INFOS -FILE NOT OPEN
14	000211 .DUSR	IOPCF	=	ICFNO+1	;INFOS -PERIPHERAL CONFLICT
15	000212 .DUSR	IOVFP	=	IOPCF+1	;INFOS -VL FILE PROCESSING ERROR
16	000213 .DUSR	IOURC	=	IOVFP+1	;INFOS -UNRESOLVED RESOURCE CONFLICT
17	000214 .DUSR	IORMP	=	IOURC+1	;INFOS -REWRITE MODE PROCESSING ERROR
18	000215 .DUSR	IODVF	=	IORMP+1	;INFOS -DUPLICATE VL FILE
19	000216 .DUSR	IOBEW	=	IODVF+1	;INFOS -BLOCK SIZE EXCEEDS WINDOW SI E
20	000217 .DUSR	IOVME	=	IOBEW+1	;INFOS -VIRTUAL MEMORY EXHAUSTED
21	000220 .DUSR	IOXTL	=	IOVME+1	;INFOS -TRANSLATE TABLE LOAD ERROR
22	000221 .DUSR	IOVFE	=	IOXTL+1	;INFOS -VL FILE OPEN ERR
23	000222 .DUSR	IOVCE	=	IOVFE+1	;INFOS -VL FILE CLOSE ERR
24	000223 .DUSR	IOIFO	=	IOVCE+1	;INFOS -INSUF FREESPACE FOR OPEN
25	000224 .DUSR	IOLEF	=	IOIFO+1	;INFOS -LOGICAL END OF FILE
26	000225 .DUSR	IOUXS	=	IOLEF+1	;INFOS -USER TRANSLATE SPECIFICATION ER
27	000226 .DUSR	IONSV	=	IOUXS+1	;INFOS -NO SUCH VOLUME
28	000227 .DUSR	IONOH	=	IONSV+1	;INFOS -NO HOLD ON LOCKED REQUEST
29	000230 .DUSR	IONMD	=	IONOH+1	;INFOS -NO MORE DISK SPACE
30	000231 .DUSR	IOROF	=	IONMD+1	;INFOS -RAM ACCESS OUTSIDE FILE
31	000232 .DUSR	IOICL	=	IOROF+1	;INFOS -ILLEGAL CLOSE
32	000233 .DUSR	IOPIO	=	IOICL+1	;INFOS -PHYSICAL I/O ERROR
33	000234 .DUSR	IORDE	=	IOPIO+1	;INFOS -RESIDUAL DISK ERRCR
34	000235 .DUSR	IOTMO	=	IORDE+1	;INFOS -DISK OR MAG-TAPE TIME-OUT
35	000236 .DUSR	IOIAM	=	IOTMO+1	;INFOS -ILLEGAL ACCESS METHOD
36	000237 .DUSR	IOXER	=	IOIAM+1	;INFOS -ILLEGAL TRANS REQEST
37	000240 .DUSR	IOPRO	=	IOXER+1	;INFOS -PREOPEN OPEN ERROR
38	000241 .DUSR	IOPRC	=	IOPRO+1	;INFOS -PREOPEN CLOSE ERRCR
39	000242 .DUSR	IOFCE	=	IOPRC+1	;INFOS -FILE CLOSE ERROR
40	000243 .DUSR	IOROE	=	IOFCE+1	;INFOS -RDOS OPEN ERROR
41	000244 .DUSR	IOVAX	=	IOROE+1	;INFOS -VOLUME ALREADY EXISTS
42	000245 .DUSR	IOZDX	=	IOVAX+1	;INFOS -ZERO LEN DISK XFER REQ
43	000246 .DUSR	IOIUC	=	IOZDX+1	;INFOS -ISAM UPDATE CONFLICT
44					;INFOS -TABLE OVERFLOW
45	000247 .DUSR	IONMR	=	IOIUC+1	;INFOS -INDEX NAME ERROR
46	000250 .DUSR	IONSI	=	IONMR+1	;INFOS -NO SUCH INDEX
47	000251 .DUSR	IONTL	=	IONSI+1	;INFOS -NAME TOO LONG
48	000252 .DUSR	IONNS	=	IONTL+1	;INFOS -NO NODE SPACE
49	000253 .DUSR	IOYTE	=	IONNS+1	;INFOS -MAG-TAPE I/O ERROR
50	000254 .DUSR	IODNS	=	IOYTE+1	;INFOS -DEVICE NOT SUPPORTED
51	000255 .DUSR	IOEVO	=	IODNS+1	;INFOS -OUTPUT END VOLUME ERROR
52	000256 .DUSR	IOEVI	=	IOEVO+1	;INFOS -INPUT END VOLUME ERROR
53	000257 .DUSR	IOCMP	=	IOEVI+1	;INFOS -COMPARE ERROR (ISAM)
54	000260 .DUSR	IOPEZ	=	IOCMP+1	;INFOS -RESOLUTION ERROR (ISAM)
55	000261 .DUSR	IOIRM	=	IOPEZ+1	;INFOS -ILLEGAL REL MOTION
56	000262 .DUSR	IOINA	=	IOIRM+1	;INFOS -INVALID NODE ADDRESS
57	000263 .DUSR	IOICE	=	IOINA+1	;INFOS -INVALID CURRENT ENTRY
58	000264 .DUSR	IOTLV	=	IOICE+1	;INFOS -TOP LEVEL ERROR
59	000265 .DUSR	IOSNA	=	IOTLV+1	;INFOS -SUB INDICES NOT ALLOWED

INFOS User Parameters for Assembly Language (concluded)

01									
02	000267	.DUSR	IOESI	=	IOSNP+1	;INFOS	-END OF SUB-INDEX		
03	000270	.DUSR	IODPE	=	ICESI+1	;INFOS	-DELETE POSITIONING ERROR		
04	000271	.DUSR	IOMKW	=	IODPE+1	;INFOS	-MULTI KEY WRITE ERROR		
05	000272	.DUSR	IOIKL	=	ICMKW+1	;INFOS	-ILLEGAL KEY LENGTH		
06	000273	.DUSR	IOIEN	=	IOIKL+1	;INFOS	-INVALID ENTRY NUMBER		
07	000274	.DUSR	IOIPS	=	IOIEN+1	;INFOS	-ILLEGAL COMMAND CONTROL		
08	000275	.DUSR	IOKAE	=	IOIPS+1	;INFOS	-KEY ALREADY EXISTS		
09	000276	.DUSR	IOKPE	=	IOKAE+1	;INFOS	-KEY POSITIONING ERROR		
10	000277	.DUSR	IOIRL	=	IOKPE+1	;INFOS	-INVALID RECORD LENGTH		
11	000400	.DUSR	IORNP	=	400	;INFOS	-DATA BASE REC NOT PRESENT		
12	000401	.DUSR	IONTB	=	IORNP+1	;INFOS	-MIN NODE SIZE TOO BIG		
13	000402	.DUSR	IONTS	=	IONTB+1	;INFOS	-MIN NODE SIZE TOO SMALL		
14	000403	.DUSR	IODRL	=	IONTS+1	;INFOS	-DATA RECORD LOCKED		
15	000404	.DUSR	IOSIA	=	IODRL+1	;INFOS	-SUB-INDEX IN USE		
16	000405	.DUSR	IOVER	=	IOSIA+1	;INFOS	-VERSION CONFLICT ERROR		
17	000406	.DUSR	IOSIL	=	IOVER+1	;INFOS	-SUB-INDEX LINK COUNT		
18						;INFOS	-OVERFLOW		
19	000407	.DUSR	ICALS	=	IOSIL+1	;INFOS	-ALREADY LINKED		
20						;INFOS	-TO SUB-INDEX		
21	000410	.DUSR	IOSLO	=	ICALS+1	;INFOS	-SUB-INDEX LEVEL OVERFLOW		
22	000411	.DUSR	IOSSI	=	IOSLO+1	;INFOS	-SUB-INDEX HAS SUB-INDEX		
23						;INFOS	-DELETE SUB-INDEX ERROR		
24	000412	.DUSR	IODWK	=	IOSSI+1	;INFOS	-ATTEMPT TO DELETE ENTRY		
25						;INFOS	-WITHOUT KEYED ACCESS		
26	000413	.DUSR	IOENL	=	IODWK+1	;INFOS	-INDEX ENTRY LOCKED		
27	000414	.DUSR	IOWWK	=	IOENL+1	;INFOS	-NO WRITE WITHOUT KEY		
28	000415	.DUSR	IOILL	=	IOWWK+1	;INFOS	-ILLEGAL LABEL		
29	000416	.DUSR	IOILS	=	IOILL+1	;INFOS	-ILLEGAL LABEL SPEC		
30	000417	.DUSR	IOVID	=	IOILS+1	;INFOS	-VOL ID DOESNT MATCH		
31	000420	.DUSR	IOFID	=	IOVID+1	;INFOS	-FILE ID DOESNT MATCH		
32	000421	.DUSR	IOFSQ	=	IOFID+1	;INFOS	-FILE SEQ NUM DOESNT MATCH		
33	000422	.DUSR	IOGEN	=	IOFSQ+1	;INFOS	-GEN NUM DOESNT MATCH		
34	000423	.DUSR	IOEXD	=	IOGEN+1	;INFOS	-EXP DATE NOT EXPIRED		
35	000424	.DUSR	IOBCT	=	IOEXD+1	;INFOS	-BLOCK COUNT INCORRECT		
36	000425	.DUSR	IORFT	=	IOBCT+1	;INFOS	-RECORD FORMAT CONFLICT		
37	000426	.DUSR	IOFSN	=	IORFT+1	;INFOS	-FILE SECTION NUMBER		
38	000427	.DUSR	IOEIL	=	IOFSN+1	;INFOS	-EXCESSIVE POSITION LEVELS		
39	000430	.DUSR	IOSLS	=	IOEIL+1	;INFOS	-SYSTEM LOAD SIZE ERROR		
40	000431	.DUSR	IOFNF	=	IOSLS+1	;INFOS	-TAPE FILE NOT FOUND		
41	000432	.DUSR	IOBTS	=	IOFNF+1	;INFOS	-BLOCKSIZE < 8 BYTES		
42	000433	.DUSR	IORTL	=	IOBTS+1	;INFOS	-RECORD+OVERHEAD > BLOCKSIZE		
43	000434	.DUSR	IOWNE	=	IORTL+1	;INFOS	-WRITE IS NOT AT END-OF-FILE		
44						;INFOS	-FOR SHARED SAM UPDATE FILE		
45	000435	.DUSR	IOUWO	=	IOWNE+1	;INFOS	-WRITE ALLOWED ONLY FOR ONE		
46						;INFOS	-USER OF SHARED SAM UPDATE FILE		
47	000436	.DUSR	IOSPL	=	IOUWO+1	;INFOS	-SPOOLING ENABLED ON ILLEGAL DE		
48	000437	.DUSR	IORKR	=	IOSPL+1	;INFOS	- INFOS RETRIEVE KEY ERROR		
49	000440	.DUSR	IODIP	=	IORKR+1	;INFOS	- DELETE INDEX POS ERROR		
50	000441	.DUSR	IOMPR	=	IODIP+1	;INFOS	- SPACE MANAGEMENT INCONSISTE..CY		
51	000442	.DUSR	IOSTR	=	IOMPR+1	;INFOS	- SEARCH CP TABLE ERROR		
52									
53									
54									
			.EOT						

NOTE: See Appendix D for the octal values of the above error codes and further explanation of what each one means.

End of Chapter

Part Three: Appendixes

Labeled Magnetic Tapes

Subindex and Database File Properties

The INFOS/FORTRAN Interface

INFOS System Error Messages

Device Characteristics

Appendix A

Labeled Magnetic Tapes

General Concepts

A labeled magnetic tape is one which contains your data plus information about your data. This information is contained in labels and consists of things like volume name, filename, file format, etc. You get two significant benefits from using labeled tapes:

- Your file information is retained in a consistent format;
- You can call your files by a logical name rather than by a device name.

Labels come in two types: system labels and user labels. The INFOS system will automatically generate system labels from the information you supply when you create your file. If you want to store additional file information, you may also specify the contents of your own user labels.

You can fully or partially initialize volumes (i.e., reels of tape) with the INFOS system, and you can initialize and release volumes through system calls or at runtime. Runtime initialization allows you to process a multivolume file, even though you may have fewer tape drives than the number of volumes in your file.

Label Types and Levels

You can use any of the following label types and levels for your INFOS files:

- ANSI levels 1, 2, or 3 (recorded in ASCII); or
- IBM levels 1 or 2 (recorded in EBCDIC).

The ANSI labels are defined by The American National Standard, X3.27 - 1969, as modified by X3L5/36ST, dated September 27, 1973. The IBM labels are defined in various IBM publications. If you are not sure which level to use for either label type, choose the highest - i.e., ANSI 3 or IBM 2. This will give you the most flexibility when you use your files, but won't take much more space on your tapes than the lower levels. In other words, if you use the highest level when you're writing to your file, it will give you maximum information about how your tape is organized when you go to use it again. Also, when you are reading such a file, the system will allow you to access any of the lower label levels in addition to the highest one. For example, if you specify ANSI 3 input, you can read ANSI levels 1, 2, or 3; if you specify IBM 2, you can read IBM levels 1 or 2.

The charts in Tables A-1 to A-5 summarize the required and allowable label identifiers for each label type and level. The terms *Required* and *Allowed* refer to the Input processing mode. "Required" means that the specified label must be on the tape if you want to process it. "Allowed" means that the presence of that label will not cause an error (i.e., the system will ignore it). The term *Processed* refers to both the Input and Output processing modes. On Input, the system will read the "Processed" labels; on Output, it will write the specified label on the tape. Finally, note that End-of-Volume (EOV) labels are required on *all* multivolume tape files.

Table A-1. Level 1 ANSI Labels

Label Group Name	Label Set Name	Label Identifier	Required	Allowed	Processed
Beginning of Volume	Volume Header	VOL	VOL1	VOL1	VOL1
Beginning of File Section	File Header	HDR	HDR1	HDR1-HDR9	HDR1
End of File Section	End of Volume	EOV	EOV1	EOV1-EOV9	EOV1
End of File	End of File	EOF	EOF1	EOF1-EOF9	EOF1

- NOTES: • You may only have Fixed Length data records at this level.
- You can record a single file on more than one volume.

Table A-2. Level 2 ANSI Labels

Label Group Name	Label Set Name	Label Identifier	Required	Allowed	Processed
Beginning of Volume	Volume Header	VOL	VOL1	VOL1	VOL1
Beginning of File Section	File Header	HDR	HDR1	HDR1-HDR9	HDR1
	User Header	UHL	None	UHL1-UHL9	UHL1-UHL9
End of File Section	End of Volume	EOV	EOV1	EOV1-EOV9	EOV1
	User Trailer	UTL	None	UTL1-UTL9	UTL1-UTL9
End of File	End of File	EOF	EOF1	EOF1-EOF9	EOF1
	User Trailer	UTL	None	UTL1-UTL9	UTL1-UTL9

- NOTES: • You may have Fixed or Variable length data records at this level.
- You can record more than one file on a single volume.

Table A-3. Level 3 ANSI Labels

Label Group Name	Label Set Name	Label Identifier	Required	Allowed	Processed
Beginning of Volume	Volume Header	VOL	VOL1	VOL1	VOL1
	User Volume	UVL	None	UVL1-UVL9	UFL1-UVL9
Beginning of File Section	File Header	HDR	HDR1	HDR1-HDR9	HDR1-HDR2
	User Header	UHL	None	UHL1-UHL9	UHL1-UHL9
End of File Section	End of Volume	EOV	EOV1	EOV1-EOV9	EOV1-EOV2
	User Trailer	UTL	None	UTL1-UTL9	UTL1-UTL9
End of File	End of File	EOF	EOF1	EOF1-EOF9	EOF1-EOF2
	User Trailer	UTL	None	UTL1-UTL9	UTL1-UTL9

- NOTES: • You may have Fixed, Variable, or Undefined length data records, but your records may not span blocks.
- You may record more than one file on a single volume.

Table A-4. Level 1 IBM Labels

Label Group Name	Label Set Name	Label Identifier	Required	Allowed	Processed
Beginning of Volume	Volume Header	VOL	VOL1	VOL1	VOL1
Beginning of File Section	File Header	HDR	HDR1	HDR1-HDR9	HDR1
	User Header	UHL	None	UHL1-UHL9	UHL1-UHL9
End of File Section	End of Volume	EOV	EOV1	EOV1-EOV9	EOV1
	User Trailer	UTL	None	UTL1-UTL9	UTL1-UTL9
End of File	End of File	EOF	EOF1	EOF1-EOF9	EOF1
	User Trailer	UTL	None	UTL1-UTL9	UTL1-UTL9

- NOTES:
- You can have Fixed, Variable, or Undefined length data records.
 - You may record more than one file on a single volume.
 - These labels correspond to those generated by the following IBM operating systems: BPS, BOS, TOS, and DOS.

Table A-5. Level 2 IBM Labels

Label Group Name	Label Set Name	Label Identifier	Required	Allowed	Processed
Beginning of Volume	Volume Header	VOL	VOL1	VOL1	VOL1
Beginning of File Section	File Header	HDR	HDR1	HDR1-HDR9	HDR1-HDR2
	User Header	UHL	None	UHL1-UHL9	UHL1-UHL9
End of File Section	End of Volume	EOV	EOV1	EOV1-EOV9	EOV1-EOV2
	User Trailer	UTL	None	UTL1-UTL9	UTL1-UTL9
End of File	End of File	EOF	EOF1	EOF1-EOF9	EOF1-EOF2

- NOTES:
- You may have Fixed, Variable, or Undefined length data records, but records may not span blocks.
 - You may record more than one file on a single volume.
 - These labels correspond to those generated by the IBM operating systems OS, OS/VS1, and OS/VS2.

User Labels

There are three types of User Labels: Volume, Header, and Trailer. They are useful if you want to specify information about a file which doesn't fit into the usual label categories - such as file creation time, machine configuration, or whatever. At the highest levels of label processing, you can specify up to nine of each type of label, which should be sufficient for most applications.

You can specify User Labels for each volume of a file, but, if you're going to use them, you have to build a label table into your program for each type of label you want to use. Each table will contain an entry count followed by an entry for each label which will point to the User Label area of your program. Each of your User Label areas can be up to 76 Bytes long. (The actual label is 80 Bytes long, but the system automatically supplies the first four bytes which contain the label identifier and number.)

During input processing, the system reads each User Label into its corresponding label area; on output, it takes the label from your label area and writes it to your tape.

One labor-saving feature to note: You don't have to build a unique table for each volume of your files. That is, the definition areas of several volumes can point to a single label table within your program. You can also modify the contents of your label table or that of a particular label area just by monitoring the status word in the processing part of your program.

Volume Initialization and Release

Before you can open a file on a labeled magnetic tape to process your records, you have to initialize the tape for label processing. You can do this three ways:

1. with the LBINIT utility (as described in the *INFOS Utilities Users' Manual*); or
2. with the system call .IINFOS; or
3. via runtime initialization.

However, once you have initialized a labeled magnetic tape, you must use the INFOS SAM processing and utility functions; you cannot process it with standard operating system (RDOS) calls.

How to Initialize and Release Tapes Through System Calls

The system call to partially or fully initialize a magnetic tape volume for label processing is .IINFOS. When you use this call, the system will automatically rewind your selected drive. This call also allows you to access a volume by its identifier rather than by a device specifier, since it equates the volume identifier with the selected drive. For example, you can access a volume by its identifier PAYROLL, rather than MT0. If your installation has multiple tape drives, you can also use .IINFOS to initialize several volumes before you process them.

When you partially initialize a volume with .IINFOS, the system reads the label group at the beginning of the volume which contains the volume identification, header labels, and user labels (if any). The system then compares the volume identification in the label of the first volume to the volume identifier which you specify. If the two identifiers are not the same, you'll get the error message, "VOL ID DOESN'T MATCH". You can also tell the system not to compare identifiers, in which case the system will initialize the volume regardless of its volume identification.

You perform a full initialization in the same way that you do a partial. When you fully initialize a tape, however, the system writes the volume label plus dummy header and trailer labels on the tape. Consequently, you will lose any previous information on a volume when you fully initialize it. If you're using virgin tape, you *must* fully initialize it before you use it. Then you can either open the tape for use in your current run, or release it and partially initialize it later.

If the system encounters no errors in either a full or partial initialization, it sets a flag which indicates that the volume has been initialized for label processing. Once this is set, you cannot process the volume with standard operating system (RDOS) calls.

After initialization, if the system encounters an error when you are pre-opening, opening, or attempting your first read or write request, it will automatically rewind the tape so that you won't be mispositioned. This rewind will occur whether you requested it or not.

When you have finished processing a volume, release it with the system call .RLSE.

Runtime Initialization and Release (General)

The INFOS system also allows you to initialize tapes at runtime. This can be very useful when you have not been able to initialize a tape beforehand, or if you have a multivolume labeled tape file but fewer tape drives than file volumes. This feature lets you partially initialize tapes as you need them for processing and release them when you're finished. In order to use this feature, however, you must enable runtime initialization and release when you set up your file. The system will not perform runtime initialization if the tape is already initialized - it will simply ignore the runtime request.

If you're processing a multivolume file, you will probably want to mount all your tapes before you start in order to save time. However, if you have more file volumes than tape drives, you must request runtime release for each volume that you are replacing with another. After you mount the next volume, you must request runtime initialization for it and supply the necessary information described below.

Runtime Initialization

When the system wants you to set up a tape so that it can initialize it, you will receive the messages MOUNT VOLUME (name) and ENTER DEVICE SPECIFIER on your system console. This tells you to do two things:

1. Mount the tape, and
2. Tell the system the device specifier of the drive on which you mounted the tape (e.g., MT0).

The system will then try to initialize the tape. If the volume identifier on the tape does not match the one you specified when you set up the file, the system will send you the following message: "VOLUME (name) MOUNTED, USE ANYWAY?" If you still want to use the volume you mounted, type in "YES" or "Y". If you respond "NO" or "N", the system will ask you "CANCEL (Y or N)?" If you then answer "Y" or "YES", the system will cancel your processing request and rewind the tape. If you answer "N" or "NO", the system will rewind the tape so that you can mount another.

