SC24-5210-0
File No. S370/4300-30

Program Product

VSE/Advanced Functions Macro User's Guide

SC24-5210-0
File No. S370/4300-30

Program Product

VSE/Advanced Functions Macro User's Guide

Program Number 5746-XE9

Release 2

IBM

First Edition (October 1979)

This edition, SC24-5210-0, applies to Release 2 of IBM VSE/Advanced Functions (Program Number 5746-XE9), and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein. Before using this publication in connection with the operation of IBM systems, consult the latest edition of *IBM System/370 and 4300 Processors Bibliography,* GC20-0001 for the editions that are applicable and current.

Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

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

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication; if the form has been removed, comments may be addressed to IBM Laboratory, Publications Department, Schoenaicher Strasse 220, D-7030 Boeblingen, Germany. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

© Copyright International Business Machines Corporation 1979

Summary of Amendments for VSE/Advanced Functions Macro User's Guide

This manual contains information previously published in *DOS/VSE Macro User's Guide,* GC24-5139. Changes reflect support for:

- Sharing of data on DASD
- Device independence for files on disk
- Improved label processing
- Extended multiprogramming and subtask support
- Simplified supervisor assembly
- VSE/VSAM Space Management Feature

announced for Release 2 of VSE/Advanced Functions.

Additional changes include 3262 printer support, APAR corrections, and miscellaneous editorial corrections.

Preface

This book is a guide to programmers using the VSE/Advanced Functions macro instructions (macros). Use of both IOCS (Input/Output Control System) macros and the system control macros is described.

After a brief introduction to the nature and use of macros, the file processing and system control functions are discussed from a conceptual and functional point of view. Included in the discussion of file processing are a chapter on file organization and one on concepts of the IOCS access methods. These serve as a foundation for several following chapters on processing files found on various types of I/O devices, device-independent system files, and the use of PIOCS (Physical IOCS).

Included in the chapters on system control functions are discussions of such topics as virtual storage control, checkpointing, and multitasking.

Several appendixes on a variety of topics, a glossary, and an index complete the book.

As this book is intended as a guide to macro usage, before consulting it, you should be familiar with others that introduce important prerequisite information on the nature and use of IOCS and system control programs:

- *Introduction to DOS/VSE*, GC33-6108
- *IBM System/370 Principles of Operation*, GA22-7000
- *IBM 4300 Processors Principles of Operation*, GA22-7070
- *VSE/Advanced Functions System Management Guide*, SC33-6094
- *VSE Systems Data Management Concepts*, GC24-5209

In addition, you should be familiar with the device manuals for those devices that you plan to use, such as:

- *DOS/VS IBM 3800 Printing Subsystem Programmer's Guide*, GC26-3900

System publications related to this one and referred to in this book:

- *VSE/Advanced Functions Macro Reference*, SC24-5211
- *VSE/Advanced Functions Tape Labels*, GC24-5212
- *VSE/Advanced Functions DASD Labels*, GC24-5213
- *OS/VS-DOS/VSE-VM/370 Assembler Language*, GC33-4010
- *VSE/Advanced Functions System Control Statements*, SC33-6095
- *VSE/Advanced Functions System Generation*, SC33-6096
- *VSE/Advanced Functions Operating Procedures*, SC33-6097
- *VSE/Advanced Functions Serviceability Aids and Debugging Procedures*, SC33-6099

In addition, see the following for information on IBM Program Products:

- *VSE/VSAM General Information*, GC24-5143
- *DL/I DOS/VS General Information*, GH20-1246

These and other VSE/Advanced Functions publications are described in the *IBM System/370 and 4300 Processors Bibliography*, GC20-0001. Terminology is defined in the *Data Processing Glossary*, GC20-1699.