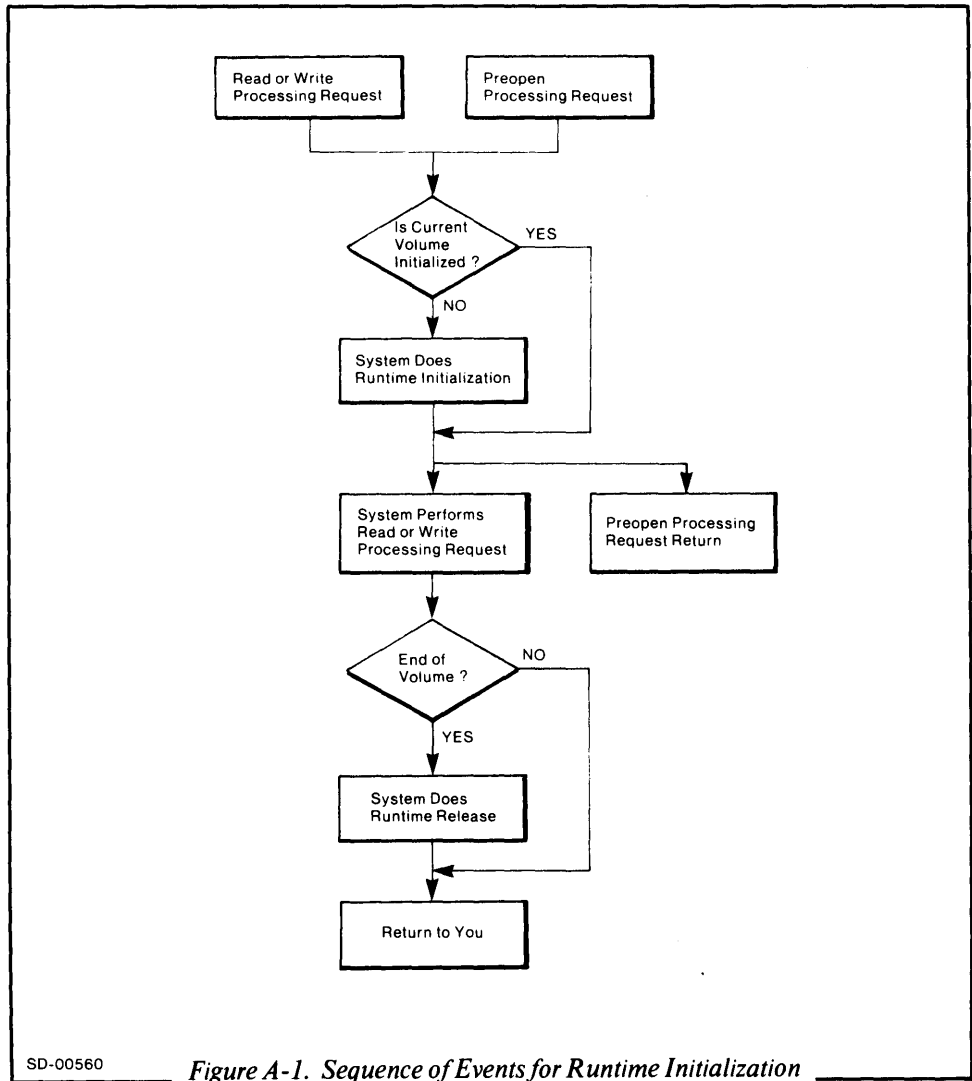
NOTE: While the system is processing a runtime initialization request, it will wait indefinitely for you to respond to a prompt. However, while the system is waiting for you to answer, it suspends all other tasks it is processing at the time in both the background *and* the foreground. In other words, no one else can process data while the system is waiting for you. Therefore you should secure your desired tape or respond to the prompt as quickly as possible to avoid incurring the wrath of your fellow users.

Runtime Release

Runtime release is an operation the system automatically performs when it reaches the end of a volume or when it encounters any situation that requires the close of the volume or the file. After the system has released your tape, it will tell you (via your

system console) the volume identifier and the drive specifier. If you desire runtime release as part of your close request, you must also specify rewind or else the system will not perform the runtime release.

The diagram in Figure A-1 shows the sequence of events for runtime initialization and release.



SD-00560

Figure A-1. Sequence of Events for Runtime Initialization and Release

Processing Labeled Magnetic Tapes

After you have initialized your tape and gone through the pre-opening procedures, you're ready to open the tape for processing. The first thing to note here is that the physical opening of your tape occurs when you issue your first read or write request. On a write request, the system compares the expiration date of the volume to the current date; on a read request, the system compares the record format of the tape to that which you requested. If either of these don't match, you'll get an error message. However, you can specify Undefined records when you open a file for reading; then the system will read all the records on the tape up to the next tape mark, regardless of this format.

Next, you'll want to position yourself to your first desired record. To do this, the system needs a Sequence Number, a Filename and a Generation Number. Once you have decided whether to supply or default these, the system will follow these three tape positioning rules:

1. A. The system looks at the Sequence Number you request, if any, first.
 - B. All subsequent searches will be *forward only* from that Sequence Number position.
2. A. The system next verifies the Filename of the record whose Sequence Number you requested, unless you default the Filename.
 - B. If you did not specify a new Sequence Number, the system will search forward from the end of the last record you read.
3. A. If the Filename matches, the system verifies the Generation Number (if you gave one) and proceeds in a forward search.
 - B. If the system does not match either the Filename or the Generation number, it will let you know via an error message.

As these rules indicate, you can default the Filename and Generation Number if you don't know (or care about) them, but your position on the tape is always governed by the Sequence Number. If you default this number by answering -1, the system will automatically give you the next record on the tape. If you do not default the Sequence Number, the system will position the tape to the record whose Sequence Number you specify. Then it will compare the Filename and Generation Number of that record to those which you specified (if any). If they don't match, you'll get an error message. So it's easy to move forward through the records on your tape - just give -1 or a Sequence Number.

Now, what if you want to go backwards on the tape to a record you read or passed over earlier? There are two ways to do this: Give the system the exact Sequence Number for the record you want to access, or rewind the tape to the beginning and search for the record using the -1 Sequence Number.

For example, suppose you have a file named PAYROLL whose records represent several months of weekly payroll accounting. To keep things simple, you have numbered the records 0101, 0102, 0103, 0104, 0201, 0202, etc. The Filename PAYROLL will be part of each record, as will the numbers 0101, 0102, 0103, etc. Therefore, the name of the record which is Sequence Number 1 is PAYROLL 0101; Sequence Number 2 is PAYROLL 0102, etc. If you have just read the record PAYROLL 0502 (which is Sequence Number 18), and you want to go back to the record whose name is PAYROLL 0203, you can do so in two ways. 1) You can code the equivalent of READ 7 if you know that PAYROLL 0203 is, indeed, Sequence Number 7. Or if you don't know *what* the exact Sequence Number of PAYROLL 0203 is: 2) You can rewind the tape to its beginning and search forward, using -1 as a Sequence Number, until you find PAYROLL 0203.

If you had simply told the computer to search for PAYROLL 0203 and used the Sequence Number -1, the system would have gone forward from your current position (Sequence Number 18) to the end of the tape. Since it would not have found the record whose Filename was PAYROLL and whose Generation Number was 0203, it would have sent you the message, TAPE FILE NOT FOUND. At this point you would probably scratch your head and say, "!!@**#\$*!!! I *know* that record is on this tape". And you'd be right. It *is* on the tape, but the system will only search forward unless you specify a Sequence Number or ask for rewind before you search. So if you want to find a record which you may have already passed, follow either of the steps described above.

Positioning When Writing

So far we've dealt primarily with reading from a tape. However, the same general positioning principles apply when you're writing to a tape - except that it's much simpler.

When you want to (re)write a record, the system looks *only* at the Sequence Number; it will not verify the Filename or Generation Number when you reposition. So if you're not absolutely sure of the Sequence Number, read the record to which you have repositioned *before* you rewrite it.

There is one final caution about processing labeled magnetic tapes: If you give the system a Sequence Number which is greater than the number of records on the tape, the hardware will position the tape beyond the end-of-tape markers and the INFOS system will hang. (Ouch!) This is not a desirable situation.

Graphic Arts Section

The diagrams in Figures A-2 to A-6 and Tables A-6 to A-12 show the labels used in each label type and level and their sequence on tape.

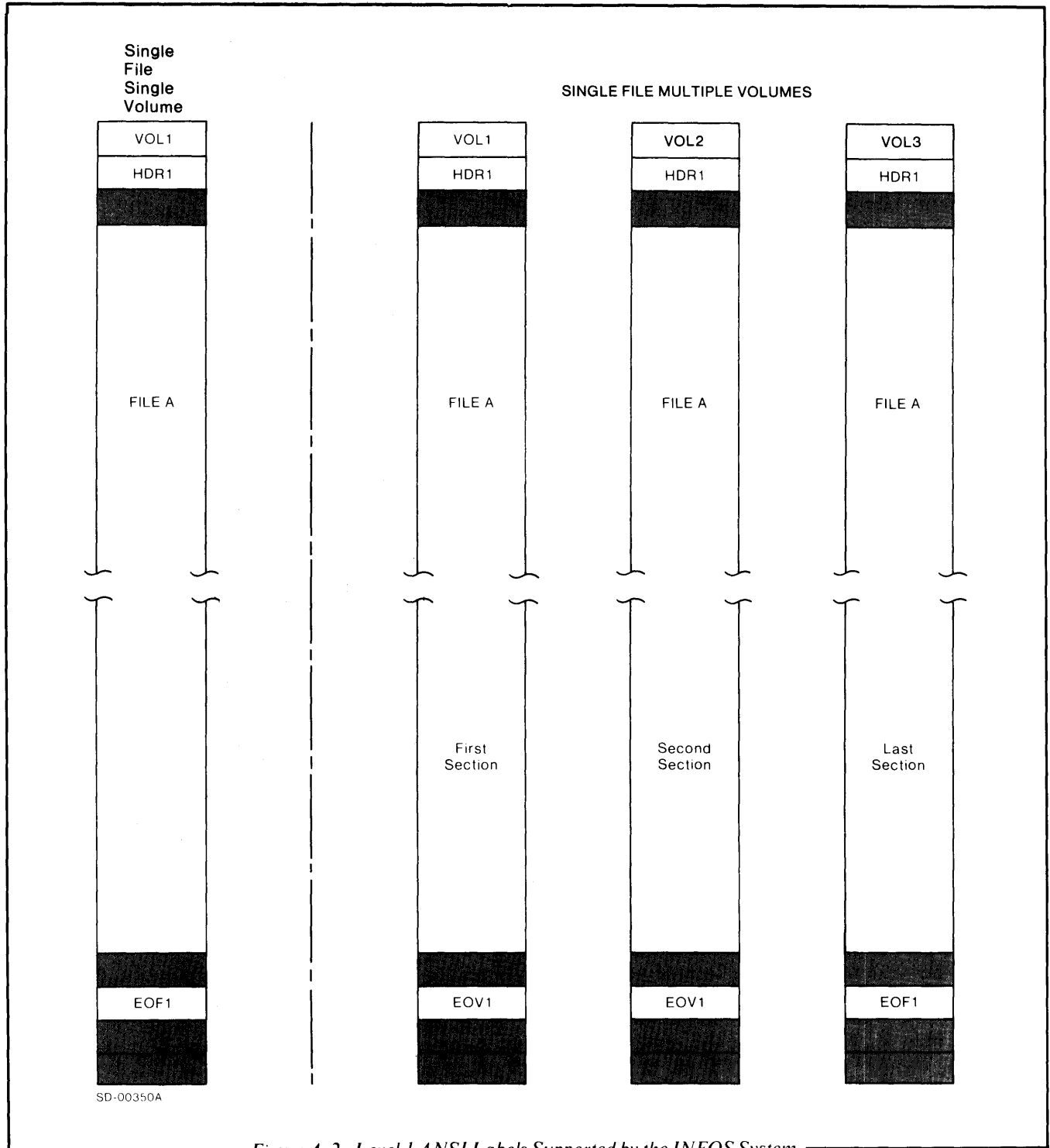
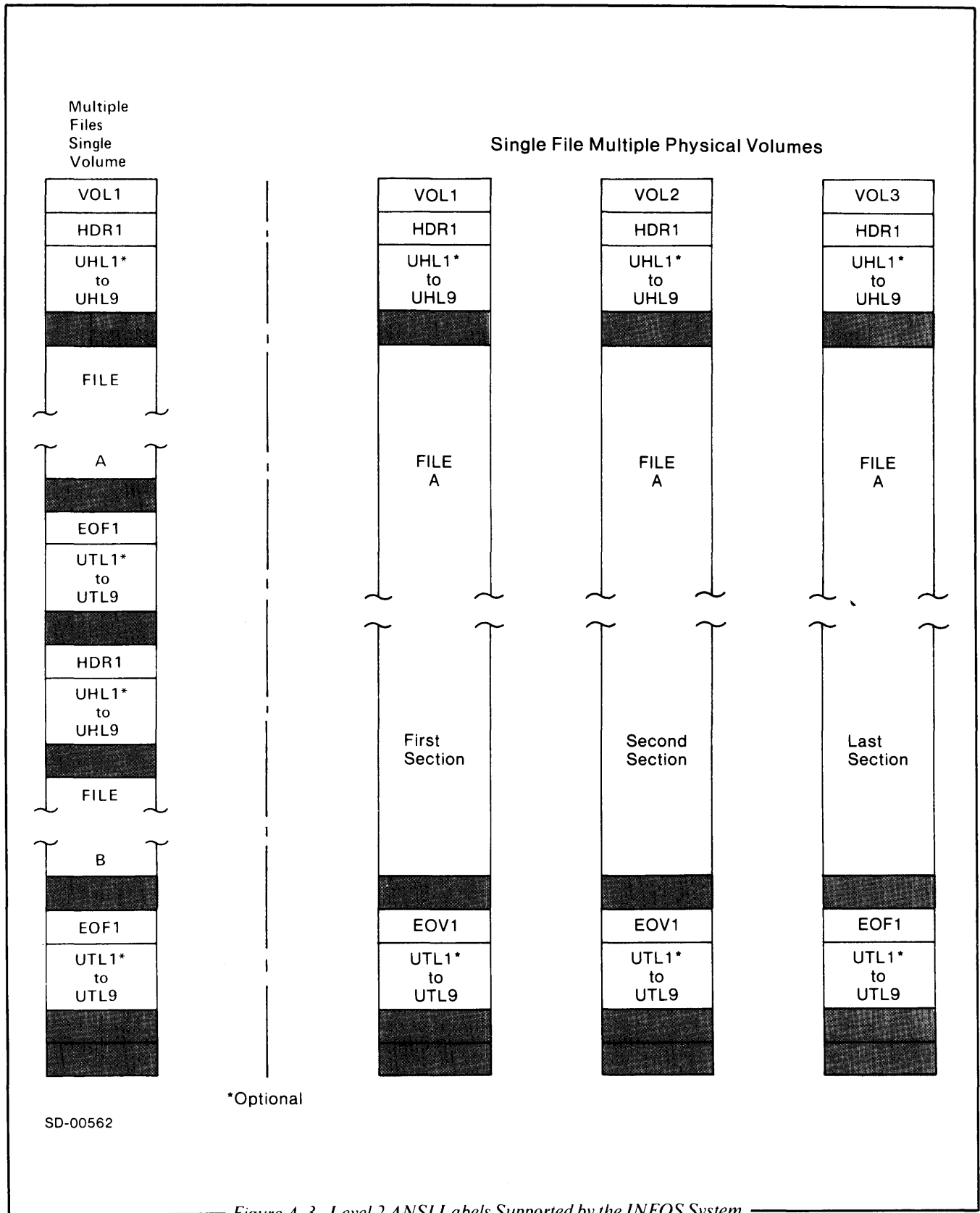


Figure A-2. Level 1 ANSI Labels Supported by the INFOS System



*Optional

SD-00562

Figure A-3. Level 2 ANSI Labels Supported by the INFOS System

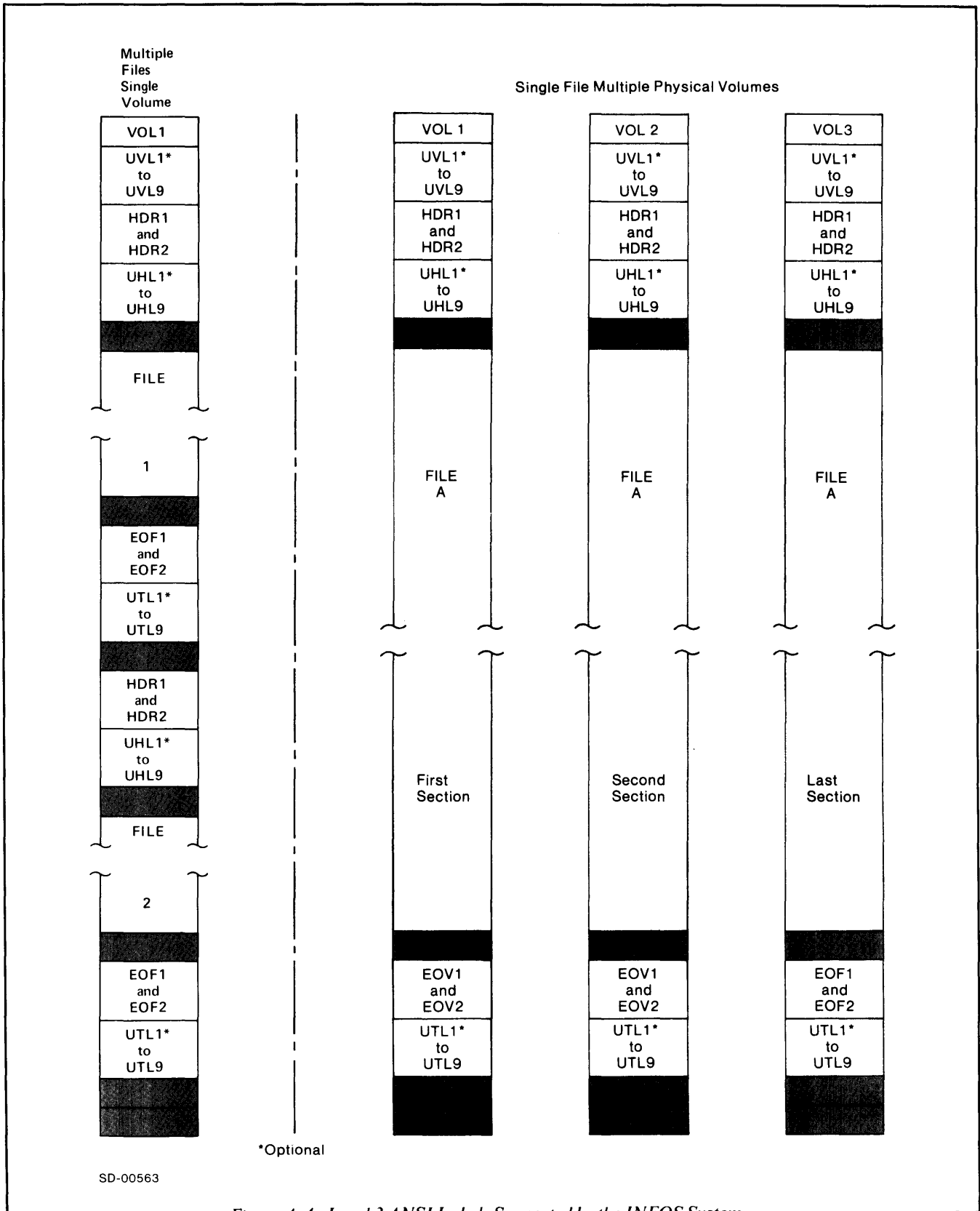


Figure A-4. Level 3 ANSI Labels Supported by the INFOS System

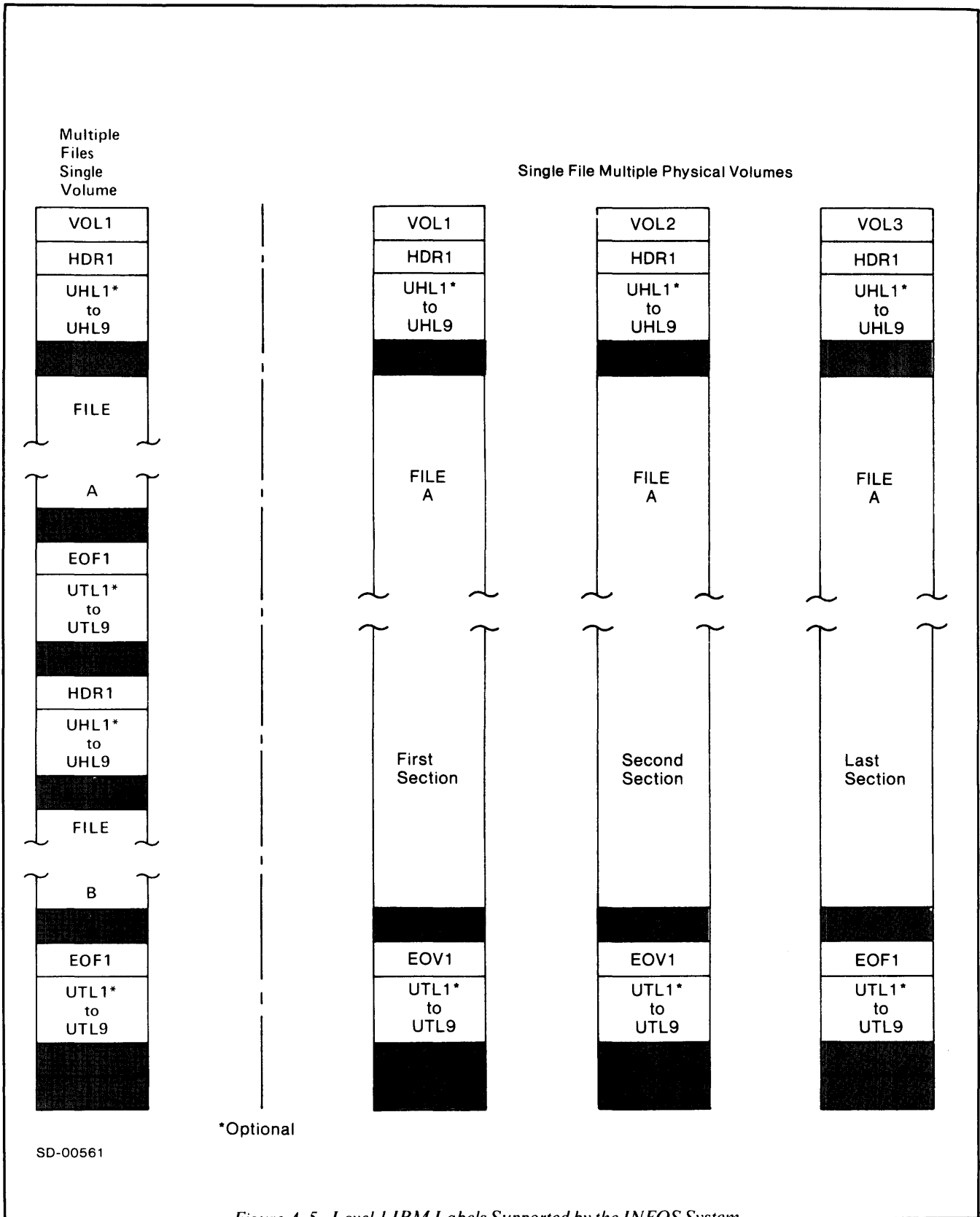


Figure A-5. Level I IBM Labels Supported by the INFOS System

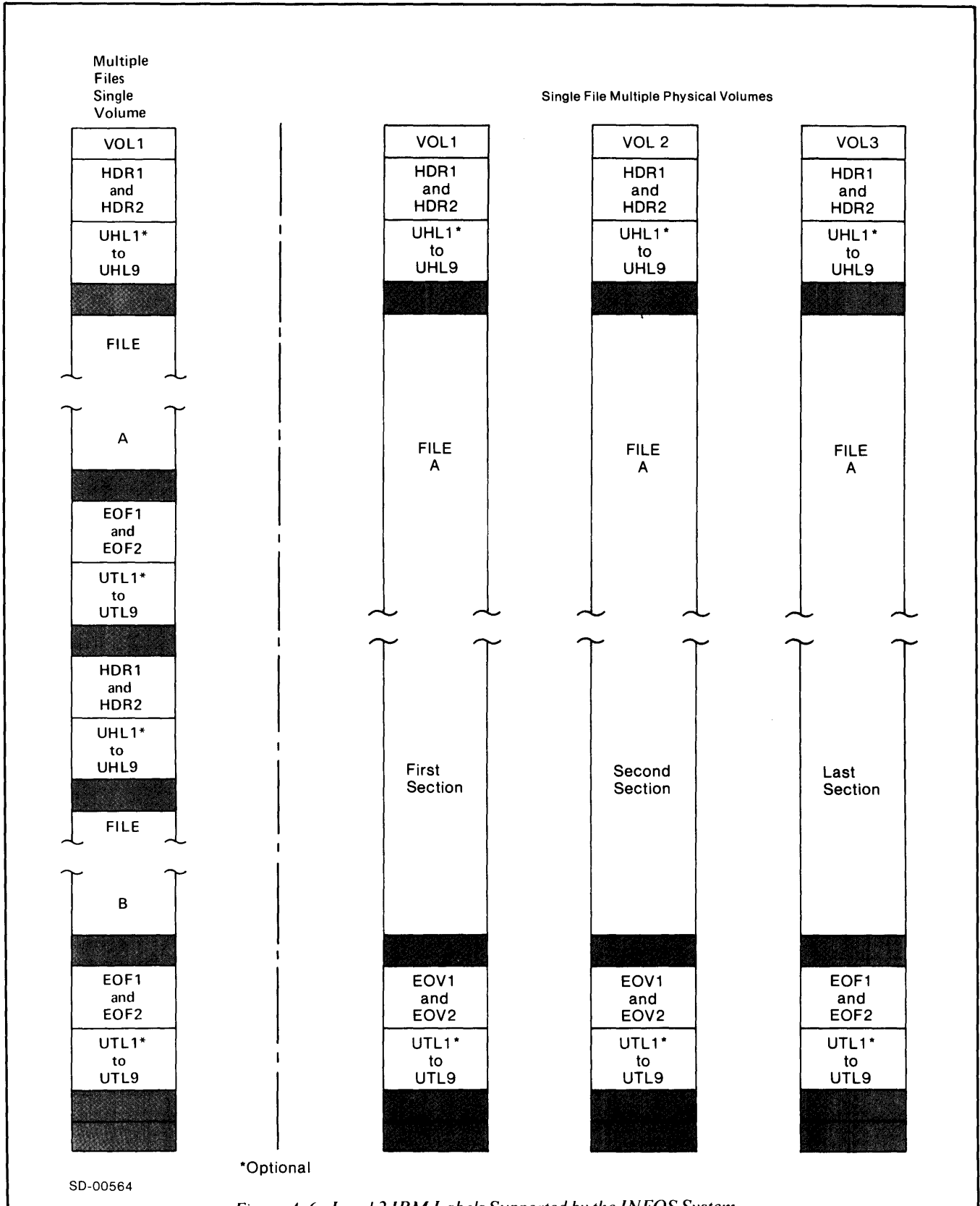


Figure A-6. Level 2 IBM Labels Supported by the INFOS System

Table A-6. ANSI Standard Volume Label Format

Byte Position	Field Name	Field Size	Field Content
1 to 3	LABEL IDENTIFIER	3	VOL
4	LABEL NUMBER	1	1
5 to 10	VOLUME IDENTIFIER	6	Up to six alphanumeric characters assigned by the owner to permanently identify this volume.
11	ACCESSIBILITY	1	A single alphanumeric character indicating restrictions on access to this volume. A blank space indicates no restrictions.
12 to 37	RESERVED	26	Should contain blanks.
38 to 51	OWNER IDENTIFICATION	14	Up to 14 alphanumeric characters that identify the owner of this volume.
52 to 79	RESERVED	28	Should contains blanks.
80	LABEL STANDARD VERSION	1	A one-digit integer that indicates the ANSI version to which the labels and record formats on this volume conform. The digit 3 means: ANSI x 3.27 - 1969.

Table A-7. ANSI Standard User Volume Labels

Byte Position	Field Name	Field Size	Field Content
1 to 3	LABEL IDENTIFIER	3	UVL
4	LABEL NUMBER	1	1 to 9
5 to 80	USER INFORMATION	76	Up to 76 alphanumeric characters. Not intended for use in an interchange environment.

Table A-8. ANSI Standard HDR 1, EOVI, EOF 1 Labels

Byte Position	Field Name	Field Size	Field Content
1 to 3	LABEL IDENTIFIER	3	HDR or EOVI or EOF
4	LABEL NUMBER	1	1
5 to 21	FILE IDENTIFIER	17	Up to 17 alphanumeric characters assigned by the originator to identify the file.
22 to 27	FILE-SET IDENTIFIER	6	Up to six alphanumeric characters to identify this file set among other file sets.
28 to 31	FILE SECTION NUMBER	4	A four-digit number to identify this section among other sections of this file.
32 to 35	FILE SEQUENCE NUMBER	4	A four-digit number to identify this file among the files of this file set.
36 to 39	GENERATION NUMBER	4	A four-digit number to distinguish among successive generations of this file.
40 to 41	GENERATION VERSION NUMBER	2	A two-digit number to distinguish among successive iterations of the same generation.
42 to 47	CREATION DATE	6	One space followed by a two-digit number for the year, followed by a three-digit number for the day of the year. Space and five zeros means no date.
48 to 53	EXPIRATION DATE	6	One space followed by a two-digit number for the year, followed by a three-digit number for the day of the year. Space and five zeros means file expired.
54	ACCESSIBILITY	1	One alphanumeric character indicating restrictions on access to this file. Space means no restrictions.
55 to 60	BLOCK COUNT	6	For EOVI and EOF, six digits giving the number of data blocks since the preceding Beginning-of-File Label Group. For HDR, all zeros.
61 to 73	SYSTEM CODE	13	Up to 13 Alphanumeric characters identifying the system that recorded the file.
74 to 80	RESERVED	7	Blank filled.

Table A-9. ANSI Standard HDR 2, EO V 2, EOF 2 Labels

Byte Position	Field Name	Field Size	Field Content
1 to 3	LABEL IDENTIFIER	3	HDR or EO V or EOF
4	LABEL NUMBER	1	2
5	RECORD FORMAT	1	F = Fixed Length D = Variable Length U = Undefined Length
6 to 10	BLOCK LENGTH	5	Five digits specifying the maximum number of characters per block.
11 to 15	RECORD LENGTH	5	Five digits specifying record length. F = actual record length D = maximum record length including count field U = content of Record Length field is undefined
16 to 50	RESERVED	35	Blank fill.
51 to 52	BUFFER-OFFSET LENGTH	2	Two digits specifying the number of characters of any additional field inserted before first record in the data block.
53 to 80	RESERVED	28	Blank fill.

Table A-10. IBM Standard Volume Label Format

Byte Position	Field Name	Field Size	Field Content
1 to 3	LABEL IDENTIFIER	3	VOL
4	LABEL NUMBER	1	1
5 to 10	VOLUME SERIAL NUMBER	6	A unique identification code assigned to the volume when it enters the system or assigned by the operator when the volume is labeled.
11	RESERVED	1	Must contain a zero.
12 to 21	RESERVED	10	Not used for magnetic tape files. Should contain blanks.
22 to 31	RESERVED	10	Should contain blanks.
32 to 41	RESERVED	10	Should contain blanks.
42 to 51	OWNER NAME AND ADDRESS	10	Names the specific owner of the volume. Any code or name can be used.
52 to 80	RESERVED	29	Should contain blanks.

Table A-11. IBM Standard HDR 1, EOVS 1, EOF 1 Labels

Byte Position	Field Name	Field Size	Field Content
1 to 3	LABEL IDENTIFIER	3	HDR or EOVS or EOF
4	LABEL NUMBER	1	1
5 to 21	DATA SET IDENTIFIER	17	Up to 17 alphanumeric characters giving the data set name. Can include the generation and version number.
22 to 27	DATA SET SERIAL NUMBER	6	Up to six alphanumeric characters giving the volume serial number for this volume.
28 to 31	VOLUME SEQUENCE NUMBER	4	A four-digit number giving the order of this volume in a multivolume group.
32 to 35	DATA SET SEQUENCE NUMBER	4	A four-digit number giving the relative position of the data set in a multidata-set group.
36 to 39	GENERATION NUMBER	4	A four-digit number giving the absolute generation number of the data set or blanks.
40 to 41	GENERATION VERSION NUMBER	2	A two-digit number giving the data set version number or blanks.
42 to 47	CREATION DATE	6	A six-digit number giving the year and the day of the year the data set was created.
48 to 53	EXPIRATION DATE	6	A six-digit number giving the year and the day of the year the data set can be scratched.
54	DATA SET SECURITY	1	A one-digit number giving the security status of the data set.
55 to 60	BLOCK COUNT	6	In EOVS and EOF labels the number of data blocks is given. In HDR labels zeros must be given.
61 to 73	SYSTEM CODE	13	A unique 13-byte code that identifies the operating system.
74 to 80	RESERVED	7	Should contain blanks.

Table A-12. IBM Standard HDR 2, EOVS 2, EOF 2 Labels

Byte Position	Field Name	Field Size	Field Content
1 to 3	LABEL IDENTIFIER	3	HDR or EOVS or EOF
4	LABEL NUMBER	1	2
5	RECORD FORMAT	1	F = Fixed Length V = Variable Length U = Undefined Length
6 to 10	BLOCK LENGTH	5	F = Must be a multiple of logical record length. V = Must give maximum block length and four bytes. U = Must give maximum block length.
11 to 15	RECORD LENGTH	5	F = Must give logical record length. V = Must give maximum block length and four bytes. U = Must be zero filled.
16 to 34	NOT USED	19	Blank fill.
35 to 36	TAPE RECORDING TECHNIQUE	2	T <input type="checkbox"/> = Odd parity with translation. C <input type="checkbox"/> = Odd parity with conversion. E <input type="checkbox"/> = Even Parity no translation. ET = Even parity with translation. <input type="checkbox"/> <input type="checkbox"/> = Odd parity-no translation or conversion.
37	CONTROL CHARACTERS	1	A = ASCII control characters M = Machine control characters <input type="checkbox"/> = No control characters.
38	NOT USED	1	Blank fill.
39	BLOCK ATTRIBUTE	1	B = Blocked records <input type="checkbox"/> = Not blocked
40 to 80	NOT USED	41	Blank fill.

End of Appendix

Appendix B

Subindex and Database File Properties

Introduction

As you know by now, ISAM and DBAM files have two independent parts -- an index file and a database file -- which the INFOS system links into a single unit for processing. What you may not realize is that ISAM and DBAM databases have identical physical properties, as do an ISAM index file and any single DBAM subindex. In this appendix, we'll examine the properties of subindexes and databases and the effects your design decisions have on them.

Subindexes

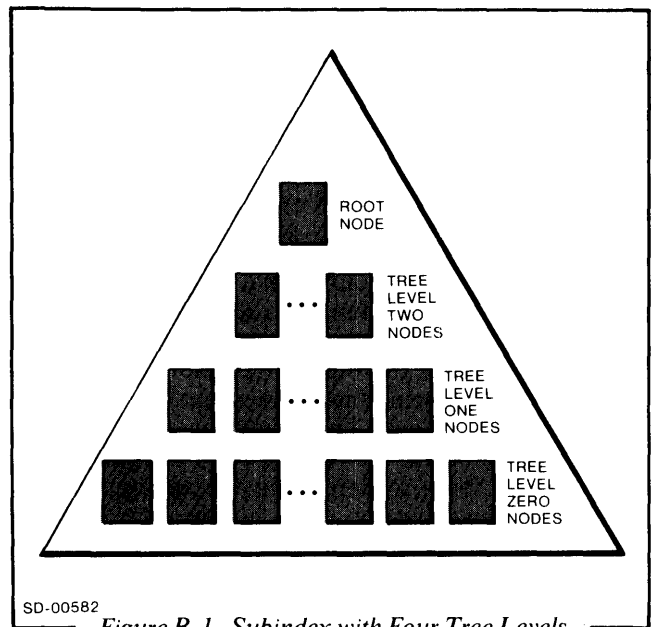
Normally, you'll want to design an index structure which can hold your maximum expected number of entries, but which you can also access as quickly as possible. Furthermore, you'll want to be able to use as many buffers as necessary, and to keep your disk storage space to a minimum. This is a good general goal, but remember, also, that it is your application which will ultimately determine the number and the properties of your subindexes. So, in order to achieve this goal, let's examine the structure of a subindex and how you can design one to best suit your needs.*

Nodes, which contain your index entries, are the basic "building blocks" of a subindex. The system allocates space for a subindex in node-sized pieces, and, as the number of entries (and therefore the number of

*If you are familiar with techniques for index file storage, note that subindexes in Data General's INFOS system are stored internally as a type of multiway search tree, in particular, a variant of B-trees. Adjacent nodes at the same level are cross-linked to simplify sequential operations. Key pointers in a node are ordered to allow fast binary searching for specific key values. Our Key Compression is a form of front-end compression which stores common prefixes of adjacent keys in a node only once.

nodes) in a subindex increases, the system places the various nodes on different levels within the subindex. We call these different levels *tree levels*; any subindex can have up to 256 of them, although you will probably never need more than three or four. Figure B-1 shows a subindex with four tree levels.

Note that the tree levels are numbered sequentially from zero at the bottom to the single Root Node at the top of the structure. The Root Node is the first node built in a subindex; as long as it is the only node in the subindex, it is also the level zero node. When you first write entries in a subindex, the system places them in the Root Node. As you continue to write entries, the Root Node grows from no entries to its maximum size.



SD-00582
Figure B-1. Subindex with Four Tree Levels

(This maximum depends on your index file's page, node, and entry sizes; we'll discuss how you can compute these a little later.) When the Root Node reaches its maximum count, it moves its contents into two new nodes, which become tree level zero. The system then creates two new index entries in the Root Node which point to the entries in level zero.

As you continue to write index entries in this subindex, the system places them in nodes at tree level zero. As it creates each new level zero node, the system also adds an entry to the Root Node. Eventually, the Root Node will again grow to its maximum count, and there will be one full node at level zero for each entry in the Root Node. When this happens, the Root Node will again split in two, and its contents become tree level one, that is, an intermediate level of nodes between the Root Node and level zero. In other words, as you write your index entries, the system puts them into nodes at tree level *zero* and builds a tree structure to access them.

Now you may ask, "What is the maximum number of entries that any node at any tree level can contain?" This number is called the *branching factor*, and the answer is: "It depends on your page size, the length of your index entries, and whether you use key compression".

Your *page size* determines the depth of your subindexes. (Page size, remember, is the length of data transferred between an index file and its I/O buffers, and pages consist of nodes.) Node size is normally six bytes less than your page size. Therefore, the larger your page size, the fewer subindex levels the system needs, because large nodes contain more index entries than small ones. In other words, the system doesn't have to set up three or four tree levels if your page size allows large enough nodes to contain all the subindex entries on one or two levels.

The length of your *index entries* depends on your maximum key size, your partial record length, and whether or not you allowed subindexing. And when you take all these factors together with your page size, you can figure out the branching factor for any given single node by slipping their values into one or more of the formulas below. The formulas give you the branching factors on the basis of one node per page, or you can use them to determine the appropriate page size for a given subindex, or the number of tree levels a subindex will require, and/or the number of entries at level zero.

Licensed Material - Property of Data General Corporation

Note that these formulas assume full nodes, keys whose lengths are very nearly the same, and no key compression. If your key lengths tend not to be close to their maximum, the branching factor value you get will be too small. Similarly, your nodes will probably be full only if you entered the data in the same sequence as the keys are stored. Therefore, keep these conditions in mind when using the formulas; they may not accurately predict performance if your application varies greatly from the average.

1. When the Root Node is not also the level zero node, its branching factor is:

$$\frac{\text{Page Size} - 42}{\text{Max Key Length} + 10}$$

2. When the Root Node is also the level zero node, its branching factor depends on whether or not you allowed subindexing.

- A. If you allowed subindexing, use:

$$\frac{\text{Page Size} - 42}{\text{Max Key Length} + \text{Partial Rec Length} + 14}$$

- B. If you did not allow subindexing, use:

$$\frac{\text{Page Size} - 42}{\text{Max Key Length} + \text{Partial Rec Length} + 10}$$

3. When the level zero node is also the Root Node, use 2A or 2B. However, when the level zero nodes are *not* the Root Node, their branching factor also depends on whether or not you allowed subindexing.

- A. If you allowed subindexing, use:

$$\frac{\text{Page Size} - 28}{\text{Max Key Length} + \text{Partial Rec Length} + 14}$$

- B. If you did not allow subindexing, use:

$$\frac{\text{Page Size} - 28}{\text{Max Key Length} + \text{Partial Rec Length} + 10}$$

4. The branching factor for any intermediate level node is:

$$\frac{\text{Page Size} - 28}{\text{Max Key Length} + 10}$$

Licensed Material - Property of Data General Corporation

Notes:

1. The lower half of each formula reflects your index entry size. If the sum of its components is not an even number, round it up to the next even number.
2. The top half of each fraction is the page size minus the number of bytes the system needs for one node.
3. You can use the INDEXCALC utility to perform all branching factor calculations that you need. See the *INFOS System Utilities Manual* for further details on how to use INDEXCALC.

Now let's examine these formulas with some real numbers.

In Chapter 5 of Part One, we described a DBAM database that had two independent indexes: one for a mailing list, and one for customer accounts. Under the mailing list index, we also assigned a subindex to carry customer's names and addresses in partial records. Now let's assume that we also have a customer base of 50,000 entries, that our keys occupy 5 bytes each, and that the partial records are 50 bytes long. Since the purpose of this subindex is to allow the rapid production of address labels, there is no need for its entries to have subindexes. We can use these numbers to examine the effects of page size on the number of tree levels and on the amount of disk storage space that you'll need for the subindex.

First we'll try the smallest possible page size -- 512 bytes. To find how many level zero nodes we'll need, we'll calculate the number of entries that can fit in a maximum size level zero node and divide it into 50,000. Using formula 3B, above, we get:

$$(3B) \quad \frac{\text{Page Size} - 28}{\text{Max Key Length} + \text{Partial Rec Length} + 10}$$

$$\frac{512 - 28}{5 + 50 + 10} = \frac{484}{65} = 7 \text{ entries in each level zero node}$$

$$50,000 / 7 = 7143 \text{ nodes required at level zero}$$

Now we need to find out how many entries will be in the nodes at all the other levels. To do this, we use formulas 4 (for all the intermediate level nodes), and 1 (for the Root Node):

$$(4) \quad \frac{\text{Page Size} - 28}{\text{Max Key Length} + 10}$$

$$\frac{512 - 28}{5 + 10} = \frac{484}{15} = 32 \text{ entries in each intermediate level node}$$

$$(1) \quad \frac{\text{Page Size} - 42}{\text{Max Key Length} + 10}$$

$$\frac{512 - 42}{5 + 10} = \frac{470}{15} = 31 \text{ entries in the Root Node}$$

Since we know that:

- we need 7143 level zero nodes; and
- each node in level zero has an entry in level one; and
- each node in level one contains 32 entries,

we can determine how many level one nodes we need simply by dividing 7143 by 32. Thus we'll need 224 nodes on tree level one. Now, since the Root Node only holds 31 entries and not 224, this means that we'll have to have a level two. And to determine the number of nodes at level two, we divide 224 (the number of nodes on level one) by 32 (the number of entries in each intermediate level node). This gives us 7 level two nodes, and it means that we don't need any more tree levels because the Root Node can hold up to 31 entries.