Contents

Chapter 1: Macro Types and Their Use 1-1
Source-Program Macros 1-1
IOCS Macros 1-1
Control Program Function Macros 1-1
Macro Processing 1-1
DTF Declarative Macros 1-1
DTF Table 1-2
Symbolic Unit Addresses in the DTFxx Macro 1-3
Logic Module Generation Macros 1-4
Module Names 1-5
Interrelationship of the Macros 1-6
Link-Editing Logical IOCS Programs 1-6
Self-Relocating Programs and IOCS 1-7
Using DASD Support 1-7
Register Usage 1-8
Registers for VSE/Advanced Functions Use 1-8
Registers for Your Use 1-8
Macro Format 1-8
Notational Conventions 1-9
Declarative Macro Statements 1-10

Chapter 2: File Organization 2-1
Sequential Organization without Index 2-1
Direct Organization 2-1
Sequential Organization with Index 2-3
Prime Data Area 2-4
Indexes 2-4
Overflow Area 2-7
Example of an ISAM File 2-8

Chapter 3: Access Methods Concepts 3-1
SAM (Sequential Access Method) 3-1
Control Interval Format 3-1
Defining Files and Functions 3-2
Activating a File for Processing 3-3
Processing Data Files with SAM 3-4
Processing Work Files with SAM 3-12
Deactivating a File After Processing 3-16
Non-Data Device Operations 3-17
Logic Modules for SAM 3-17
DAM (Direct Access Method) 3-18
Record Types 3-19
I/O Area Specification 3-19
Creating a File or Adding Records 3-21
Locating Data: Reference Methods 3-21
Locating Free Space 3-24
Logic Modules for DAM 3-24
ISAM (Indexed Sequential Access Method) 3-24
Record Types 3-26
Storage Areas 3-26
Activating (Opening) a File for Processing 3-26
Processing an ISAM File 3-26
Deactivating (Closing) an ISAM File After Processing 3-30
Reorganizing an ISAM File 3-30
Logic Modules for ISAM 3-31
PIOCS (Physical IOCS) 3-33

Chapter 4: Processing DASD Files 4-1
DASD Capacities 4-1
Processing with SAM 4-2
FBA (Fixed Block Architecture) DASD Processing 4-3
Opening a File 4-3
Label Processing 4-5

Error Handling 4-6
Deactivating a Sequential DASD File 4-8
Processing with DAM 4-8
Initialization 4-9
Processing 4-10
Completion 4-22
Processing with ISAM 4-22
DTFIS Operands for I/O Area Specification 4-22
Initialization 4-24
Processing 4-26
Completion 4-33
Programming Considerations 4-34
ISAM Disk Storage Space Formulas 4-35

Chapter 5: Processing Diskette Files 5-1
Opening a File 5-1
Output 5-1
Input 5-2
Processing Records with Command Chaining 5-2
Closing a File 5-3
Error Handling 5-3

Chapter 6: Processing Magnetic Tape Files 6-1
Label Processing 6-1
Block Size 6-1
Reading Magnetic Tape Backwards 6-1
Forcing End-Of-Volume 6-1
Error Handling 6-4
Programming Your Error Processing Routines 6-5
Non-Data Device Operations 6-6

Chapter 7: Processing Unit Record Files 7-1
Processing Punched Card Files 7-1
Associated Files 7-1
OMR Data 7-3
Updating Records 7-4
End-of-File Handling 7-5
Error Handling 7-6
Programming Considerations 7-6
Card Device and Printer Control 7-7
GET/CNTRL/PUT Sequence for Associated Files 7-10
Processing Printer Files 7-10
Associated Files 7-10
Printer Overflow 7-10
Printer Control 7-11
Error Handling 7-14
Processing Console Files 7-14
Programming Considerations 7-15
Processing Magnetic Reader Files 7-15
Characteristics of Magnetic Ink Character Reader (MICR) 7-15
Programming Considerations 7-18
Processing Optical Reader Files 7-20
Non-Data Device Operations 7-24
Programming Considerations for Optical Readers 7-27
Processing Paper Tape Files 7-35
Programming Considerations for Paper Tape 7-35

Chapter 8: Processing Device-Independent System Files 8-1
Record Size 8-1
Error Handling 8-2
End-Of-File Handling 8-3