Therefore, we now have nodes with the following branching factors (i.e., entries per node):

- Root Node = 7
- Level two nodes = 32
- Level one nodes = 32
- Level zero nodes = 7

Figure B-2 illustrates this particular subindex.

This four-level structure can accommodate 50,000 index entries (keys plus partial records) at level zero, but it may take four disk accesses to retrieve any entry at level zero. This is all done on the basis of one node per page, and, since our page size (512 bytes) in this example is the same size as a disk block, we'll need 7375 blocks of disk storage for this subindex (7143 blocks for level zero, plus 224 for level one, plus 7 for level two, plus one for the Root Node).

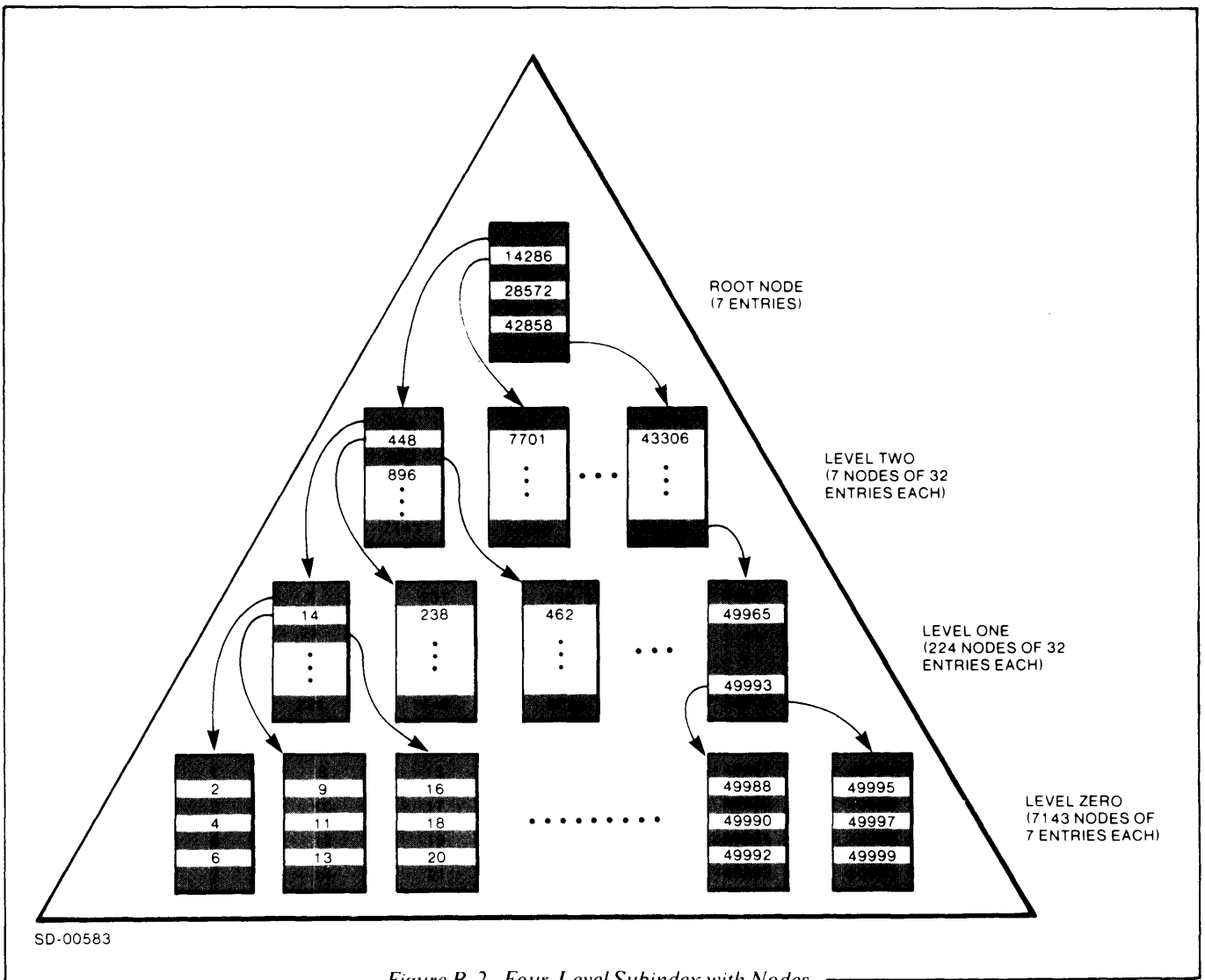


Figure B-2. Four-Level Subindex with Nodes

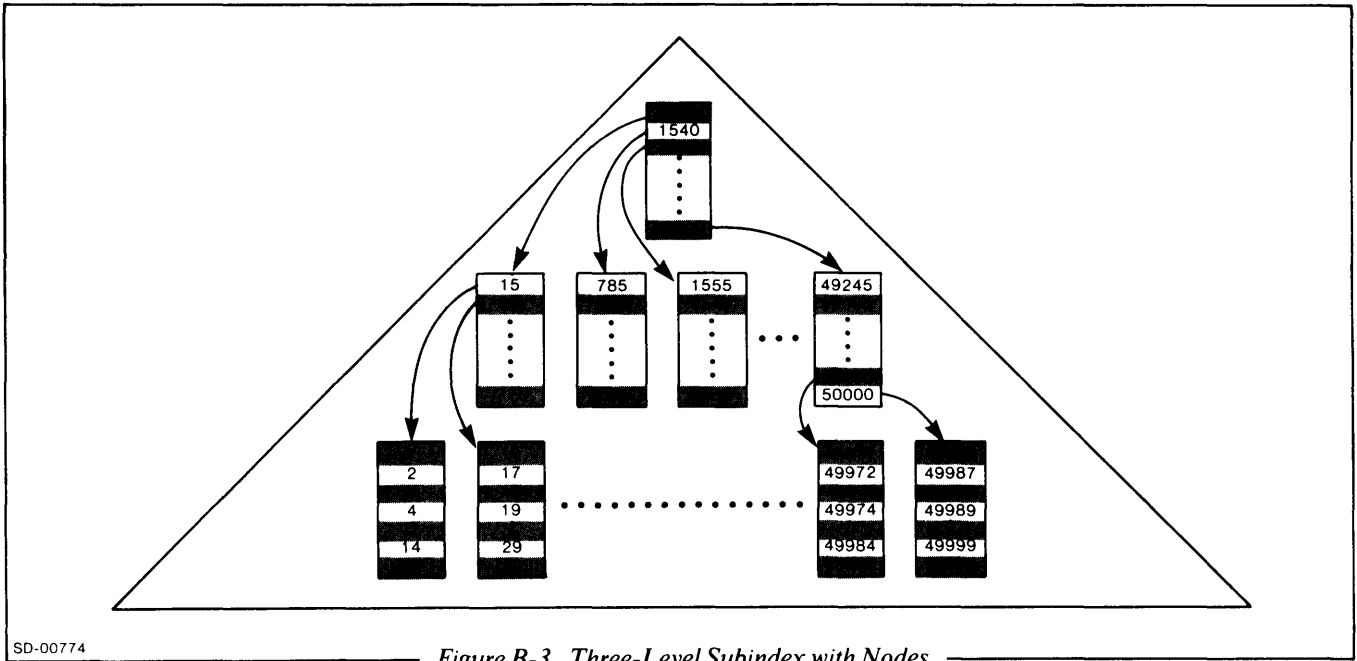
Licensed Material - Property of Data General Corporation

Now, if we double our page size to 1024 bytes, we find that each level zero node can hold 15 entries, which means that we only need 3334 nodes at level zero. Also, each level one node can hold 66 entries, so we'll only need 51 nodes there. Finally, we find that the Root Node can hold 65 entries, so we won't need any intermediate tree levels between the Root Node and level one. Thus, we have a *three-level* structure which can store 50,000 entries, just by doubling the page size. The branching factors for this subindex (illustrated in Figure B-3) are as follows:

- Root Node = 51
- Level one nodes = 66
- Level zero nodes = 15

Our total number of nodes is now 3386 (3354 + 51 + 1), and each one requires two disk blocks (1024 bytes); therefore we only need 6772 blocks of disk storage for this subindex.

By increasing our page size from 512 to 1024 bytes, we reduced the number of tree levels (and, therefore, disk accesses per entry) by one, and the amount of necessary disk storage by 603 blocks. Note, however, that while we designed this subindex to hold 50,000 entries, that is not its maximum capacity; it will hold 50,000 entries if we only use 51 Root Node entries. But, since the maximum number of entries which we can have in the Root Node is 65, the absolute



SD-00774

Figure B-3. Three-Level Subindex with Nodes

maximum capacity of this subindex is actually 60,450 entries. We derive this figure by multiplying the Root Node branching factor (BF) by the level zero BF, then multiplying that product by the intermediate level BF raised to the power which corresponds to the number of intermediate levels. In other words:

$$\left(\frac{\text{page size} - 42}{\text{max key length} + 10} \right) * \left(\frac{\text{page size} - 28}{\text{max key length} + \text{partial rec length} + 10} \right) * \left(\left(\frac{\text{page size} - 28}{\text{max key length} + 10} \right)^n \right) =$$

maximum entry capacity of any subindex

where n equals the number of intermediate levels. In our case, we did this:

$$\left(\frac{1024-42}{5+10} \right) * \left(\frac{1024-28}{5+50+10} \right) * \left(\left(\frac{1024-28}{5+10} \right)^1 \right) = 65 * 15 * 62 = 60,450.$$

On the other hand, our earlier example with the four-level tree will hold 222,208 entries:

$$\left(\frac{512-42}{5+10} \right) * \left(\frac{512-28}{5+50+10} \right) * \left(\left(\frac{512-28}{5+10} \right)^2 \right) = 31 * 7 * 32^2 = 222,208.$$

We square the intermediate level BF (32) because there were two intermediate tree levels. Note that you wouldn't need any additional tree levels if you expanded this subindex to its maximum, but you would, naturally, need additional disk storage space.

Licensed Material - Property of Data General Corporation

Now let's take a different, smaller example. We'll use a relatively small subindex with 5000 entries, each of which is 26 bytes long, and has a 30-byte partial record field. Furthermore (just to complicate things), some of the index entries will have their own subindexes. Because we'll be using keyed access to get to this subindex and we want top access speed, we'll also want the subindex to have no more than two tree levels. This means that we'll need a fairly large page size -- 2048 bytes, for instance, or 4096. Let's start with 2048.

First we have to find out how many nodes we'll need at level zero, so we'll apply formula 3B, above. This will tell us the number of entries per node, which we can then divide into 5000 to get the number of required nodes.

$$(3B) \quad \frac{\text{Page Size} - 28}{\text{Max Key Length} + \text{Partial Rec Length} + 14}$$

$$\frac{2048 - 28}{26 + 30 + 14} = 28 \text{ entries in each level zero node}$$

$$5000 / 28 = 179 \text{ nodes needed at level zero}$$

Now we need to find out if the Root Node branching factor is equal to (or greater than) 179. If it is, we're all set; if not, then we'll need at least three tree levels. Using formula 1, we get:

$$(1) \quad \frac{\text{Page size} - 42}{\text{Max Key Length} + 10}$$

$$\frac{2048 - 42}{26 + 10} = 55 \text{ entries in the Root Node}$$

Hmmm. This means that we can't use a 2048-byte page size to get a two-level tree. Let's try a 4096-byte page:

$$(3B) \quad \frac{4096 - 42}{26 + 30 + 14} = 58 \text{ entries in each level zero node}$$

$$5000 / 58 = 87 \text{ nodes at level zero}$$

$$(1) \quad \frac{4096 - 42}{26 + 10} = 112 \text{ entries in the Root Node}$$

Ah-ha! Now we're in business because there are more entries in the Root Node than nodes at level zero.

Selector Subindexes

Often you'll find it convenient to use a *selector* subindex in your index structure. A selector subindex has just a few entries and the purpose of these entries is, logically enough, to select some large portion of the total index structure.

To illustrate the convenience of a selector subindex, let's first look at the database, in Figure B-4, which has two unique index structures.

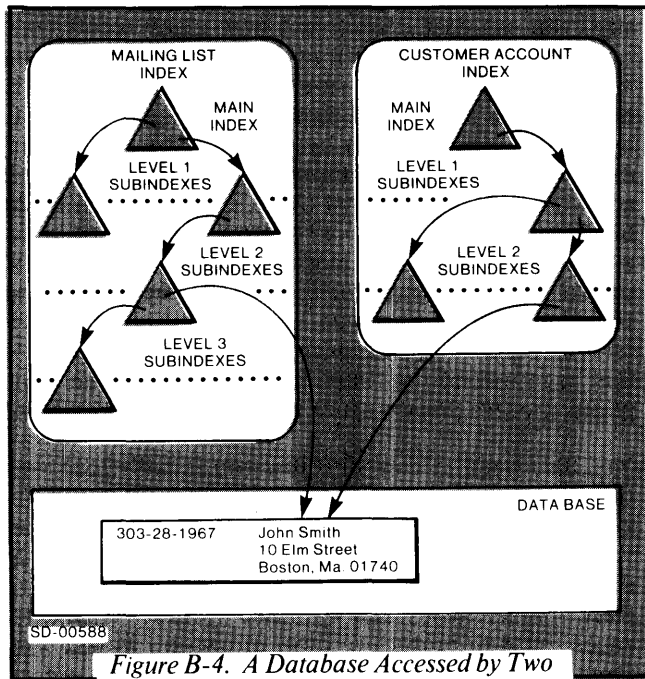


Figure B-4. A Database Accessed by Two Unique Indexes

Remember that when you open each index, the system allocates a set of I/O buffers in your User Area, as well as a complete set of control blocks in the system's File Control Area. A selector subindex, however, gives you all the versatility of the two unique indexes, but saves you File Control Space, and reduces the number of buffers you need. Thus you can probably use a larger, more efficient page size for your index. Figure B-5 shows what happens when we combine our two unique indexes into one. Note that the original main indexes have become level one subindexes under the new selector subindex.

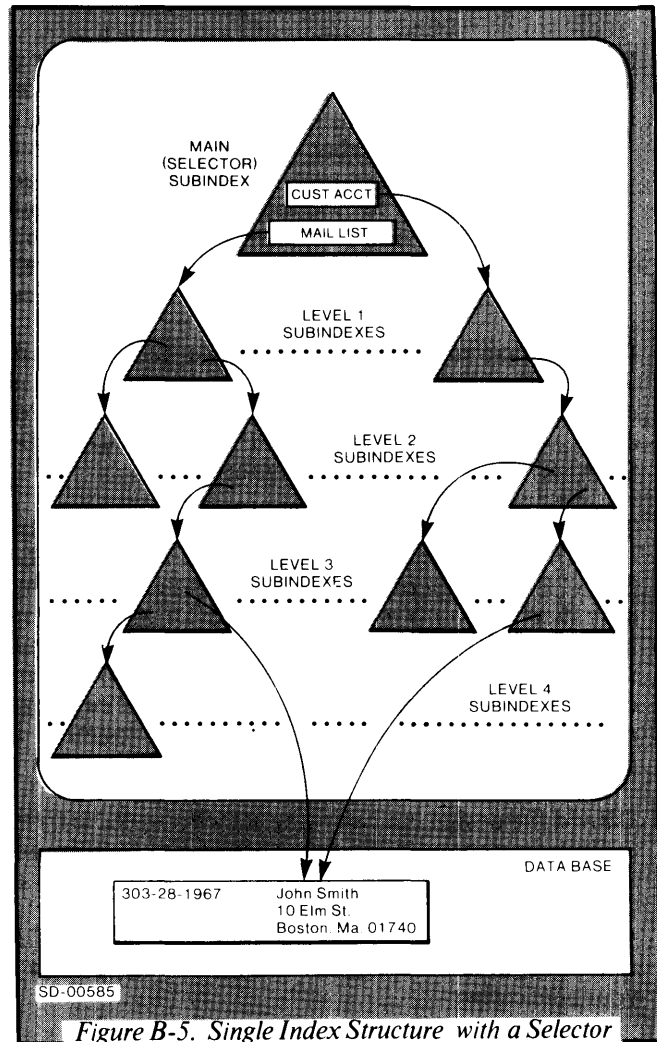


Figure B-5. Single Index Structure with a Selector Subindex

In general, a selector subindex will only consist of a Root Node, and you can find its maximum size by using formula 2A, above. For this application, let's say that our page size is 2048 bytes and our keys are 9 bytes long. We won't use partial records here, since we just want to use the entries in this subindex to select one of the two major processing paths. Thus:

$$(2A) \quad \frac{\text{Page Size} - 42}{\text{Max Key Length} + \text{Partial Record Length} + 14}$$

$$\frac{2048 - 42}{9 + 0 + 14} = \frac{2006}{24} = 83 \text{ (maximum) entries in the Root Node}$$

NOTE: The sum of 9 + 14 is 23, but we rounded it up to 24 because index entries begin on word boundaries, not on byte boundaries. Also, if you have a number of selector subindexes in your index structure, and if you select Space Management for the index, the INFOS system will automatically put as many of those small subindexes as it can on a single page, thereby making the most out of your disk storage space.

Finally, make sure that the initial node size you specify when you define *any* subindex is large enough to contain at least three of your index entries. If you don't, you'll get the message "MIN NODE SIZE TOO SMALL", and you'll have to respecify it.

To sum up subindex design, first note that the INDEXCALC utility (described in the *INFOS Utilities Users' Manual*) performs all the operations we've just described for you. You simply give INDEXCALC your maximum key size and the partial record length, and tell it whether or not the subindexes you're considering can have subindexes. You can then solve for page size, number of subindex tree levels, or the number or entries the subindex can hold.

DBAM file design is a balancing act where you're trying to find the right page size, number of I/O buffers, and index depth to best match your chosen access path (i.e., keyed, relative, or combined). By using INDEXCALC (or by doing the calculations yourself), you can fine tune the system to perform as efficiently as possible in your application.

Database Files

Database files also use pages as the unit of transfer between the database and its I/O buffers. But, since the system always transfers whole disk blocks, you should specify your page size as some exact multiple of 512 bytes to avoid wasting buffer space. In addition, database files use variable length records, so you have to specify a maximum record size, as well as the page size, when you create your file.

Once you know your maximum record size and page size, you can easily figure out how many records you can get on a page (known as the *blocking factor*), and how much space you'll have left. (Before you do this, however, be aware that there is an overhead of four bytes per record and four bytes per database page.) For example, if your maximum record length is 275 bytes and your page size is 1024 bytes, then your minimum blocking factor is 3, with 183 unused bytes per page; for a 2048-byte page, the blocking factor is 7, with 92 unused bytes per page, or 13 bytes per record wasted. In other words, you can only get seven 275-byte records on a 2048-byte page; the remaining space on that page will not be filled. In general, the larger your pages, the less unused space you'll have in the buffers.

You also need to consider how you initially load your file onto your disk(s). Normally you'll want to sort the keys for a given subindex from lowest to highest before you load them because the INFOS system can handle them most efficiently that way. That is, the system will place index entries and database records in adjacent pages (in their respective files, of course) as you write them. Therefore, if you subsequently want to process these records sequentially, the database blocking factor will determine how many times the system has to access the disk for read and rewrite operations. If you load your records in order, the system won't have to keep moving blocks of records around in the buffers and you'll save a lot of access time. Conversely, if you process the data records associated with a subindex via normal keyed accesses, ordered loading will greatly reduce the number of times the system has to access the database for each read or rewrite.

End of Appendix

Appendix C

The INFOS/FORTRAN Interface

NOTE: If you're going to program your INFOS application in FORTRAN IV or 5, read Chapter 6 in Part Two before reading this Appendix. This will acquaint you with the parameters within the various packets and how they fit together, and will clarify the descriptions which follow.

General Information

The INFOS/FORTRAN interface is a set of library routines which apply to FORTRAN IV (F4INFOS.LB) and FORTRAN 5 (F5INFOS.LB). These routines allow you to place word or byte addresses into parameter packet locations and to make INFOS system calls. However, the routines do not provide all the parameters which the various functions require, so you must provide these 'missing' parameters in assignment or data-initialization statements.

We have grouped the routines in this appendix into four types to help you understand and remember them. These 'types', however, have no meaning to either FORTRAN or to the INFOS system.

Packet Building Routines

These routines do not interact with the INFOS system; therefore, the system cannot detect specification errors within them. Consequently, you will find no error arguments in the calls to the following routines:

- INFFDP To build SAM and RAM File Definition Packets
- INFIFDP To build ISAM and DBAM File Definition Packets
- INFVDP To build Volume Definition Packets
- INFVIP To build Volume Initialization Packets for use with SAM labeled tape files
- INFKEY To build ISAM and DBAM Key Definition Packets
- INFLSP To build a DBAM Link Subindex Processing Packet

Table Building Routine

There is only one table building routine: INFULT. You will use this routine to build a User Label Table for use with SAM labeled tape files. This routine uses a variable number of arguments, and, similar to the Packet Building routines, does not interact with the system; therefore, no error argument is provided.

Pure Processing Routines

Unlike the above routines, pure processing routines *do* interact with the system; therefore, you will find error arguments in the lists of arguments for these calls. These error arguments enable the system to notify you if something goes awry. The three pure processing routines are:

INFPREP To perform Pre-open processing

INFOPEN To perform Open processing

INFINIT To perform volume initialization for SAM labeled tape files

NOTE: These routines each have a single, fixed-sequence call with no optional arguments.

Building/Processing Routines

There are two building/processing routines:

INFOS To perform SAM or RAM file processing

INFOX To perform ISAM or DBAM file processing

These are similar to the packet building routines in that they help you fill in packet slots, and to the pure processing routines in that they interact with the system and contain error arguments in the calls.

Arguments

The calling sequences of both the packet building and the building/processing routines contain optional arguments. In the descriptions of the calls which follow, we have enclosed optional arguments within brackets like this: *[arg1]*. In general, you can omit all the optional arguments in a calling sequence if they do not apply, but note that if you use any *one* of the arguments within brackets, you must include *all* of them. To pass any optional arguments which you don't want in a particular call, specify NIL, DEFAULT, or DEF for that argument.

For example, suppose the arguments for a call appear in this sequence:

```
(arg1, arg2 [,arg3, arg4, arg5, arg6] [,arg7]).
```

If you want to use only *arg3* and *arg6*, you can write (ARG1, ARG2, ARG3, NIL, NIL, ARG6).

You would simply omit the last argument (*arg7*) if you didn't want to use it.

Licensed Material - Property of Data General Corporation

NOTE: NIL, DEFAULT, and DEF have special meanings. See the paragraphs which follow for their descriptions.

Finally, sometimes arguments for a call will appear in this sequence:

```
(arg1, arg2 [,arg3 [, arg4 [, arg5]]])
```

To use one or more of the inner, optional arguments, you must include all of the outer ones. That is, if you want *arg5*, then you must also include *arg3* and *arg4*; if you want *arg4*, you must include *arg3*, but you can omit *arg5*.

Using DEFAULT (or DEF)

There are many instances where you can pass DEFAULT or DEF for an argument. However, before you use them, you must declare either (or both) EXTERNAL.

Defaulting an argument instructs the interface routines to place -1 in the corresponding packet slot, which, in turn, causes the system to use one of these values:

- The system's default value; or
- A default value which you have previously defined; or
- No value. (This is used only when the argument has no meaning to the access method.)

In the call descriptions which follow, we will tell you when you may or may not default an argument.

Using NIL

Passing NIL for an argument is different from defaulting an argument. When you specify NIL for an argument, the interface routines do not alter the parameter slot corresponding to that argument. Thus, the INFOS system will use any previously-set parameter value for that parameter. Therefore, you can pass NIL for a required argument if the packet involved already contains a parameter value acceptable to both you and the INFOS system.

You may also use NIL as a "place-holder" in a group of optional arguments, all of which must be present, but only some of which you need to specify: for example, (ARG1, ARG2, ARG3, NIL, NIL, ARG6) in the example above.

Finally, similar to DEFAULT, you must declare NIL as EXTERNAL before you use it.

Error Conditions

The interface routines which interact with the INFOS system all contain as their last, mandatory argument, an integer variable which we call the "error argument". When the routine terminates, it places an integer code in this argument to inform the calling program of the call's status. This status code conforms to the standard FORTRAN IV and FORTRAN 5 conventions described in the *FORTRAN IV* and *FORTRAN 5 User's Manuals* (manual numbers 093-000053 and 093-000085, respectively). Briefly, this code means:

- ≤ 0 (not used)
- 1 means no errors have occurred
- 2 (not used)
- ≥ 3 an error has occurred

You should always check the returned status code before proceeding. Even if your program does not include special code to recover from particular errors, you should at least verify that the returned status code is 1. The system error messages exist for your benefit; do not ignore them. (Refer to Appendix D for explanations of all INFOS error codes.)

An example of a typical application might look like this:

```

      :
      :
CALL INFINIT (VIP,IER)
IF (IER.NE.1) GO TO 999
      :

```

(where 999 is the label of a statement which deals with errors from this call.)

Using FINFOS.FR

The tape set which contains the FORTRAN library routines also includes a "parameter tape" for the INFOS/FORTRAN interface called FINFOS.FR. This is a source file which defines the mnemonic names for the many parameters you need when programming an INFOS application in FORTRAN. We strongly urge you to use mnemonics; your programs will be much easier for both you and others to understand. We have included a copy of the current (as of the printing of this manual) FINFOS.FR at the end of this appendix. Use it to look up the parameters you want for your application; the call descriptions on the following pages will show you how to set those parameters.

FINFOS.FR is structured as a FORTRAN source file so that the FORTRAN compiler will accept it as a part of your FORTRAN program. This allows you to write your FORTRAN program using mnemonic names. Therefore, do one of the following before compiling your program:

1. Attach FINFOS.FR to the beginning of your FORTRAN source file; or
2. (FORTRAN 5 only) Write an INCLUDE "FINFOS.FR" statement near the beginning of your program. (You may also wish to compile your program using "FORTRAN/I"; this will shorten your listing considerably.)
3. Insert selected portions of FINFOS.FR into your FORTRAN source file. However, if you choose this method, be careful how you edit FINFOS.FR. Remember that the pieces you extract must be proper FORTRAN statements.

Loading Your Program and the Interface Routines

Once you have compiled all your FORTRAN subprograms, you can load your program according to the procedures described in either Appendix D of the *FORTRAN IV USER's Manual* (number 093-000053) or Chapter 2 of the *FORTRAN 5 Supplement* (number 093-000185). Then you can name the appropriate interface library (F4INFOS.LB or F5INFOS.LB) in your RLDR command line between your FORTRAN subprograms and the standard FORTRAN runtime libraries.

Call Formats

In the following call descriptions, we have indicated, where possible, specific data types for arguments. However, certain arguments are shown as "aggregates". These can be:

- An array containing an entity (e.g., a data item, packet address, word address, array address).
- An array element designating the beginning of an entity stored in a portion of an array.
- A Hollerith constant large enough to contain an entity.
- A scalar large enough to contain an entity.
- An EXTERNAL defined in Assembly Language.

Packet Building Routines

INFFDP

To build SAM and RAM File Definition Packets

Call Format:

CALL INFFDP(fdp, vdt [,itt, ott, dsdt, ftt] [,fsid])

Arguments:

fdp	File Definition Packet aggregate (i.e., a variable which is the first word of the FDP). Must be passed.
vdt	Volume Definition Table aggregate made up of one or more Volume Definition Packets which must be located sequentially. Can be passed as NIL. Be sure you indicate the number of VDP's in word FDNVD of the FDP.
itt	User Input Translation Table aggregate; only needed if you want to translate from ASCII to something other than EBCDIC, or from EBCDIC to something other than ASCII. Can be passed as NIL.
ott	User Output Translation Table aggregate; only needed if your translation is not ASCII to or from EBCDIC. Can be passed as NIL.
dsdt	Data Sensitive Delimiter Table aggregate. If you are not using Data Sensitive records, pass NIL for this argument. If your records are delimited by a carriage return, form feed or null, pass DEF or DEFAULT. If your records are delimited by any other character(s), you must build your own table, according to the CMT instruction described in the <i>Programmer's Reference Manual, ECLIPSE-Line Computers (015-024)</i> .
ftt	Selective Field Translation Table aggregate. (See Comment 3 under Table II-6-1, earlier in this manual.) Can be passed as NIL.
fsid	File Set Identifier aggregate (applies to labeled mag tapes only). Can be passed as NIL, DEF, or DEFAULT.

Licensed Material - Property of Data General Corporation

Example:

- (1) INTEGER FDP (FDLN1)
- (2) INTEGER VDT (VDLEN,4)
- (3) INTEGER ITRAN (128), OTRAN (128)
- (4) CALL INFFDP (FDP, VDT, ITRAN, OTRAN, DEF, NIL, "MINE")
- (5) FDP (FDL1) = F1UBR+F1SAM+F1EXF
- (6) FDP (FDBLK) = 512
- (7) FDP (FDBUF) = 2
- (8) FDP (FDLEN) = 512
- (9) FDP (FDNVD) = 4

- Line (1) allocates a File Definition Packet for a SAM labeled tape file.
- Line (2) allocates a four-volume file. Note that you must build a Volume Definition Table with an INFVDP call (described later in this appendix).
- Line (3) allocates Input and Output Translation Tables (as described in Chapter 1 of Part One). You can define these tables in Assembly language or as DATA-Initialized FORTRAN arrays.
- Line (4) calls the routine. The INFOS/FORTRAN interface will fill the File Definition Packet with pointers to the Volume Definition Table, the Input and Output Translation tables, and with the aggregate name of the File Set. The Data Sensitive Delimiter Table aggregate is defaulted since a labeled SAM file cannot use Data Sensitive records. (No value is used in this case.) The Selective Field Translation Table slot in the parameter packet remains untouched since its corresponding argument was passed as NIL.
- Line (5) specifies unblocked records, the Sequential Access method, and an Exclusive file. The FORTRAN interface will communicate this information to the INFOS system and all specifications will take effect when opening occurs.
- Line (6) specifies a block size of 512 bytes.
- Line (7) specifies two I/O buffers.
- Line (8) specifies a record length of 512 bytes.
- Line (9) specifies that there are four Volume Definition Packets in the Volume Definition Table.

Licensed Material - Property of Data General Corporation

INFIFDP

To build ISAM and DBAM File Definition Packets.

Call Format:

CALL INFIFDP(*fdp*, *vdt* [, *dbfdp* [, *dbn*]])

Arguments:

- fdp* File Definition Packet aggregate (i.e., a variable which is the first word of the FDP). Must be passed.
- vdt* Volume Definition Packet aggregate made up of one or more VDP's located sequentially. Can be passed as NIL. Be sure you indicate the number of VDP's in word FDNVD of the FDP.
- dbfdp* Database File Definition Packet aggregate. Can be passed as NIL, DEF, or DEFAULT.
- NOTE: You must notify the system that a database FDP is present by setting bit F2FDP in word FDFL2 of the index FDP.
- dbn* Database Name aggregate. Can be passed as NIL, DEF, or DEFAULT.

NOTE: The *dbfdp* and *dbn* arguments both deal with the FDDBP slot in the File Definition Packet. If you want to specify a Database FDP, pass such an aggregate for *dbfdp* and either omit *dbn* or pass NIL, DEF, or DEFAULT in its place. If you want to specify a Database name, pass NIL, DEF, or DEFAULT for *dbfdp* and pass your Database name aggregate for *dbn*. If you do not wish to alter the FDDBP slot, simply omit the *dbfdp* and *dbn* arguments.

Example:

- (1) INTEGER FDP (FDLN2)
- (2) INTEGER DBFDP (FDLN1)
- (3) INTEGER VDT (VDLEN, 2)
- (4) CALL INFIFDP(FDP, VDT, DBFDP, NIL)
- (5) FDP (FDLEN) = 50
- (6) FDP (FDPRL) = 10
- (7) FDP (FDFL1) = F1INV
- (8) FDP (FDFL2) = F2ORD + F2FDP + F2DHR
- (9) FDP (FDIFL) = IXPKC + ICTMP
- (10) DBFDP (FDBUF) = 2

- Line (1) allocates an Index File Definition Packet.
- Line (2) allocates a Database FDP.
- Line (3) allocates a two-volume file. You must build a Volume Definition Table for each with a call to INFVDP (described on the next page).
- Line (4) calls the routine. This fills the FDVTP slot with a pointer to the Volume Definition Table and the FDDBP slot with a pointer to the database FDP aggregate.
- Line (5) specifies that the maximum record length is 50 bytes.
- Line (6) specifies a partial record length of 10 bytes.
- Line (7) specifies that the Database will be inverted.
- Line (8) specifies optimized record distribution for the Index and Disable Hierarchical replacement (i.e., LRU technique).
- Line (9) specifies key compression and that the Index is temporary.
- Line (10) specifies a runtime specification of two database buffers.

INFVDP

To build Volume Definition Packets

Call Format:

CALL INFVDP (vdp, fnam [, vlt, hlt, tlt] [, valid])

Arguments:

- vdp* Volume Definition Packet aggregate made up of one or more VDP's located sequentially. Must be passed.
- fnam* Volume Name aggregate. Can be passed as NIL.
- vlt* Volume Label Table aggregate; applies only to labeled magnetic tape files. Can be passed as NIL, DEF, or DEFAULT.
- hlt* Header Label Table aggregate; applies only to labeled magnetic tape files. Can be passed as NIL, DEF, or DEFAULT.
- tlt* Trailer Label Table aggregate; applies only to labeled magnetic tape files. Can be passed as NIL, DEF, or DEFAULT.
- valid* Volume owner identifier aggregate; applies only to labeled magnetic tape files. Can be passed as NIL, DEF, or DEFAULT.

Example:

- (1) PARAMETER ASTER = 52K
- (2) INTEGER VDT (VDLEN,2)
- (3) CALL INFVDP (VDT(1,1), NIL, DEF, NIL, DEF, "SMITH")
- (4) CALL INFVDP (VDT(1,2), NIL, DEF, DEF, DEF, "SMITH")
- (5) VDT (VDPAD,1) = ASTER
- (6) VDT (VDPAD,2) = ASTER

- Line (1) defines the character Asterisk. (See lines 6 and 7).
- Line (2) allocates a two-volume Volume Definition Table.
- Line (3) calls the routine. This sets up the first Volume Definition Packet in the Volume Definition Table.
- Line (4) calls the routine. This sets up the second Volume Definition Packet in the Volume Definition Table.
- Lines (5-6) specify that the padding character for the two volumes is the asterisk.

INFVIP

To build Volume Initialization Packets for use with labeled SAM files.

Call Format:

CALL INFVIP (vip, dnam [, vid[, void[, ult]]])

Arguments:

- vip* Volume Initialization Packet aggregate. Must be passed.
- dnam* Device name aggregate. Can be passed as NIL.
- vid* Volume Identifier aggregate. Can be passed as NIL, DEF, or DEFAULT.
- void* Volume Owner Identifier aggregate. Can be passed as NIL, DEF, or DEFAULT.
- ult* User Volume Label Table aggregate. Can be passed as NIL, DEF, or DEFAULT.

Example:

- (1) INTEGER VIP(VILEN)
- (2) CALL INFVIP (VIP, "MT2", "VOL4", "SMITH", NIL)

- Line (1) allocates a Volume Initialization Packet.
- Line (2) calls the routine and fills VDVNP with a pointer to volume 4, and VDOID with a pointer to "SMITH".

INFKEY

To build ISAM and DBAM Key Definition Packets

Call Format:

CALL INFKEY (kdp, key [, len])

Arguments:

kdp Key Definition Packet aggregate. Must be passed.

key Key value aggregate. Must be passed.

len Integer value giving flagged key length. Can be passed as NIL, DEF, or DEFAULT.

If you pass a key length other than NIL, DEF, or DEFAULT, the INFKEY routine stores it in the right-hand byte of KDKYL. If you do not pass a key length (or if you pass it as DEF or DEFAULT), INFKEY computes the length of the key as the number of nonnull bytes in the string and stores the computed length in the right-hand byte of KDKYL. (Note that null is a delimiting character only when INFKEY computes key length.) If you pass key length as NIL, the right-hand byte of KDKYL is not changed.

NOTES:

1. The INFKEY routine never alters the flag bits in the left-hand byte of KDKYL. You should set up these flags before calling INFKEY.
2. INFKEY sets up only one level of a key table at a time. It does *not* set up the null word which must follow the key packet, nor does it set up KDDKO (Duplicate Key occurrence).

Example:

The three following examples all produce the same result. The only difference among them is the way that the key length is specified.

A)

- (1) INTEGER KTAB (KDLEN,4)
- (2) KTAB (KDTYP,1) = KTDUP
- (3) CALL INFKEY (KTAB(1,2), "SMITH")
- (4) CALL INFKEY (KTAB (1,1), "NAME")

B)

- (1) INTEGER KTAB (KDLEN,4)
- (2) KTAB (KDTYP,2) = KTDUP
- (3) CALL INFKEY (KTAB (1,2), "SMITH", 5)
- (4) CALL INFKEY (KTAB(1,1), "NAME")

C)

- (1) INTEGER KTAB (KDLEN,4)
- (2) KTAB (KDTYP,2) = KTDUP+5
- (3) CALL INFKEY (KTAB (1,2), "SMITH", NIL)
- (4) CALL INFKEY (KTAB(1,1), "NAME")

Line (1) allocates a Key Definition Packet.

Line (2) specifies duplicate keys (and, for example C, indicates the key length).

Line (3) calls the routine, fills the PRKTP slot in the Extended Processing Packet with a pointer to the key table, and fills KDKYP with a pointer to "SMITH", which is the second key in a table which has room for up to three keys.

Line (4) calls the routine to set up KDKYP with a pointer to "NAME," which is the first key in the key table.

Since INFKEY does not write the null word which must follow a Key Table to terminate it, we recommend that you specify a zero length for the terminating entry in the table. In our example, this line would look like this:

- (5) KTAB(KDKYL, 3) = 0

This will make sure that the system reads the end of the Key Table properly; otherwise, the results would be unpredictable.

INFLSP

To build a DBAM Link Subindex Processing Packet.

Call Format:

CALL INFLSP (lsp, skt, dkt)

Arguments:

- lsp** Link Subindex processing packet aggregate; must be passed. (See Figure II-6-9 in this manual.)
- skt** Source Key Table aggregate. Can be passed as NIL. (See Figure II-6-9.)
- dkt** Destination Key Table aggregate. Can be passed as NIL.

NOTE: You must use this call in conjunction with an INFOX call as follows:

- (1) CALL INFLSP (lsp, skt, dkt)
- (2) CALL INFOX (chan, LNKSI, lsp, nil, nil, nil, ier)

where the arguments for INFOX are:

- chan** Integer variable containing the file's pseudo-channel number, as returned from INFOPEN. Must be passed.
- LNKSI** Integer value specifying the Link Subindex processing function. Must be passed.
- ier** Integer variable error return. Must be passed.

Line (1) calls the Link Subindex packet building routine.

Line (2) calls the DBAM processing routine. The interface routines will set up pointers in the processing packet, call INFOS, and return a status code to the program.

Table Building Routine

INFULT

To build a User Label Table for use with SAM labeled tape files.

Call Format:

CALL INFULT (ult, lab1 [,lab2 [, ...[,labn]]])

Arguments:

- ult** User Label Table aggregate. Must be passed.
- lab1** Aggregates giving any user labels. Each can be passed as NIL, DEF, or DEFAULT.
- labn**

You can give any label as NIL, and the INFULT routine will not touch the corresponding entry in an existing table. Also, you can DEFAULT any label and INFULT will place -1 in the corresponding entry.

If you're using the routine to modify the contents of an existing table, the number of entries in the modified table will be equal to the number of filenames passed as aggregates or as NIL, DEF, or DEFAULT when the routine has finished.

Example:

```
CALL INFULT (VLT, "USERNO")  
CALL INFULT (HLT, "START")  
CALL INFULT (TLT, "END")
```

Pure Processing Routines

INFPREP

To perform Pre-open file processing.

Call Format:

CALL INFPREP (fdp, ier)

Arguments:

fdp File Definition Packet aggregate (i.e., a variable which is the first word of the FDP). Must be passed.

ier Integer variable error argument. Must be passed.

Examples:

To do Pre-open processing of a SAM or RAM file:

```
(1) INTEGER FDP (FDLN1)
(2) CALL INFPREP (FDP, IER)
(3) IF (IER.NE.1) GO TO 999
```

To do Pre-open processing of an ISAM or DBAM file:

```
(1) INTEGER FDP (FDLN2)
(2) CALL INFPREP (FDP, IER)
(3) IF (IER.NE.1) GO TO 999
```

Line (1) allocates a File Definition Packet.

Line (2) calls the routine

Line (3) sends control to an error recovery routine if an error has occurred.

INFOPEN

To perform Open file processing

Call Format:

CALL INFOPEN (chan, fdp, ier)

Arguments:

chan Integer variable for return of file's pseudo-channel number. Must be passed.

NOTE: Save this channel number. You will need it for all subsequent calls to the INFOS system.

fdp File Definition Packet aggregate. Must be passed.

ier Integer variable error argument. Must be passed.

Examples:

To perform Open processing of a SAM or RAM file:

```
(1) INTEGER FDP (FDLN1)
(2) CALL INFOPEN (NC, FDP, IER)
(3) IF (IER.NE.1) GO TO 999
```

To perform Open processing of an ISAM or DBAM file:

```
(1) INTEGER FDP (FDLN2)
(2) CALL INFOPEN (NC, FDP, IER)
(3) IF (IER.NE.1) GO TO 999
```

Line (1) allocates a File Definition Packet.

Line (2) calls the routine and sets up the channel number.

Line (3) sends control to an error recovery routine if an error has occurred.

INFINIT

To perform volume initialization for SAM labeled tape files

Call Format:

CALL INFINIT (vip, ier)

Arguments:

- vip Volume Initialization Packet aggregate, as described in Table II-6-10. Must be passed.
- ier Integer variable error return. Must be passed.

Example:

```
CALL INFINIT (VIP, IER)  
IF (IER.NE.1) GO TO 999
```

This calls the routine and supplies the address of the Volume Initialization Packet.

Building/Processing Routines

INFOS

To perform SAM or RAM file processing

Call Format:

CALL INFOS (chan, funct, pp [,dat],ier)

Arguments:

- chan A variable which is resolvable as an integer, to contain the file's pseudo-channel number, as returned by INFOPEN. Must be passed.
- funct A value (resolvable as an integer), which specifies a SAM or RAM processing or utility function. Must be passed. (You will find the mnemonics for each processing function under "Processing Function Codes" in FINFOS.FR.)
- pp Processing Packet aggregate. Must be passed.
- dat Data Area aggregate giving area to or from which transfer will be made. Can be passed as NIL.
- ier Integer variable error argument. Must be passed.

Example:

```
(1) INTEGER PP(PRLN1)  
(2) INTEGER AREA (25)  
(3) PP(PRSTA) = PFLOC  
(4) PP(PRREC) = 0  
(5) PP(PRREC + 1) = N  
(6) PP(PRLEN) = 50  
(7) CALL INFOS (NC, IREAD, PP, AREA, IER)  
(8) IF (IER.NE.1) GO TO 999
```

- Line (1) allocates a SAM or RAM processing packet.
- Line (2) allocates a 50-byte record processing area.
- Line (3) specifies locked record.
- Lines (4-5) set up a two-word record number whose value is Integer N.
- Line (6) specifies a 50-byte record length.
- Line (7) calls the routine.
- Line (8) sends control to an error recovery routine if an error has occurred.

INFOX

To perform ISAM or DBAM file processing

Call Format:

CALL INFOX (chan, funct, pp, ktab, dat, pra [*,sip*], ier)

Arguments:

chan	Integer variable containing the file's pseudo-channel number, as returned by INFOPEN. Must be passed.
funct	A value (resolvable as an integer), which specifies an ISAM or DBAM processing or utility function. Must be passed. (For the mnemonics you should use, see the section on "Processing Function Codes" in FINFOS.FR.)
pp	Processing Packet aggregate (i.e., a variable which is the first word of the packet). Must be passed.
ktab	Key Table aggregate. Can be passed as NIL.
dat	Data Area aggregate giving the area to or from which transfer will be made. Can be passed as NIL.
pra	Partial Record area aggregate. Can be passed as NIL.
sip	Subindex Packet aggregate. Can be passed as NIL.
ier	Integer variable error argument. Must be passed.