Chapter 9: Processing Files with PIOCS (Physical IOCS) 9-1
Initialization 9-1
Processing 9-3
Processing Labels and Extents 9-8
Forcing End-of-Volume 9-9
Termination 9-9
PIOCS Programming Considerations 9-10
Situations Requiring LIOCS Functions in PIOCS Processing 9-10
Command Chaining Retry 9-10
Data Chaining 9-11
CKD DASD Channel Programs 9-11
RPS (Rotational Position Sensing) 9-11
Channel Programs for FBA Devices 9-11
Diskette Channel Programs 9-12
Console (Printer-Keyboard) Buffering 9-12
Alternate Tape Switching 9-12
Bypassing Embedded Checkpoint Records on Magnetic Tape 9-13

Chapter 10: Requesting Control Functions 10-1
Program Loading 10-1
Shared Virtual Area Considerations for Program Load Macros 10-1
Fast Loading of Frequently Used Phases 10-1
Virtual Storage Control 10-2
Fixing and Freeing Pages in Real Storage 10-2
Determining the Execution Mode of a Program 10-3
Extracting Partition-Related Information 10-3
Influencing the Paging Mechanism 10-3
Dynamic Allocation of Virtual Storage 10-4
Program Communication 10-4
Assigning and Releasing I/O Units 10-5
Assigning and Releasing Tape Drives 10-6
Timer Services and Exit Control 10-6
Timer Services 10-6
Linkages to User Exit Routines 10-8
Requesting Storage Dumps 10-10
Ending a Task or a Job 10-11
Normal End of the Main Task 10-11
Normal End of a Subtask 10-12
Program-Requested Abnormal Ends 10-12
Using the EXIT Macro 10-12
Checkpointing a Program 10-12
Choosing a Checkpoint 10-12
Timing the Entry to the Checkpoint Routine 10-12
Saving Data for Restart 10-14
Restarting a Checkpointed Program 10-14
Checkpoint File 10-14
Repositioning I/O Files 10-15
Program Linkage 10-16
Linkage Registers 10-19
Save Areas 10-19
CALL, SAVE, RETURN Macros 10-20
Multitasking Functions 10-21
Subtask Initiation 10-21
Subtask Termination 10-24
Resource Protection 10-24
Resource-Share Control 10-27
Intertask Communication 10-28
DASD Record Protection (Track Hold) 10-29
Shared Modules and Files 10-32
Loading a Forms Control Buffer 10-33
Requesting System Information 10-34

Appendix A: Control Character Codes A-1

Appendix B: Assembling Your Program, DTF's, and Logic Modules B-1
Comparison of the Five Methods B-12

RPS Example B-15
FBA DASD Example B-18

Appendix C: Label Processing C-1
DASD Standard Labels C-1
OPEN Macro Processing C-1
End-of-Volume Processing C-1
End-of-File Processing C-1
User Standard Labels C-2
Diskette Labels C-3
OPEN Macro Processing C-3
End-of-Volume Processing C-3
End-of-File Processing C-3
Tape Labels C-3
Tape Output Files C-3
Tape Input Files C-5
Reading, Writing, and Checking with Nonstandard Labels C-8

Appendix D: Writing Self-Relocating Programs D-1
Rules for Writing Self-Relocating Programs D-1
Programming Techniques D-2

Appendix E: American National Standard Code for Information Interchange (ASCII) E-1

Appendix F: Page Fault Handling Overlap F-1
Register Usage F-1
Entry Linkage F-1
Page Fault Queue F-1
Processing at the Initiation of a Page Fault F-1
Processing at the Completion of a Page Fault F-2

Appendix G: Using System Control Macros in Reenterable Programs G-1

Index I-1

Chapter 1: Macro Types and Their Use

A *macro* is a single assembler language instruction that generates a sequence of assembler language instructions. The macros you code in your program are called the *source program macros.* The assembler uses what is called the *macro definition* to generate the sequence of instructions requested by the source program macro. Use of macros simplifies the coding of programs and reduces the possibility of programming errors.

A macro definition is a set of statements that defines the name and format of and the conditions for generating a sequence of assembler language instructions from a single macro instruction. Macro definitions are stored in the macro sublibrary of the source statement library.

Source-Program Macros

Source-program macros are those you specify in your program; they indicate to the assembler which macro definition is to be called from the library. With a source-program macro you specify operands and parameters which the assembler uses, together with the called macro definition, to determine what assembler instructions to generate. There are two different types of source-program macros: logical IOCS (input/output control system) macros, and control program function macros.

IOCS Macros

IOCS macros are divided into two basic categories: imperative IOCS macros and declarative IOCS macros.

Imperative IOCS Macros

These macros identify what I/O operation you want to perform. The GET macro, for example, indicates that you want to obtain a record.

Declarative IOCS Macros

Declarative IOCS macros for all basic access methods are of two related types -- DTFxx macros and logic module generation (xxMOD) macros. The DTF macros Define The File for the various access methods and I/O devices. The logic module generation macros define the logic modules that will handle the conditions that you specify in the macro.

VSE/VSAM Macros: The Virtual Storage Access Method (VSE/VSAM) has a set of declarative macros different from the DTFxx and logic module generation macros described above.

The VSE/VSAM macros are discussed briefly in *VSE System Data Management Concepts* and are fully described in VSE/VSAM publications.

Control Program Function Macros

These macros, which are frequently referred to as *supervisor macros,* enable you to make use of functions performed by programs and routines of VSE/Advanced Functions. The RUNMODE macro, for example, determines whether your program runs in virtual or real mode.

Macro Processing

A direct relationship exists between the source-program macros and the macro definitions.

During assembly, the source-program macro specifies which macro definition is to be called from the library. Figure 1-1 depicts schematically the source program before and after inclusion of the macro expansion. This is accomplished by a selection and substitution process using the general information in the macro definition and the specific information in the source-program macro. The insertion consists of a module, a table, or a small in-line routine and is called the macro expansion.

After the insertion is made, the complete program consists of both source program statements and assembler language statements generated from the macro definition. In subsequent phases of the assembly, the entire program is processed to produce the machine-language program.

IBM provides a number of tested macro definitions. The source-program macros needed to use these definitions are described in this manual. You can also write your own macro definitions and include them in your source statement library. For additional information on this subject, see *OS/VS-DOS/VSE-VM/370 Assembler Language,* as listed in the Preface.

DTF Declarative Macros

All basic access methods require a DTF declarative macro to be coded for each file that your program wants to access by means of logical IOCS imperative macros such as GET, PUT, READ, WRITE, CNTRL. The DTF (define the file) macro describes the characteristics of the file, indicates the type of processing for the file, and specifies the virtual storage areas and routines to be used in processing the file.