Example:

```
(1) INTEGER XPP(PRLN2)
(2) INTEGER BUF(80)
(3) INTEGER PBUF(5)
(4) INTEGER KTAB(KDLEN,3)
(5) KTAB(KDTYP,1) = 0
(6) CALL INFKEY (KTAB(1,1), "AB")
(7) KTAB (KDTYP,2) = 0
(8) CALL INFKEY (KTAB(1,2), "CD", 2)
(9) KTAB(1,3) = 0
(10) XPP(PRCCW) = CCKEY
(11) XPP(PRLN) = 160
(12) CALL INFOX (NC, IREAD, XPP, KTAB, BUF, PBUF,
IER)
(13) IF(IER.NE.1) GO TO 999
```

Line (1) allocates a DBAM Processing Packet.

Line (2) allocates a 160-byte data area.

Line (3) allocates a 10-byte partial record area.

Line (4) allocates a Key Table aggregate for two-level keys. Note that the Key Table is terminated by an entry specifying a key with zero length.

Line (5-9) set up the Key Table aggregates.

Line (10) specifies keyed access.

Line (11) specifies a record length of 160 bytes

Line (12) calls the routine. First, the INFOX routine sets up pointers in the processing packet, then calls the INFOS system to transfer the record and the partial record to the appropriate areas. Finally, it returns status in IER.

Line (13) sends control to an error recovery routine if an error has occurred.

INFOS User Parameters for FORTRAN

```

C=====
C  INFOS USER PARAMETERS FOR FORTRAN
C=====

C      FINFOS.FR

C      FILE DEFINITION PACKET

      PARAMETER
+  FDFL1  =  4      ;;INFOS -FILE DEF FLAGS, I
+  FDFL2  =  5      ;;INFOS -FILE DEF FLAGS, II
+  FDBLK  =  6      ;;INFOS -BLOCK SIZE
+  FDBUF  =  7      ;;INFOS -NUMBER OF BUFFERS
+  FDLEN  =  8      ;;INFOS -RECORD LENGTH
+  FDNIL  =  8      ;;INFOS -NUMBER OF INDEX LEVELS
+  FDNVD  =  11     ;;INFOS -NUM OF VOL TABLE ENTRIES
+  FDVTP  =  12     ;;INFOS -VOLUME TABLE POINTER
+  FDUIT  =  14     ;;INFOS -USER INPUT TRANS TBL PTR
+  FDUOT  =  16     ;;INFOS -USER OUTPUT TRANS TBL PTR
+  FDTCF  =  18     ;;INFOS -TRANS & LABEL FLAGS
+  FODSD  =  19     ;;INFOS -DATA SENS DELIM TABLE PTR
+  FDFSI  =  21     ;;INFOS -FILE SET ID PTR
+  FDEXP  =  24     ;;INFOS -EXPIRATION DATE
+  FDSEQ  =  27     ;;INFOS -SEQUENCE NUMBER
+  FDGEN  =  28     ;;INFOS -GENERATION NUMBER
+  FDACC  =  29     ;;INFOS -FILE ACCESSABILITY
+  FDIDO  =  30     ;;INFOS -INITIAL DATA OFFSET
+  FDSFT  =  31     ;;INFOS -SEL FIELD TRANS TABLE PTR
+  FDLN1  =  40     ;;INFOS -SAM & RAM FDP LENGTH
+
+  FDEFT  =  41     ;;INFOS -EXCLUDED FILE TABLE PTR
+  FDDBP  =  43     ;;INFOS -DATA BASE FILE DEF PACKET
+                ;;INFOS -OR NAME POINTER
+  FDMNS  =  45     ;;INFOS -MINIMUM NODE SIZE
+  FDMKL  =  47     ;;INFOS -MAX KEY LENGTH (LH BYTE)
+  FDPRL  =  47     ;;INFOS -PART REC LEN (RH BYTE)
+  FDRMF  =  49     ;;INFOS -RT ND MERIT FACTOR
+  FDIFL  =  50     ;;INFOS -INDEX FLAGS
+  FDLN2  =  50     ;;INFOS -ISAM & DBAM FDP LENGTH

C      SUBINDEX DEFINITION PACKET

      PARAMETER
+  SDMNS  =  1      ;;INFOS -MINIMUM NODE SIZE
+  SDMKL  =  3      ;;INFOS -MAX KEY LEN (LH BYTE)
+  SDPRL  =  3      ;;INFOS -PART REC LEN (RH BYTE)
+  SDRMF  =  5      ;;INFOS -RT ND MRT FACT (RH BYTE)
+  SDIFL  =  6      ;;INFOS -INDEX FLAGS
+  SDLEN  =  6      ;;INFOS -SUBINDEX DEF PACKET LENGTH

```


INFOS User Parameters for FORTRAN (continued)

C FILE DEFINITION FLAGS, I (FDFL1)

PARAMETER			
+	F1UBR	= 100000K	;;INFOS -UNBLOCKED RECORDS
+	F1AM1	= 40000K	;;INFOS -ACCESS METHOD FIELD
+	F1AM2	= 20000K	;;INFOS -
+	F1FT1	= 10000K	;;INFOS -RECORD FORMAT FIELD
+	F1FT2	= 4000K	;;INFOS -
+	F1RAW	= 2000K	;;INFOS -READ AFTER WRITE VER
+	F1OVR	= 200K	;;INFOS -OVERWRITE (APPEND IF 0)
+	F1EXF	= 100K	;;INFOS -EXCLUSIVE FILE
+	F1PM1	= 20K	;;INFOS -PROCESSING MODE FIELD
+	F1PM2	= 10K	;;INFOS -
+	F1RER	= 4	;;INFOS -REWRITE (NORMAL IF 0)
+	F1INV	= 1	;;INFOS -INVERTING (ISAM)

C ACCESS METHOD SPECIFIERS (F1AM1,F1AM2)

PARAMETER			
+	F1AMM	= 60000K	;;INFOS -FIELD MASK
+	F1SAM	= 20000K	;;INFOS -SAM
+	F1RAM	= 0	;;INFOS -RAM
+	F1ISM	= 40000K	;;INFOS -ISAM
+	F1DBM	= 40000K	;;INFOS -DBAM

C RECORD FORMAT SPECIFIERS (F1FT1,F1FT2)

PARAMETER			
+	F1FTM	= 14000K	;;INFOS -FIELD MASK
+	F1FIX	= 10000K	;;INFOS -FIXED LENGTH
+	F1VAR	= 4000K	;;INFOS -VARIABLE LENGTH
+	F1UND	= 0	;;INFOS -UNDEFINED LENGTH
+	F1SEN	= 14000K	;;INFOS -DATA SENSITIVE

C PROCESSING MODE SPECIFIERS (F1PM1,F1PM2)

PARAMETER			
+	F1PMM	= 30K	;;INFOS -FIELD MASK
+	F1INP	= 0	;;INFOS -INPUT
+	F1OUT	= 20K	;;INFOS -OUTPUT
+	F1UPD	= 10K	;;INFOS -UPDATE
+	F1CRU	= 30K	;;INFOS -CREATE UPDATE

C FILE DEFINITION FLAGS, II (FDFL2)

PARAMETER			
+	F20TI	= 20000K	;;INFOS -OPEN ONLY THIS INDEX
+	F23PM	= 2000K	;;INFOS -SPACE MANAGEMENT
+	F20KD	= 100K	;;INFOS -OPTIMIZE REC DISTRIBUTION
+	F2DHR	= 40K	;;INFOS -DISABLE HIERARCHICAL REPLACEMENT
+	F2FDP	= 10K	;;INFOS -DATA BASE FDP PRESENT

INFOS User Parameters for FORTRAN (continued)

```

C      TRANSLATION AND LABEL CONTROL FLAGS      (FDTCF)

      PARAMETER
+     TCLT1  = 100000K      ;;INFOS -LABEL TYPE FIELD
+     TCLT2  =  40000K      ;;INFOS -
+     TCLT3  =  20000K      ;;INFOS -
+     TCSFT  =  10000K      ;;INFOS -SELECTIVE FIELD TRANS
+     TCOT1  =   4000K      ;;INFOS -OUTPUT TRANS FIELD
+     TCOT2  =   2000K      ;;INFOS -
+     TCOT3  =   1000K      ;;INFOS -
+     TCOT4  =    400K      ;;INFOS -
+     TCLL1  =   200K      ;;INFOS -LABEL LEVEL
+     TCLL2  =   100K      ;;INFOS -
+     TCLL3  =    40K      ;;INFOS -
+     TCIT1  =    10K      ;;INFOS -INPUT TRANS FIELD
+     TCIT2  =     4       ;;INFOS -
+     TCIT3  =     2       ;;INFOS -
+     TCIT4  =     1       ;;INFOS -

C      LABEL TYPE SPECIFIERS      (TCLT1-TCLT3)

      PARAMETER
+     TCLTM  = 100000K      ;;INFOS -FIELD MASK
+     TCANS  =     0       ;;INFOS -ANSI STANDARD
+     TCIBM  =  20000K      ;;INFOS -IBM STANDARD

C      LABEL LEVEL SPECIFIERS      (TCLL1-TCLL3)

      PARAMETER
+     TCLLM  =   340K      ;;INFOS -FIELD MASK
+     TCLV1  =    40K      ;;INFOS -LEVEL 1
+     TCLV2  =   100K      ;;INFOS -LEVEL 2
+     TCLV3  =   140K      ;;INFOS -LEVEL 3

C      OUTPUT TRANSLATION SPECIFIERS      (TCOT1-TCOT4)

      PARAMETER
+     TCOTM  =   7400K      ;;INFOS -FIELD MASK
+     TCNTO  =     0       ;;INFOS -NO TRANS ON OUTPUT
+     TCEAU  =   400K      ;;INFOS -EBCDIC TO ASCII
+     TCAEO  =  1000K      ;;INFOS -ASCII TO EBCDIC
+     TCUTO  =   7400K      ;;INFOS -USER TABLE

C      INPUT TRANSLATION SPECIFIERS      (TCIT1-TCIT4)

      PARAMETER
+     TCITM  =    17K      ;;INFOS -FIELD MASK
+     TCNTI  =     0       ;;INFOS -NO TRANS ON INPUT
+     TCEAI  =     1       ;;INFOS -EBCDIC TO ASCII
+     TCAEI  =     2       ;;INFOS -ASCII TO EBCDIC
+     TCUTI  =    17K      ;;INFOS -USER TABLE

C      INDEX FLAGS      (FDIFL)

      PARAMETER
+     IXPKC  = 100000K      ;;INFOS -PERFORM KEY COMPRESSION
+     IXNSI  =  40000K      ;;INFOS -NO SUBINDICES
+     IXHPN  =  20000K      ;;INFOS -HIGH PRIORITY NODE
+     IXTSI  =  10000K      ;;INFOS -TEMPORARY INDEX (PRIMARY OR SUB)
+     IXPRM  =   4000K      ;;INFOS -MAKE DATA RECORDS PERMANENT

```

INFOS User Parameters for FORTRAN (continued)

```

C      VOLUME INITIALIZATION PACKET

      PARAMETER
+   VIFLG  =  1      ;;INFOS -VOL INIT FLAGS
+   VIACC  =  2      ;;INFOS -VOL ACCESSABILITY (LH BYTE)
+   VIDVS  =  3      ;;INFOS -DEV SPECIFIER PTR
+   VIVID  =  5      ;;INFOS -VOL ID PTR
+   VIOID  =  7      ;;INFOS -OWNER ID PTR
+   VIUVT  =  9      ;;INFOS -USER VOL LABEL TAB PTR
+   VILEN  =  10     ;;INFOS -VOL INIT PACKET LENGTH
    
```

```

C      VOLUME INITIALIZATION FLAGS (VIFLG)

      PARAMETER
+   VFLT1  =  100000K ;;INFOS -LABEL TYPE FLAGS
+   VFLT2  =   40000K ;;INFOS -
+   VFLT3  =   20000K ;;INFOS -
+   VFLL1  =    200K  ;;INFOS -LABEL LEVEL FIELD
+   VFLL2  =   100K  ;;INFOS -
+   VFLL3  =    40K  ;;INFOS -
+   VFFUL  =     4    ;;INFOS -FULL INIT
+   VFIVI  =     2    ;;INFOS -IGNORE VOL ID
    
```

```

C      NOTE THAT THE LABEL TYPE & LABEL LEVEL SPECIFIERS
C      GIVEN ON THE PREVIOUS PAGE MAY ALSO BE USED FOR
C      THIS FLAG WORD
    
```

```

C      VOLUME DEFINITION PACKET

      PARAMETER
+   VDVNP  =  1      ;;INFOS -VOLUME NAME POINTER
+   VDVSZ  =  3      ;;INFOS -VOLUME SIZE
+   VDVLT  =  4      ;;INFOS -VOLUME LABEL TABLE PTR
+   VDHLT  =  6      ;;INFOS -HEADER LABEL TABLE PTR
+   VDTLT  =  8      ;;INFOS -TRAILER LABEL TABLE PTR
+   VDPDC  =  10     ;;INFOS -PHYS DEV CHARACTERISTICS
+   VDIVC  =  11     ;;INFOS -INFOS VOL CHARACTERISTICS
+   VDDTO  =  12     ;;INFOS -DEVICE TIME OUT CONSTANT
+   VDVMF  =  14     ;;INFOS -VOL MERIT FACTOR(LH BYTE)
+   VDACC  =  14     ;;INFOS -VOL ACCESSABILITY(LH BYTE)
+   VDPAD  =  14     ;;INFOS -PAD CHARACTER (RH BYTE)
+   VDOID  =  15     ;;INFOS -VOLUME OWNER ID POINTER
+   VDLEN  =  16     ;;INFOS -VOL DEF PACKET LENGTH
    
```

```

C      INFOS VOLUME CHARACTERISTICS (VDIVC)

      PARAMETER
+   ICDDR  =  100000K ;;INFOS -DISABLE DEVICE RESTART
+   ICVLB  =   40000K ;;INFOS -VARIABLE LENGTH BLOCKS
+   ICDVC  =   20000K ;;INFOS -DUPLICATE VOLUME CONTROL
+   ICDSL  =   10000K ;;INFOS -DISABLE SYSTEM LABELING
+   ICPAR  =    2000K ;;INFOS -GENERATE PARITY
+   ICDDC  =    1000K ;;INFOS -DISABLE CONFLICT CHECKING
+   ICCTG  =    200K  ;;INFOS -CONTIGUOUS ALLOCATION
+   ICDFI  =    400K  ;;INFOS -DISABLE FILE INITIALIZATION
+   ICERI  =    40K   ;;INFOS -ENABLE RUN TIME INIT
+   ICERR  =    20K   ;;INFOS -ENABLE RUN TIME RELEASE
+   ICRW0  =    10K   ;;INFOS -REWIND ON VOL OPEN
    
```

INFOS User Parameters for FORTRAN (continued)

C PROCESSING PACKET

PARAMETER		
+ PRSTA	= 1	;;INFOS -STATUS FLAGS
+ PRDAI	= 2	;;INFOS -DATA AREA POINTER
+ PKREC	= 4	;;INFOS -RECORD NUMBER (RAM)
+ PRKTP	= 4	;;INFOS -KEY TABLE POINTER (ISAM)
+ PKLEN	= 6	;;INFOS -RECORD LENGTH
+ PRDSP	= 6	;;INFOS -MAG TAPE DISPOSITION
+ PRDFB	= 9	;;INFOS -DATA RECORD FEEDBACK
+ PRSIL	= 11	;;INFOS -SUB-INDEX LEVEL (RH BYTE)
+ PRLN1	= 11	;;INFOS -SAM & RAM PROC PACKET LEN
+		
+ PRRMF	= 12	;;INFOS -RECORD MERIT FACTOR
+ PRCCW	= 13	;;INFOS -COMMAND CONTROL WORD
+ PRPRA	= 14	;;INFOS -PARTIAL RECORD AREA PTR
+ PRSRL	= 19	;;INFOS -RETURNED LEN (LH BYTE)
+ PRSRS	= 19	;;INFOS -RETURNED STATUS (RH BYTE)
+ PRSID	= 20	;;INFOS -SUBINDEX DEF PACKET PTR
+ PRLN2	= 21	;;INFOS -ISAM & DBAM PACKET LEN

C PROCESSING PACKET STATUS FLAGS (PRSTA)

PARAMETER		
+ PFLOC	= 100000K	;;INFOS -LOCK RECORD
+ PFHLD	= 40000K	;;INFOS -HOLD REQUEST
+ PFUNL	= 20000K	;;INFOS -UNLOCK RECORD
+ PFVCI	= 10000K	;;INFOS -VOLUME CHANGE INDICATOR
+ PFPEV	= 4000K	;;INFOS -PHYSICAL END OF VOLUME
+ PFWIF	= 2000K	;;INFOS -WRITE IMMED IF MODIFIED
+ PFREB	= 1000K	;;INFOS -RECORD EXCEEDS BUF SIZE
+ PFRIN	= 400K	;;INFOS -READ INHIBIT ON (RAM)
+ PFXLS	= 200K	;;INFOS -XFER LENGTH SHORT
+ PFEXL	= 100K	;;INFOS -EXCESSIVE XFER LENGTH
+ PFPEF	= 40K	;;INFOS -PHYSICAL END OF FILE
+ PFVER	= 20K	;;INFOS -VERIFICATION FAILURE
+ PFMTR	= 10K	;;INFOS -MAG TAPE CONTROL ERROR
+ PFUVL	= 4	;;INFOS -USER VOL LABEL PROCESSED
+ PFUHL	= 2	;;INFOS -USER HDR LABEL PROCESSED
+ PFUTL	= 1	;;INFOS -USER TRAILER LAB PROCESSED
+ PFERM	= 15377K	;;INFOS -EXCEPTIONAL RETURN MASK

C RECORD LENGTH FLAG (PRLN)

PARAMETER		
+ PRSPL	= 100000K	;;INFOS -SPECIFIED RECORD LENGTH REQUESTED ;;INFOS -(INPUT TO ISAM OR DBAM READ) ;;INFOS -RECORD EXCEEDS LENGTH REQUESTED ;;INFOS -(RETURNED FROM ISAM OR DBAM READ)

C INDEX COMMAND CONTROL FLAGS (PRCCW)

PARAMETER		
+ CCKEY	= 100000K	;;INFOS -KEYED
+ CCREL	= 40000K	;;INFOS -RELATIVE TO LUR POS
+ CCMC1	= 20000K	;;INFOS -MOTION CONTROL FIELD
+ CCMC2	= 10000K	;;INFOS -
+ CCMC3	= 4000K	;;INFOS -
+ CCSCP	= 2000K	;;INFOS -SET CURRENT POSITION

INFOS User Parameters for FORTRAN (continued)

```

+ CCINV = 400K ;;INFOS -INVERTED ENTRY
+ CCSPR = 100K ;;INFOS -SUPPRESS PARTIAL RECORD
+ CCSDB = 20K ;;INFOS -SUPPRESS DATA BASE
+ CCLOG = 10K ;;INFOS -LOGICAL KEY DELETE
+ CCLOC = 4 ;;INFOS -LOCAL LOG DELETE
+ CCGLB = 2 ;;INFOS -GLOBAL LOG DELETE
    
```

C MOTION CONTROL SPECIFIERS (CCMC1-CCMC3)

```

PARAMETER
+ CCMCM = 34000K ;;INFOS -FIELD MASK
+ CCFWD = 0 ;;INFOS -FORWARD
+ CCBK = 4000K ;;INFOS -BACKWARD
+ CCDWN = 10000K ;;INFOS -DOWN
+ CCDFW = 14000K ;;INFOS -DOWN & FORWARD
+ CCUFW = 20000K ;;INFOS -UP & FORWARD
+ CCUBK = 24000K ;;INFOS -UP & BACKWARD
+ CCUP = 30000K ;;INFOS -UP
+ CCSTA = 34000K ;;INFOS -STATIC
    
```

C SYSTEM RETURNED STATUS FLAGS (PRSR)

```

PARAMETER
+ SRLLD = 200K ;;INFOS -LOCAL LOGICAL DELETE
+ SRDUP = 100K ;;INFOS -DUPLICATE KEY
+ SRDRL = 40K ;;INFOS -DATA RECORD LOCKED
+ SRGLD = 20K ;;INFOS -GLOBAL LOGICAL DELETE
+ SRLDM = 220K ;;INFOS -LOGICAL DELETE MASK
+ SRSFM = 360K ;;INFOS -STATUS FIELD MASK
    
```