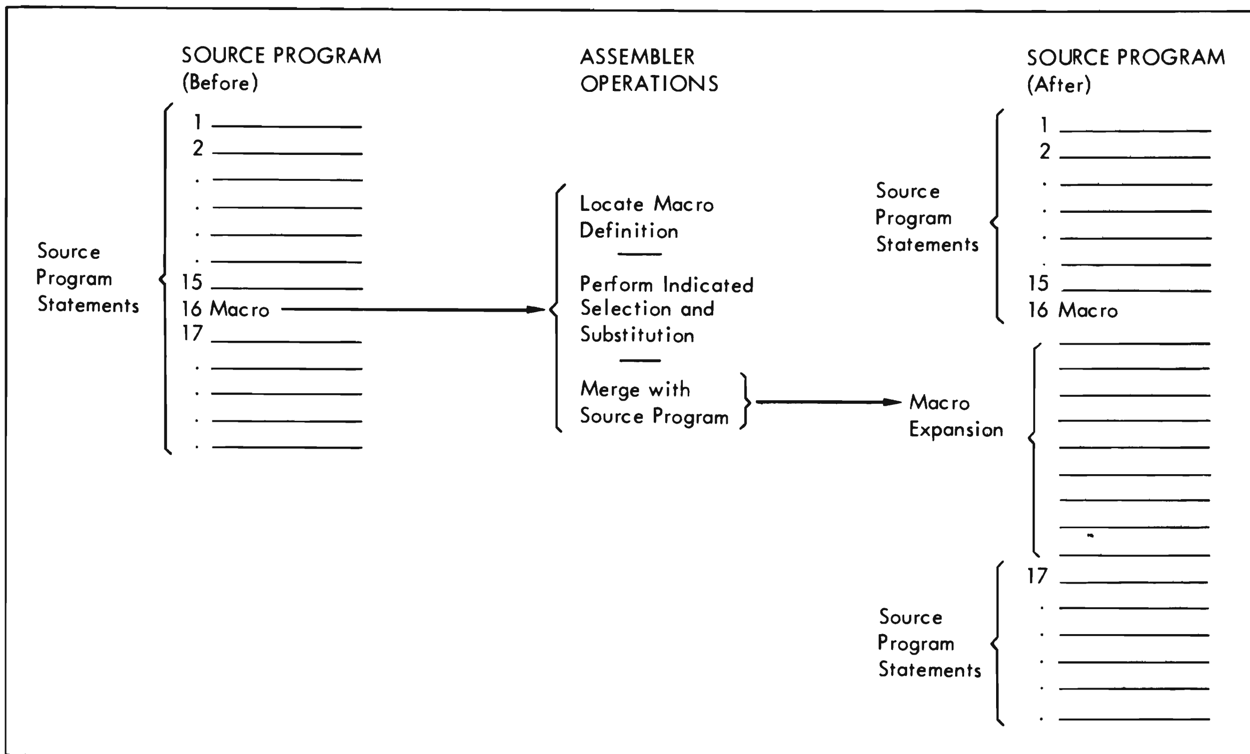


Figure 1-1. Schematic of macro processing.

For example, if a GET is issued, the DTF macro supplies such information as:

- Record type and length.
- Input device from which the record is to be retrieved.
- Address of the area in storage where the record is to be located for processing by your program.

Device-oriented DTF macros are available for defining files processed by LIOCS (Logical Input/Output Control System). An additional DTF macro is available for magnetic tape or DASD files processed by PIOCS (Physical Input/Output Control System). Figure 1-2 contains an example of a DTF source statement. For LIOCS operations, the DTF macro used depends on the type of processing that will be performed and upon the type of device used.

Processing with SAM: Applies to input/output with serial or diskette devices, or with direct access devices when records are processed sequentially. (ISAM may also be used with direct access devices when records are to be processed sequentially). The DTF macros used for SAM processing are listed by device name in alphabetical order in Figure 1-3.

Processing with DAM: Whenever a file on a direct access device is to be processed by DAM, DTFDA must be used.

Processing with ISAM: Whenever a file on a direct access device is to be organized or processed by ISAM, DTFIS must be used.

Processing with PIOCS: When PIOCS macros (EXCP, WAIT, etc.) are used for a file, the DTFPH macro is required only if standard labels are to be checked or written on a file on a direct access device, magnetic tape or diskette, or if the file on a direct access device is file-protected.

DTF Table

A DTFxx macro generates a DTF table that contains indicators and constants describing the file. You can reference this table by using the symbol *filename+constant,* or *filenamex* where *x* is a letter. When referencing the DTF table, you must ensure addressability through the use of an A-type constant, or through reference to a base register. Should you need to reference a DTF table in your program (for example, to test error information in the CCB, which is contained in the table), you can obtain detailed information on the layout of DTF tables in the VSE/Advanced Functions Diagnosis Reference: LIOCS manuals.

```
                                                  Column 72
                                                          ↓
OLDMSTR  DTFMT                                            X
                BLKSIZE=400,                              X
                DEVADDR=SYS001,                           X
                EOFADDR=EOFMSTR,                          X
                FILABL=STD,                               X
                IOAREA1=AREAONE,                          X
                ERROPT=CKOLDBLK,                          X
                HDRINFO=YES,                              X
                IOAREA2=AREATWO,                          X
                IOREG=(3),                                X
                LABADDR=CKOLDBLK,                         X
                READ=FORWARD,                             X
                RECFORM=FIXBLK,                           X
                RECSIZE=80,                               X
                REWIND=UNLOAD,                            X
                SEPASMB=YES,                              X
                TYPEFLE=INPUT,                            X
                WLRERR=REG6
```

Figure 1-2. Sample DTFMT macro.