INFOS User Parameters for FORTRAN (continued)

```

C      MAG-TAPE CONTROL PROCESSING PACKET

      PARAMETER
+     PRCFC  =  4      ;;INFOS -CONTROL FUNCTION CODE
+     PRNWD  =  5      ;INFOS -NUMBER OF WORDS

C      MAG-TAPE CONTROL FUNCTION CODES   (PRCFC)

      PARAMETER
+     MCSFF  =  0      ;;INFOS -SPACE FORWARD FILE
+     MCSBF  =  1      ;;INFOS -SPACE BACKWARD FILE
+     MCRD   =  2      ;;INFOS -READ
+     MCWRT  =  3      ;;INFOS -WRITE
+     MCWRF  =  4      ;;INFOS -WRITE EOF
+     MCREW  =  5      ;;INFOS -REWIND
+     MCSFR  =  6      ;;INFOS -SPACE FORWARD REC
+     MCSBR  =  7      ;;INFOS -SPACE BACKWARD REC
+     MCERS  =  8      ;INFOS -ERASE

C      POINT PROCESSING PACKET

      PARAMETER
+     PRMOD  =  2      ;;INFOS -INPUT MODE
+     PRHLB  =  3      ;;INFOS -HI LOGICAL BLOCK
+     PRLLB  =  4      ;;INFOS -LOW LOGICAL BLOCK
+     PRBOF  =  5      ;INFOS -BYTE OFFSET

C      POINT INPUT MODE   (PRMOD)

      PARAMETER
+     PMEUF  =  0      ;;INFOS -POINT TO EOF
+     PMLBN  =  1      ;INFOS -LOGICAL BLOCK NUM

C      LINK SUB-INDEX PROCESSING PACKET

      PARAMETER
+     PROKT  =  2      ;;INFOS -DEST KEY TABLE PTR
+     PRSKT  =  4      ;;INFOS -SOURCE KEY TABLE PTR
+     PRDCC  =  12     ;;INFOS -DEST COMMAND CONTROL
+     PRSCC  =  13     ;INFOS -SOURCE COMMAND CONTROL

C      KEY DEFINITION PACKET

      PARAMETER
+     KDTPY  =  1      ;;INFOS -KEY TYPE FLAGS (LH BYTE)
+     KDKYL  =  1      ;;INFOS -KEY LEN (RH BYTE)
+     KDKYP  =  2      ;;INFOS -KEY POINTER
+     KDDKU  =  4      ;;INFOS -DUP KEY OCCURENCE
+     KOLEN  =  5      ;INFOS -KEY DEF PACKET LENGTH

C      KEY TYPE FLAGS   (KDTPY)

      PARAMETER
+     KTDUP  =  10000K  ;;INFOS -DUPLICATE KEY
+     KIGEN  =  40000K  ;;INFOS -GENERIC KEY
+     KTAPX  =  20000K  ;INFOS -APPROX KEY

```

INFOS User Parameters for FORTRAN (continued)

C		PROCESSING FUNCTION CODES	
PARAMETER			
+	POINT	= 0	;;INFOS -POINT
+	CNTRL	= 1	;;INFOS -CONTROL
+	FEUV	= 2	;;INFOS -FORCE END OF VOL
+	ICLOSE	= 3	;;INFOS -INFOS CLOSE
+	SETX	= 4	;;INFOS -SET EXCLUSIVE USE
+	RELX	= 5	;;INFOS -RELEASE EXCLUSIVE USE
+	TRNCT	= 6	;;INFOS -TRUNCATE BLOCK
+	IREAD	= 7	;;INFOS -INFOS READ
+	IWRITE	= 8	;;INFOS -INFOS WRITE
+	DEFSI	= 9	;;INFOS -DEFINE SUB-INDEX
+	LNKSI	= 10	;;INFOS -LINK SUB-INDEX
+	DELRC	= 11	;;INFOS -DELETE RECORD
+	DELSI	= 12	;;INFOS -DELETE SUB-INDEX
+	RETST	= 13	;;INFOS -RETURN STATUS
+	RETHK	= 14	;;INFOS -RETURN HIGH KEY
+	RETKY	= 15	;;INFOS -RETURN KEY
+	REINS	= 16	;;INFOS -REINSTATE REC
+	RDDIR	= 17	;;INFOS -READ DIRECT
+	WRDIR	= 18	;;INFOS -WRITE DIRECT
+	RELRC	= 19	;;INFOS -RELEASE REC
+	RELSE	= 20	;;INFOS -RELEASE BUFFER
+	REWRT	= 21	;;INFOS -REWRITE
+	RETDF	= 22	;;INFOS -RETURN SUB-INDEX DEF
+	PRERD	= 23	;;INFOS -PREREAD

INFOS User Parameters for FORTRAN (continued)

C	PARAMETER	INFOS ERROR CODES
+	IOILF = 131	;;INFOS -ILLEGAL FUNCTION
+	IUVTI = 132	;;INFOS -VARIABLE LENGTH TRANSFER
+		;;INFOS -ILLEGAL ON THIS DEVICE
+	IURDU = 133	;;INFOS -REWRITE ON DISK ONLY
+	IOIFD = 134	;;INFOS -ILLEGAL FUNCTION FOR DEV
+	IOOPE = 135	;;INFOS -OPEN PROCESSING ERROR
+	IORFC = 136	;;INFOS -REC FMT & FUNC CONFLICT
+	IOEXF = 137	;;INFOS -FILE IN USE
+	IOLOK = 138	;;INFOS -FILE LOCKED
+	IOFNO = 139	;;INFOS -FILE NOT OPEN
+	IOPCF = 140	;;INFOS -PERIPHERAL CONFLICT
+	IUVFP = 141	;;INFOS -VL FILE PROCESSING ERROR
+	IOURC = 142	;;INFOS -UNRESOLVED RESOURCE CONFLICT
+	IORMP = 143	;;INFOS -REWRITE MODE PROCESSING ERROR
+	IODVF = 144	;;INFOS -DUPLICATE VL FILE
+	IOBEW = 145	;;INFOS -BLOCK SIZE EXCEEDS WINDOW SIZE
+	IUVME = 146	;;INFOS -VIRTUAL MEMORY EXHAUSTED
+	IOXTL = 147	;;INFOS -TRANSLATE TABLE LOAD ERROR
+	IUVFE = 148	;;INFOS -VL FILE OPEN ERR
+	IOVCE = 149	;;INFOS -VL FILE CLOSE ERR
+	IOIFO = 150	;;INFOS -INSUF FREESPACE FOR OPEN
+	IOLEF = 151	;;INFOS -LOGICAL END OF FILE
+	IOUXS = 152	;;INFOS -USER TRANSLATE SPECIFICATION ERROR
+	IUNSV = 153	;;INFOS -NO SUCH VOLUME
+	IONOH = 154	;;INFOS -NO HOLD ON LOCKED REQUEST
+	IUNMD = 155	;;INFOS -NO MORE DISK SPACE
+	IOROF = 156	;;INFOS -RAM ACCESS OUTSIDE FILE
+	IOICL = 157	;;INFOS -ILLEGAL CLOSE
+	IOPIC = 158	;;INFOS -PHYSICAL I/O ERROR
+	IORDE = 159	;;INFOS -RESIDUAL DISK ERROR
+	IOTMO = 160	;;INFOS -DISK OR MAG-TAPE TIME-OUT
+	IOIAM = 161	;;INFOS -ILLEGAL ACCESS METHOD
+	IOXER = 162	;;INFOS -ILLEGAL TRANS REQUEST
+	IOPRO = 163	;;INFOS -PREOPEN OPEN ERROR
+	IOPRC = 164	;;INFOS -PREOPEN CLOSE ERROR
+	IUFCE = 165	;;INFOS -FILE CLOSE ERROR
+	IOROE = 166	;;INFOS -RDOS OPEN ERROR
+	IUVAX = 167	;;INFOS -VOLUME ALREADY EXISTS
+	IOZDX = 168	;;INFOS -ZERO LEN DISK XFER REQ
+	IOIUC = 169	;;INFOS -ISAM UPDATE CONFLICT
+		;;INFOS -TABLE OVERFLOW
+	IONMR = 170	;;INFOS -INDEX NAME SPEC ERROR
+	IUNSI = 171	;;INFOS -NO SUCH INDEX
+	IONTL = 172	;;INFOS -NAME TOO LONG
+	IONNS = 173	;;INFOS -NO NODE SPACE
+	IOMTE = 174	;;INFOS -MAG-TAPE I/O ERROR
+	IODNS = 175	;;INFOS -DEVICE NOT SUPPORTED
+	IOEVO = 176	;;INFOS -OUTPUT END VOLUME ERROR
+	IOEVI = 177	;;INFOS -INPUT END VOLUME ERROR
+	IOCMP = 178	;;INFOS -COMPARE ERROR (ISAM)
+	IOKEZ = 179	;;INFOS -RESOLUTION ERROR (ISAM)
+	IOIRM = 180	;;INFOS -ILLEGAL REL MOTION
+	IOINA = 181	;;INFOS -INVALID NODE ADDRESS
+	IOICE = 182	;;INFOS -INVALID CURRENT ENTRY
+	IOTLV = 183	;;INFOS -TOP LEVEL ERROR
+	IOSNA = 184	;;INFOS -SUB INDICES NOT ALLOWED
+	IUSNP = 185	;;INFOS -SUB-INDEX NOT DEFINED
+	IOESI = 186	;;INFOS -END OF SUB-INDEX
+	IODPE = 187	;;INFOS -DELETE POSITIONING ERROR

INFOS User Parameters for FORTRAN (concluded)

+	IOMKW	=	188	;;INFOS	-MULTI KEY WRITE ERROR
+	IOIKL	=	189	;;INFOS	-ILLEGAL KEY LENGTH
+	IOIEN	=	190	;;INFOS	-INVALID ENTRY NUMBER
+	IOIPS	=	191	;;INFOS	-ILLEGAL COMMAND CONTROL
+	IUKAE	=	192	;;INFOS	-KEY ALLREADY EXISTS
+	IOKPE	=	193	;;INFOS	-KEY POSITIONING ERROR
+	IOIRL	=	194	;;INFOS	-INVALID RECORD LENGTH
+	IORNP	=	259	;;INFOS	-DATA BASE REC NOT PRESENT
+	IONTB	=	260	;;INFOS	-MIN NODE SIZE TOO BIG
+	IUNTS	=	261	;;INFOS	-MIN NODE SIZE TOO SMALL
+	IODRL	=	262	;;INFOS	-DATA RECORD LOCKED
+	IOSIA	=	263	;;INFOS	-SUB-INDEX IN USE
+	IOVER	=	264	;;INFOS	-VERSION CONFLICT ERROR
+	IOSIL	=	265	;;INFOS	-SUB-INDEX LINK COUNT
+				;;INFOS	-OVERFLOW
+	IOALS	=	266	;;INFOS	-ALREADY LINKED
+				;;INFOS	-IO SUB-INDEX
+	IOSLO	=	267	;;INFOS	-SUB-INDEX LEVEL OVERFLOW
+	IOSSI	=	268	;;INFOS	-SUB-INDEX HAS SUB-INDEX
+				;;INFOS	-DELETE SUB-INDEX ERROR
+	IODWK	=	269	;;INFOS	-ATTEMPT TO DELETE ENTRY
+				;;INFOS	-WITHOUT KEYED ACCESS
+	IOENL	=	270	;;INFOS	-INDEX ENTRY LOCKED
+	IOWK	=	271	;;INFOS	-NO WRITE WITHOUT KEY
+	IOILL	=	272	;;INFOS	-ILLEGAL LABEL
+	IOILS	=	273	;;INFOS	-ILLEGAL LABEL SPEC
+	IOVID	=	274	;;INFOS	-VOL ID DOESNT MATCH
+	IOFID	=	275	;;INFOS	-FILE ID DOESNT MATCH
+	IOFSQ	=	276	;;INFOS	-FILE SEQ NUM DOESNT MATCH
+	IOGEN	=	277	;;INFOS	-GEN NUM DOESNT MATCH
+	IOEXD	=	278	;;INFOS	-EXP DATE NOT EXPIRED
+	IOBCT	=	279	;;INFOS	-BLOCK COUNT INCORRECT
+	IORFT	=	280	;;INFOS	-RECORD FORMAT CONFLICT
+	IOFSN	=	281	;;INFOS	-FILE SECTION NUMBER
+	IOEIL	=	282	;;INFOS	-EXCESSIVE POSITION LEVELS
+	IOSLS	=	283	;;INFOS	-SYSTEM LOAD SIZE ERROR
+	IOFNF	=	284	;;INFOS	-TAPE FILE NOT FOUND
+	IOBTS	=	285	;;INFOS	-BLOCKSIZE < 8 BYTES
+	IORTL	=	286	;;INFOS	-RECORD+OVERHEAD > BLOCKSIZE
+	IOWNE	=	287	;;INFOS	-WRITE IS NOT AT END-OF-FILE
+				;;INFOS	-FOR SHARED SAM UPDATE FILE
+	IOOWO	=	288	;;INFOS	-WRITE ALLOWED ONLY FOR ONE
+				;;INFOS	-USER OF SHARED SAM UPDATE FILE
+	IOSPL	=	289	;;INFOS	-SPOOLING ON ILLEGAL DEVICE
+	IURKR	=	290	;;INFOS	-RETRIEVE KEY ERROR
+	IODIP	=	291	;;INFOS	-DELETE INDEX POSITION ERROR
+	IOMPR	=	292	;;INFOS	-SPACE MANAGEMENT INCONSISTENCY
+	IOSTR	=	293	;;INFOS	-SEARCH CP TABLE ERROR

End of Appendix

Appendix D

INFOS System Error Messages

The chart which follows attempts to shed some light on the messages you'll receive if either you or the system go awry. We have included the octal code number of the error (which the system will return to your program), the mnemonic name of the error, a brief description of what that code means, and a reference to the chapter(s) within this manual where you can find further information about the operation(s) mentioned. In this last column, we use a roman numeral to denote the section of this manual in which you should look (e.g., Part I), and a decimal number to denote the chapter within that part (e.g., I,6 means refer to Part I, Chapter 6).

Note that you can get the message associated with any error code by using the INFOSER utility. To do this, simply type in the following:

INFOSER n

where n is the error code number.

The system will return the code number and its message.

Note to All FORTRAN Programmers:

The error code which the INFOS system returns to your FORTRAN programs will be in decimal, not octal. In addition, it will be three numbers higher than that returned to Assembly programs. Therefore, before using INFOSER, subtract three from the error code returned to your program by the system, then type in

INFOSER n/D

where n is the returned error code number minus three.

You may also perform the following steps on the decimal number you receive if you want to use this appendix:

1. Subtract three from the error code returned by the system.
2. Convert the result to an octal number.
3. Look up that *octal* number in the following listing.

For example, if the system returned error code 287 to your program, you would do this:

1. $287 - 3 = 284_{10}$
2. $284_{10} = 434_8$
3. Error code 434 = IOWNE

Code	Name	Description	Reference
200	IOUER	You have requested either an unknown function (check your spelling), or one which doesn't apply to either the access method or the processing mode you selected.	I-2,3,4,or5; or II-2,3,4,or 5
201	IOVTI	The system has encountered a block whose length is not equal to the block size you specified for this run. This indicates that you requested a write operation but neglected to set ICVLB (Allow Variable Length Blocks) in the FDP.	II-6
202	IORDO	You specified the Rewrite mode (F1RER in the FDP) for a nondisk device.	II-6; I-2
203	IOIFD	You requested a Mag Tape Control function for a nontape device.	II-6;II-2
205	IORFC	You requested the Rewrite function for a Data Sensitive record.	I-2;II-2
206	IOEXR	You have attempted to gain exclusive use of a file already in use.	I-2;II-6
207	IOLOK	You have attempted to open a file already opened for exclusive use.	I-2;II-6
210	IOFNO	You have issued a processing function request for a file which you have not successfully Pre-opened and Opened. (Check AC2 for Pre-open and Open error messages.)	II-1
211	IOPCE	You have attempted to open a nondisk device which is already open.	II-1
212	IOIRE	The system has encountered an error while attempting to process your file's .VL file.	II-6
213	IOPND	The system has timed out while waiting for an internal resource required to complete your current request.	I-6;III-E
214	IOIRI	You have issued a rewrite inverted request and the address of the data record (in the index entry) does not match the address you specified.	I-5;II-5
215	IODVF	You have attempted to use a currently existing name for your file (i.e., change your file's name).	I-6
216	IOBEW	Your block size exceeds your specified 'window' size.	I-6
217	IOVME	There isn't enough room in either the foreground or the background (whichever you're using) to build the buffers for the file you're trying to open.	I-6
220	IOXLE	You requested code translation, but the system cannot load its translation table into memory.	II-6
221	IOVFE	The system cannot open your file as requested because of an error in the .VL file or because the file doesn't exist.	II-6
222	IOVCE	The system has encountered an error while attempting to close the .VL file.	II-6
223	IOMEM	You did not provide enough file control space to open your file as requested.	I-6
224	IOLEF	The system has determined that it cannot process your current request without going beyond the end of the file.	
225	IOUTS	You have made an error in specifying what you want translated.	II-6
226	IONSV	You have requested access to a volume which doesn't exist.	I-1
227	IONOH	You have requested access to a record which is locked, but you did not specify "Hold" in your request.	I-2 or 3; II-6

Code	Name	Description	Reference
230	IONMD	There is no more disk space available to continue processing.	I-2,3,4, or 5
231	IOEOF	The system has determined that the record you want is outside the bounds of your existing RAM file.	I-3
232	IOCNV	The system has encountered an error while attempting to close one volume and open another.	II-6
234	IOUOR	The system cannot write a block of data to your file. (This usually indicates a hardware problem.)	
235	IOTMO	Either your disk or your mag tape has timed out.	I-6; III-E
237	IOXER	You have requested translation, but have not provided all the necessary parameters.	II-6
241	IOPRC	The system either can't find (or has encountered difficulty processing) the file's Permanent File Specification.	II-1
242	IOFCE	The system has encountered an error while trying to close the file's PFS.	II-1
243	IOFOE	The INFOS system cannot complete the operating system phase (RDOS) of the file opening procedure.	II-1
244	IOFAX	You have attempted to create a file whose name is already in use (use another filename).	
245	IOZDX	You have requested a zero length transfer for a disk.	
247	IONMR	You have attempted to open an index which does not belong to the database you specified.	II-4, 5 or 6
250	IONSI	The system is not able to close all the open files. (You can correct this by rebooting the system before processing.)	
251	IONTL	The filename you have specified is too long. (Filenames cannot exceed 48 characters, including all device specifiers, delimiters and extensions.)	I-6
252	IONIS	You have no more available space for your ISAM or DBAM file.	I-6; III-B
253	IOMTE	The system has detected an error while processing a magnetic tape request.	I-2; III-A
254	IODNS	You have attempted to open a file residing on a device which the INFOS system does not support.	I-6; III-E
256	IOEVI	The system has encountered the end of a volume while processing an input file.	
257	IOCMP	The system has encountered invalid data in an index. THIS IS A FATAL ERROR.	
260	IOREZ	(Same as IOCMP.)	
261	IOSPE	The system has encountered an error while trying to position to the entry you specified in the 'Key' and/or 'Relative Motion' field(s).	I-4 or 5; II-4 or 5; II-6
262	IOINA	The system has encountered difficulty while trying to access an index. (Try reloading your file.)	
263	IOICE	You have specified an invalid direction of relative motion from your current position. (To rectify this, try moving Up or using keyed access.)	I-5; II-5

Code	Name	Description	Reference
265	IOSNA	You have attempted to Define, Link, or Delete a subindex where subindexes are not allowed.	I-5; II-5,6 III-B
266	IOSNP	You have attempted to Link to or Delete a nonexistent subindex.	I-5; II-5,6; III-B
267	IOEST	You have attempted to use Relative Motion beyond the bounds of the subindex.	I-5; II-5; III-B
270	IODPE	You have tried to delete an index entry on which someone else is currently positioned.	I-4 or 5
271	IOMKW	You have issued a Write request using a multilevel key, and one of the keys you specified does not match any other at its indicated subindex level.	I-5; II-5,6
272	IOIKL	You have attempted to write a key which is either zero length or too long.	I-4 or 5; II-6
273	IOIEN	The system has encountered an error while trying to access your index. This indicates either an index structure problem or a system bug.	III-B
274	IOIPS	You did not specify either CCKEY or CCREL in PRCCW of your Extended Processing Packet.	II-6; I-4 or 5; II-4 or 5.
275	IOKAE	You have tried to Write an index entry which already exists in that subindex.	I-5; III-B
276	IOKPE	The system has encountered an error while trying to position to your specified entry.	I-4 or 5; II-4 or 5
277	IOIRL	1) You have attempted to Rewrite a SAM record with a length different from the one read, <i>or</i> 2) You have attempted to Write an ISAM or DBAM record that is longer than your specified page length.	I-2 I-4 or 5
400	IONDR	No database record exists for the index entry you specified.	I-4 or 5
401	IONTB	The node size you specified is too large for your specified page size.	III-B
402	IONTS	The node size you specified is not large enough to hold three index entries.	III-B
403	IODRL	The database record associated with the index entry you specified is currently locked.	I-4 or 5; II-4 or 5
405	IOVER	You have attempted to open a file which was created under a system whose parameter versions are different than your current system.	II-6
406	IOSTL	Too many subindex links currently exist to complete your request.	I-5; II-5; III-B
407	IOSAE	You have attempted to Define or Link a subindex to a subindex entry which already owns a subindex.	I-5; II-5; III-B
410	IOSLO	You have attempted to define a subindex at level 257.	I-5; II-5; III-B
411	IOSST	You have attempted to delete a subindex which owns at least one subindex and which is currently linked to only one other subindex.	I-5; II-5; III-B
412	IODWK	You have attempted to perform physical deletion, but you did not specify Keyed access.	I-5; II-5

Code	Name	Description	Reference
413	IOENL	The index entry you have attempted to access is locked.	I-4 or 5; II-4 or 5
414	IOWWK	You have attempted to write a new record without specifying Keyed access.	I-4 or 5; II-4,5 or 6
415	IOILL	You have used an illegal label format on a labeled magnetic tape file. (Usually this means that the system did not find one or more of the required labels.)	III-A
416	IOILS	The system does not support the label type and level which you specified in either the Volume Initialization Packet or in the File Definition Packet.	II-6; III-A
417	IOVID	The volume identifier you specified does not match that on the volume label.	III-A; II-6
420	IOFID	The file identifier you specified does not match that on the volume label.	II-6; III-A
421	IOSFQ	The file sequence number you specified does not match any of those on the tape.	III-A
422	IOGEN	The generation number you specified does not match that on the record you requested.	III-A
423	IOEXD	You have attempted to write over a file whose expiration date has not yet been reached.	III-A
424	IOBCT	The block count of the trailer label does not agree with the system's count.	III-A
425	IORFT	You have specified a record format which conflicts with that specified in the header label.	I-2; III-A
426	IOFSN	The file section number on the header label is not correct.	III-A
427	IOLVR	You have attempted to access more subindex levels than exist in your index.	I-5
431	IOFNF	The system has not been able to locate a tape file with the specifier you indicated.	III-A
432	IOBTS	You have specified a page size of less than eight (8) bytes for a database file. (Database pages must be at least eight bytes long.)	I-5
434	IOWNE	You have attempted to Write in a shared SAM file (opened in the Update mode) at some point other than the end-of-file.	I-2
435	IOOWO	Someone else is currently writing in the SAM file for which you issued a write request.	I-2
436	IOSPL	You have specified spooling for a device other than a line printer or teletype.	I-2
440	IODIP	You have attempted to Delete in an index file while someone else is positioned in that index for the same purpose.	I-4 or 5; II-4 or 5
441	IOMPR	You specified Space Management and the map doesn't agree with the page space available. (When this happens, the system will use the page size rather than the map size.)	
442	IOSTR	You have requested either Delete or Define Subindex, and the system cannot find the original of your specified duplicate key.	

End of Appendix

Appendix E Device Characteristics

	TTI	TTO	PTR	PTP	PLT	CDR	LPT	A/D	CRT	MUX	MTA	CAS	F.H. Disk	M.H. Disk
Device Restart	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No
Variable Blocks	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No
Normally Single Volume	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No
System Labels											Yes	Yes		
Parity	No	No	No	No	No	No	No	No	No	No	No	No	No	No
Conflict Checking	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Random Allocation											Yes	Yes	Yes	Yes
Initialize Contiguous Files											Yes	Yes	Yes	Yes
Runtime Initialization											Yes	Yes		
Runtime Release											Yes	Yes		
Rewind on Volume Open											Yes	Yes		
Seconds to Timeout	65535	30	30	30	180	10	15	3	600	180	15	600	3	5

End of Appendix

Index

In this Index, page references are shown as follows: the roman numeral denotes the section of the manual (e.g., Part I) and the decimal number denotes the chapter and page reference. Thus, I-4-2 indicates that you will find information about that topic in Part I, Chapter 4, page 2; f and ff are used to denote the page or pages following the reference.

access, concurrent

ISAM I-4-2

access methods I-1-1

keyed I-4-2

relative I-4-8

(see also relative access and keyed access)

address space I-6-5

allocation (contiguous vs. random) of disk space I-6-10

ANSI levels III-A-1ff

appending records I-2-12

approximate keys I-5-17

arguments (FORTRAN) III-C-2

assembly language parameters II-7-1, II-7-15

background I-6-5

BCBs (Buffer Control Blocks) I-6-3ff

BF

see branching factors

BLDFDP II-7-2ff

BLDIPKT II-7-15

BLDKDP II-7-11

BLDLSP II-7-13

BLDMTC II-7-10

BLDPNT II-7-10

BLDPP II-7-8

BLDULT II-7-15

BLDVDP II-7-6

BLDVIP II-7-14

blocking factor III-B-8

block-packing I-1-5

ISAM and DBAM

see page size

RAM I-3-1

SAM I-2-5f

size I-1-4f

branching factors III-B-2

buffers

control blocks (BCBs) I-6-3ff

DBAM I-5-6ff

I/O management I-1-5, I-6-8f

I/O space I-6-6f, I-6-2

ISAM I-4-4ff

RAM I-3-1

SAM I-2-1

building/processing routines

FORTRAN III-C-2, III-C-10

byte pointers II-1-1

call formats (FORTRAN) III-C-3

calls, system II-1-3

cells I-6-2

channel number II-1-3, III-C-9

close request

RAM II-3-5

SAM II-2-6

CMT instruction III-C-4

COBOL, Prefatory Note, Part II

code, resident RDOS/INFOS I-6-2

combined access (keyed plus relative) I-5-13, II-5-15

compression, key I-5-5, I-6-3

concurrent tasking I-1-2

contiguous allocation I-6-10

ISAM I-4-5

RAM I-3-2

SAM I-2-2

control area, file I-6-3

control blocks I-6-3ff

Create Update mode I-1-2, II-4-1ff, II-5-1ff

CTR instruction I-1-4

current position I-4-2

database

files III-B-8

records I-4-2

suppress

ISAM I-4-10, II-4-10f

DBAM I-5-17, II-5-16f

- data sensitive record I-1-3
 - transfers I-1-5
- data transfers I-1-4, III-B-8
 - see also appropriate access method
- DBAM (Data Base Access Method)
 - access methods I-5-11, II-5-15
 - approximate keys I-5-17
 - automatic key compression I-5-5
 - creation I-5-6ff
 - define subindex I-5-16, II-5-19
 - delete I-5-16, II-5-20
 - delete subindex I-5-16, II-5-21
 - file inversions I-5-4
 - general concepts I-1-2, I-5-1ff
 - generic keys I-5-17
 - index levels I-5-1, I-5-6
 - link subindex I-5-16, II-5-19
 - linked subindexes I-5-4
 - lock/unlock I-5-17, II-5-17
 - multiple indexes I-5-4
 - opening I-5-10, II-5-1ff
 - optimized distribution I-5-5
 - partial records I-5-3
 - processing I-5-11ff, II-6-20
 - programming II-5-1ff
 - read I-5-15, II-5-16
 - reinstate I-5-16, II-5-21
 - retrieve high key I-5-17, II-5-21
 - retrieve key I-5-17, II-5-21
 - retrieve status I-5-16, II-5-22
 - retrieve subindex definition I-5-17, II-5-21
 - rewrite I-5-16, II-5-18
 - subindexes I-5-1
 - suppress database I-5-17, II-5-16
 - suppress partial record I-5-17, II-5-16
 - temporary indexes I-5-5
 - volume merit factors I-5-8
 - write I-5-15, II-5-17
- DEFAULT (in FORTRAN) III-C-2
- define subindex I-5-16, II-5-19
- definition, retrieve subindex I-5-17, II-5-22
- delete subindex I-4-10, I-5-16, II-4-12, II-5-21
- deleting records
 - ISAM I-4-9f, II-4-12
 - DBAM I-5-16, II-5-20
- delimiter tables I-1-4
- device timeouts I-6-10, III-E
- direct access
 - see RAM
- disable file initialization I-2-2, I-4-5
- disk files
 - processing I-1-7
 - SAM I-2-1ff, II-2-1ff
- disk space allocation I-6-10
- distribution, optimized I-5-5
- duplicate keys I-4-2

- end-of-volume labels III-A-1ff
- entries, subindex III-B-1f
- error conditions (FORTRAN) III-C-3
- error messages III-D
- extended processing packets II-1-2, II-6-20
 - building II-7-8
- extensions (to filenames) I-6-10

- F4INFOS.LB III-C-1ff
- F5INFOS.LB III-C-1ff
- FCBs (File Control Blocks) I-6-3ff
- FDPs
 - see file definition packets
- FEOV I-3-6, II-3-5
- File
 - control area I-6-3f
 - control blocks (FCBs) I-6-3f
 - creation I-1-7
 - DBAM I-5-6
 - ISAM I-4-3
 - RAM I-3-1
 - SAM I-2-2ff
 - extensions I-6-10
 - initialization (SAM) I-2-2
 - inversion I-5-4
 - naming I-6-8, I-1-7
 - multivolume I-1-6
 - opening I-1-7, II-1-3
 - DBAM I-5-10ff, II-5-1ff
 - ISAM I-4-7, II-4-1ff
 - RAM I-3-3, II-3-1ff
 - SAM I-2-4ff, II-2-1ff
 - processing
 - DBAM I-5-11ff, II-5-15ff, III-C-10
 - ISAM I-4-8ff, II-4-9ff, III-C-10
 - RAM I-3-3ff, II-3-4ff, III-C-10
 - SAM I-2-12ff, II-2-4ff, III-C-10
- file definition packets II-1-1f, II-6-1ff
 - building II-7-2 (assembly), III-C-4f (FORTRAN)
 - parameters II-6-9
- file specification, permanent I-1-7, I-4-3, II-1-4f
- FINFOS.FR (FORTRAN) III-C-3, III-C-12ff
- fixed length records I-1-3
 - transfers I-1-5
- force end of volume (FEOV) I-3-6, II-3-5
- foreground I-6-5
- FORTRAN (IV or 5) III-C-1ff
 - call formats III-C-3ff
 - loading your program III-C-3
 - user parameters III-C-12ff

- general processing packets II-1-2, II-6-18ff
 - building II-7-8
- generation number III-A-9
- generation, system (INFOS) I-6-1
- generic keys I-5-17
- global deletion I-4-9f, II-4-12, II-5-20
- 'grounds (foreground and background) I-6-5

- header labels III-A-6, III-A-10ff
- hierarchical replacement I-4-4
- high key, retrieve
 - ISAM I-4-10, II-4-13
 - DBAM I-5-17, II-5-21
- high priority nodes I-5-6, I-6-8
- Hold feature I-3-6
- HPNs
 - see high priority nodes
- IBM levels III-A-1ff
- .IINFOS II-1-3
- Index
 - entries III-B-2
 - levels I-5-1, I-5-6
 - multiple I-5-4
 - structure I-4-2ff, I-5-1ff, III-B-1
- Indexcalc utility III-B-2, III-B-8, I-6-3
- INFFDP III-C-1, III-C-4
- INFIFDP III-C-1, III-C-5
- INFINIT III-C-2, III-C-10
- INFKEY III-C-1, III-C-7
- INFLSP III-C-1, III-C-8
- INFOPEN III-C-2, III-C-9
- INFOS (FORTRAN call) III-C-2, III-C-10
- .INFOS 77 II-1-4
- INFOSER utility III-D-1
- INFOX III-C-2, III-C-11
- INFPREP III-C-2, III-C-9
- INFULT III-C-1, III-C-8
- INFVDP III-C-1, III-C-6
- INFVIP III-C-1, III-C-6
- Initialization
 - labeled mag tape III-A-6ff
 - at runtime III-A-7
 - via system calls III-A-6
 - SAM files I-2-2
 - volume initialization packet II-6-31; III-C-10
- Input mode I-1-2
- interface
 - Assembly language II-7-1ff
 - FORTRAN IV or 5 III-C-1
- inversions, file I-5-4
- invoking Assembly language interface II-7-15
- I/O buffer space I-6-6
 - buffer management I-6-8
- ISAM (Indexed Sequential Access Method)
 - concurrent access I-4-2
 - creation I-4-3
 - current position I-4-2, II-4-9
 - database records I-4-2
 - duplicate keys I-4-2
 - general concepts I-1-1, I-4-1
 - global/local deletion I-4-9f, II-4-12
 - hierarchical replacement I-4-4
 - index file I-4-2ff
 - key length I-4-4
 - keyed access I-4-2, II-4-9
 - lock/unlock I-4-10, II-4-10ff
 - node size I-4-3
 - occurrence numbers I-4-2
 - opening I-4-7, II-4-1ff
 - pages I-4-3
 - permanent file specification I-4-3f
 - processing I-4-8ff, II-4-8ff, II-6-20
 - programming II-4-1ff, II-6-20
 - read I-4-9, II-4-10
 - read-after-write verification I-4-4, II-4-9
 - reinstate I-4-10, II-4-12
 - relative access I-4-8, II-4-9
 - retrieve high key I-4-10, II-4-13
 - retrieve key I-4-10, II-4-13
 - retrieve status I-4-10, II-4-13
 - rewrite I-4-9, II-4-11
 - space management I-4-3
 - suppress database I-4-10, II-4-10
 - volume definition I-4-5f
 - volume table entries I-4-4
 - write I-4-9, II-4-11
- keys
 - access I-1-1, I-4-2, II-4-9
 - approximate and generic I-5-17
 - compression I-5-5, I-6-3, III-B-1
 - definition packets II-1-2, II-6-24, II-7-11, III-C-7
 - DBAM I-5-1ff
 - duplicate I-4-2
 - ISAM I-4-4
 - retrieve high
 - ISAM I-4-10, II-4-13
 - DBAM I-5-17, II-5-21
 - retrieve
 - ISAM I-4-10, II-4-13
 - DBAM I-5-17, II-5-21
- key table II-4, II-5-15, II-6-25
- keyword parameters II-7-1
- labeled magnetic tapes I-1-6, III-A-1ff
 - control processing packet II-7-10
 - creation I-2-4ff
 - initialization I-2-4, III-A-1, III-A-6ff
 - opening I-2-7ff, II-2-1ff
 - positioning III-A-9ff
 - processing III-A-9
 - user labels III-A-6
- label table, user II-7-15
- label types and levels III-A-1ff
- LBINIT (utility) III-A-6
- least-recently-used method I-1-5, I-6-8f
- line printers (as files) I-1-7
- linking subindexes I-5-4, I-5-16, II-6-28, II-7-13
- link subindex processing packets II-1-2, II-6-28
 - building II-7-13 (Assembly); III-C-8 (FORTRAN)
- local deletion I-4-9f
- lock
 - DBAM I-5-17, II-5-16ff
 - ISAM I-4-10, II-4-10ff
 - RAM I-3-5, II-3-4
- logical volumes I-1-6

DataGeneral

SOFTWARE DOCUMENTATION

LRU

see least-recently-used method

Macroassembler II-7-1ff

magnetic tape

- control request II-2-7, II-6-33f, II-7-10
- labeled III-A, I-2-4ff, I-1-6, II-2-1ff
- unlabeled I-6-11

management, buffer I-1-5

memory space I-6-1ff

merit factors I-5-8

modes

Create Update I-1-2

Input I-1-2

Output I-1-3

processing I-1-2

Update I-1-2

multiple indexing I-6-4

see also DBAM

multitasking I-1-2

multivolume files I-1-6

names, file I-1-7, I-6-8

NIL (in FORTRAN) III-C-2

nodes I-4-3, III-B-1ff

high priority I-6-8

root I-6-8, III-B-1ff

occurrence numbers I-4-2

.OINFOS II-1-3

open procedures II-1-3

DBAM I-5-10, II-5-1ff

FORTRAN III-C-9

ISAM I-4-7, II-4-1ff

RAM I-3-3, II-3-1ff

SAM I-2-4, I-2-7, II-2-1ff

optimized distribution I-5-5

Output mode I-1-3

overwriting (SAM) I-2-12

packet building III-C-1, III-C-4ff

packets II-1-1ff, II-6-1ff

building II-7-1ff

pages I-4-3

size III-B-2

parameters

Assembly language II-7-1ff

FORTRAN user III-C-12ff

PARIU II-7-16ff

partial records I-5-3

suppression I-5-17

partitions I-6-5

peripheral devices I-2-7, I-6-10

permanent file specification (PFS) II-1-4, I-1-7, I-4-3ff

physical volumes I-1-6

.PINFOS II-1-3

pointers, word and byte II-1-1

point function I-2-12, II-2-7

processing packets II-1-2, II-6-27, II-7-10

positioning (labeled mag tapes) III-A-9

Licensed Material - Property of Data General Corporation

pre-open II-1-3

FORTRAN processing routine III-C-9

pre-read function I-3-4

priority, high (nodes) I-6-8

processing, file

disks I-1-7

FORTRAN III-C-10

tapes I-1-7, I-2-12, III-A-9

see also SAM, RAM, ISAM, DBAM

processing modes I-1-2

processing packets

building II-7-8ff

extended II-6-20, II-1-2, II-7-8

general II-1-2, II-6-18, II-7-8

link subindex II-6-28

magnetic tape control II-6-33f

point II-6-27, II-7-10

processing routines (FORTRAN) III-C-2, III-C-9ff

programming your INFOS system II-1-1ff

DBAM II-5-1ff

ISAM II-4-1ff

RAM II-3-1ff

SAM II-2-1ff

pseudo channel number II-1-3

RAM (Random Access Method)

close II-3-5

creation I-3-1f

FEOV I-3-6, II-3-5

Hold I-3-6

lock and SETX I-3-6, II-3-5

opening I-3-3, II-3-1ff

pre-read I-3-5, II-3-6

processing I-3-3, II-3-4ff, II-6-18

read I-3-3, II-3-4

read and write sequence I-3-4

read inhibit I-3-6

write I-3-4, II-3-4

write immediate I-3-5

random allocation I-6-10

DBAM I-5-8ff

ISAM I-4-5

RAM I-3-2

SAM I-2-2

RDOS

and INFOS I-6-1

read-after-write verification I-4-4, I-6-6f

read ahead-write behind technique I-1-6

read function

DBAM I-5-15, II-5-16

ISAM I-4-9, II-4-10

RAM I-3-3, II-3-4

SAM I-2-12, II-2-5

read inhibit I-3-5

records

delimiters I-1-4

formats I-1-3f

packing I-1-5

partial I-5-3

suppress partial I-5-17

- reinstatement function
 - ISAM I-4-10, II-4-12
 - DBAM I-5-16, II-5-21
- relative access/position processing I-4-8, II-4-9
- release (mag tapes) III-A-6ff
- RELX I-2-13
- resident RDOS/INFOS code I-6-2
- retrieve high key
 - ISAM I-4-10, II-4-13
 - DBAM I-5-17, II-5-21
- retrieve key
 - ISAM I-4-10, II-4-13
 - DBAM I-5-17, II-5-21
- retrieve status
 - ISAM I-4-10, II-4-13
 - DBAM I-5-16, II-5-22
- retrieve subindex definition I-5-17
- rewrite function
 - DBAM I-5-16, II-5-18
 - ISAM I-4-9, II-4-11
 - SAM I-2-12, II-2-5
- root nodes I-5-5, I-6-8, III-B-1f
- RPG II, Prefatory Note, Part II
- runtime initialization (labeled mag tapes) III-A-7
- runtime release (labeled mag tapes) III-A-7f

- SAM (Sequential Access Method)
 - appending records I-2-12
 - close request II-2-6
 - creating files I-2-2ff, I-2-4
 - disk files I-2-1
 - file definition options I-2-8f
 - file processing summary I-2-13, II-2-4
 - initializing tape files I-2-4
 - labeled tape files I-2-4, II-2-1, III-A
 - mag tape control II-2-6
 - opening files I-2-4, I-2-7, II-2-1ff
 - overwrite I-2-12
 - point feature I-2-12, II-2-5
 - processing I-2-12f, II-2-4ff, II-6-18
 - reading I-2-12, II-2-5
 - rewrite I-2-12, II-2-5
 - SETX and RELX features I-2-13
 - unlabeled tape files I-2-7
 - using peripheral devices I-2-7
 - volume definition I-2-5, I-2-2, II-2-3f
 - volume definition options I-2-10f
 - writing I-2-12, II-2-5
- selector subindex III-B-7
- sequence numbers (mag tapes) III-A-9
- SETX I-2-13, I-3-5
- single volume files I-1-6
- space management I-4-3
- stacks I-6-2
- status, retrieve
 - DBAM I-5-16, II-5-22
 - ISAM I-4-10, II-4-13
- subindexes
 - define I-5-16, II-5-19
 - definition packets II-1-2, II-6-26, II-7-12
 - delete I-5-16, II-5-21
 - file properties III-B-1ff
 - levels I-5-1, III-B-1ff
 - link I-5-16, II-5-19, I-5-4, II-6-28
 - retrieve definition I-5-17, II-5-22
 - selector III-B-7
- suppress database
 - DBAM I-5-17, II-5-16ff
 - ISAM I-4-10, II-4-10ff
- suppress partial records I-5-17, II-5-16ff
- SYSGEN considerations (system generation) I-6-1
- system
 - area (memory) I-6-2
 - buffers I-6-2ff
 - calls II-1-3f
 - generation I-6-1
 - labels III-A-1
- table building routines (FORTRAN) III-C-1, III-C-8
- tables II-1-2, II-6-25, II-6-17, II-7-15
- tape files - processing I-1-7
- tasking, concurrent I-1-2
- temporary indexes I-5-5
- timeout intervals I-6-11, III-E
- trailer labels III-A-6
- transfers, data I-1-4ff
- translation tables I-1-4
- tree levels III-B-1

- unblocked records
 - see undefined length records
- undefined length records I-1-3
 - transfers I-1-5
- unlabeled magnetic tapes I-2-7, I-6-11
- unlock
 - see lock
- Update mode I-1-2, II-4-5ff, II-5-5
- user area I-6-5
- user labels III-A-6
- user label table
 - Assembly III-C-8
 - FORTRAN III-C-8

- variable length records I-1-3
 - transfers I-1-5
- VCBs (Volume Control Blocks) I-6-3
- VDPs
 - see volume definition packets
- virtual memory I-6-5
- .VL file II-6-17

volumes

control blocks (VCBs) I-6-3
defining I-4-5f, I-2-10f
definition packets I-1-2, II-6-12, II-7-6, III-C-6
force end of I-3-6
initialization packets II-1-2, II-6-31, II-7-14, III-C-6
initializing (mag tape) III-A-6ff;
FORTRAN III-C-10
labels III-A-6, III-A-15ff
logical vs. physical I-1-6
merit factors I-5-8
size I-4-5
tables I-4-4, II-1-2, II-6-17

windows I-6-2f
word pointers II-1-1
write function
 DBAM I-5-15, II-5-17
 ISAM I-4-9, II-4-11
 RAM I-3-4, II-3-4
 SAM I-2-12, II-2-5
write immediate I-3-5, II-3-4

How Do You Like This Manual?

Title _____ No. _____

We wrote the book for you, and naturally we had to make certain assumptions about who you are and how you would use it. Your comments will help us correct our assumptions and improve our manuals. Please take a few minutes to respond.

If you have any comments on the software itself, please contact your Data General representative. If you wish to order manuals, consult the Publications Catalog (012-330).

Who Are You?

- EDP Manager
- Senior System Analyst
- Analyst/Programmer
- Operator
- Other _____

What programming language(s) do you use? _____

How Do You Use This Manual?

(List in order: 1 = Primary use)

- _____ Introduction to the product
- _____ Reference
- _____ Tutorial Text
- _____ Operating Guide
- _____ _____

Do You Like The Manual?

Yes	Somewhat	No	
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is the manual easy to read?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is it easy to understand?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is the topic order easy to follow?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Is the technical information accurate?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Can you easily find what you want?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Do the illustrations help you?
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Does the manual tell you everything you need to know?

Comments?

(Please note page number and paragraph where applicable.)

From:

Name _____ Title _____ Company _____
 Address _____ Date _____

FOLD DOWN

FIRST

FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary if Mailed in the United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Software Documentation

FOLD UP

SECOND

FOLD UP