File to be processed on	Macro
Card device	DTFCD
Console printer-keyboard	DTFCN
DASD sequential	DTFSD
Device independent	DTFDI
Diskette I/O Unit	DTFDU
Display operator console	DTFCN
Magnetic reader (MICR)	DTFMR
Magnetic tape	DTFMT
Optical reader (excluding 3886)	DTFOR
3886 Optical character reader	DTFDR
Optical reader/sorter	DTFMR
Paper tape device	DTFPT
Printer	DTFPR
Sequential DASD	DTFSD

Figure 1-3. SAM declarative macros.

Symbolic Unit Addresses in the DTFxx Macro

In most of the DTF macros you can specify a symbolic unit name in the DEVADDR operand. This symbolic unit name is also used in the ASSGN job control statement to assign an actual I/O device address to the file. For files on diskettes or direct access devices, the symbolic unit name is supplied in the DEVADDR operand and/or with the EXTENT job control statement (if both are provided, the EXTENT specification overrides the DEVADDR specification).

The symbolic unit name of a device is chosen from a fixed set of symbolic names. Programs are written considering only the device type (tape, card, etc.). At execution time, the actual physical device is determined and assigned to a given symbolic unit. For instance, a program that processes tape records can call the tape device SYS000. At execution time the operator (using ASSGN) assigns any available tape drive to SYS000.

Figure 1-4 shows the relationship between the source program, the DTF table, and the job control I/O assignment.

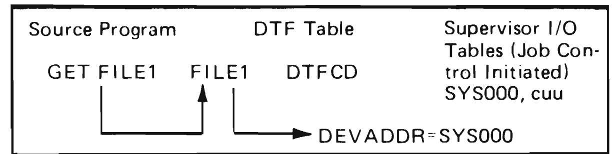


Figure 1-4. Relationship between source program, DTF table, and job control I/O assignment.

The fixed set of symbolic names that can be used with a DTF macro for a program in any partition is the same and is represented by SYSxxx. Programs in different partitions can reference the same logical unit as long as different devices, or DASD extents, are assigned.

These symbolic units are divided into system logical units and programmer logical units.

System Logical Units

SYSRES System residence extent.

SYSRDR Card reader, magnetic tape unit, disk extent, or diskette extent primarily for job control statements.

SYSIPT Card reader, magnetic tape unit, disk extent, or diskette extent as the primary input unit for programs.

SYSPCH Card punch, magnetic tape unit, disk extent, or diskette extent as the primary unit for punched output.

SYSLST Printer, magnetic tape unit, disk extent, or diskette extent as the primary unit for printed output.

SYSLOG Console printer-keyboard or display operator console for operator messages and for logging job control statements. Can also be assigned to a printer.

SYSLNK Disk extent as input to the linkage editor.

SYSCTL Used by VSE/Advanced Functions at IPL time to load the buffer(s) of FCB-type pointers.

SYSCAT Disk extent for the VSE/VSAM catalog.

SYSCLB Disk extent for a private core image library.

SYSRLB Disk extent for a private relocatable library.

SYSUSE Disk extent used by the system for internal purposes, only.

SYSSLB Disk extent for a private source statement library.

SYSREC Disk extent for error log records, program history entries, and for the hard copy file of the display operator console.

SYSDMP Disk extent used to receive high speed system dump.

SYSIN **Can** be used if you want to assign SYSRDR and SYSIPT to the same card reader or magnetic tape unit. **Must** be used if you want to assign SYSRDR and SYSIPT to the same disk extent.

SYSOUT **Must** be used if you want to assign SYSPCH and SYSLST to the same magnetic tape unit. **Cannot** be used to assign SYSPCH and SYSLST to disk because these two units must refer to separate disk extents.

Note: Of these system logical units, user programs may use SYSIPT and SYSRDR for input, SYSLST and SYSPCH for output, and SYSLOG for communication with the operator. However, other system logical units must not be used in place of programmer logical units (that is, within user programs or EXTENT statements). For instance, SYSIN and SYSOUT are valid only to job control and cannot be referenced in a user program. Examples for the use of SYSIN and SYSOUT are given in in the *VSE/Advanced Functions System Management Guide.*

Programmer Logical Units

SYSnnn SYSnnn represents all the other symbolic units that can be used under VSE. These units range from SYS000 to SYS254.

Each of these programmer logical units can be assigned to any partition without a prescribed sequence, except when using DAM (see *Note,* below).

Note: For DAM the EXTENT job control statements must be supplied in ascending order, and the symbolic units for multivolume files must be assigned in consecutive order.

Each declarative macro requiring a symbolic unit to be specified has a list of symbolic units that are valid for that macro. In that list, SYSnnn represents programmer logical units, while SYSxxx indicates either a system or a programmer logical unit.

For files processed by either SAM or DAM, only one symbolic unit may be assigned to all extents of a file on one volume.

In physical IOCS, the symbolic unit name is specified in the CCB or IORB and in the DTFPH macro. Instead, or additionally, it is specified with the EXTENT job control statement. (If more than one of these is used to provide the specification, an EXTENT specification overrides a DTFPH specification, and a CCB specification overrides an EXTENT and/or a DTFPH specification.)

Figure 1-5 shows the relationship between the source program and the job control I/O assignment.

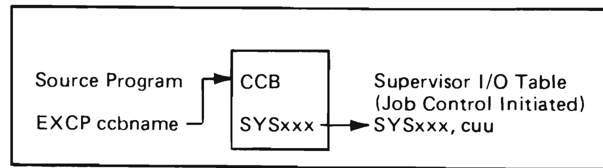


Figure 1-5. Relationship between source program and job control I/O assignment.

Logic Module Generation Macros

Each DTF, except DTFCN and DTFPH, must link to an IOCS logic module. More than one DTF can be linked to the same logic module.

A logic module is generated by a logic module generation (xxMOD) macro. The modules provide the necessary instructions to perform the input/output functions required by your program. For example, a module reads or writes data, tests for unusual input/output conditions, blocks or deblocks records if necessary, or places records in a work area. Most imperative macros use a logic module to perform the requested function.

For DTFSD, DTFDA, DTFDI DASD files, the system provides the logic modules. Any logic modules provided by the user for these DASD files are ignored by OPEN.

Providing Logic Modules

Logic modules are provided for your DTFs in three ways:

1. Code the logic module generation macros needed by your DTFs, assembling them either in-line with your program or supplying them at link-edit time.
2. If the standard logic modules needed for your installation were assembled and catalogued in the relocatable library at system generation time, you need not code them in your program. Instead, you can autolink the necessary modules from the relocatable library at link-edit time.
3. You need not code logic modules for certain I/O devices. Support for DASD I/O devices (except for ISAM files on DASDs without RPS) and the IBM 3800 Printing Subsystem includes preassembled logic modules. These logic modules are automatically loaded into the SVA (system virtual area) at IPL time and linked to the problem program during OPEN processing for the DTF.

Keeping Modules Small

Some of the module functions are provided on a selective basis, according to the parameters specified in the xxMOD macro. If you code the xxMOD macro yourself, you have the option of selecting or omitting some of these functions according to the requirements of your program. If, as described above, you do not code the xxMOD macro yourself, IOCS automatically selects or omits the appropriate functions. In either case the omission of unneeded functions saves storage space for a particular module.

Note: If you issue an imperative macro, such as WRITE or PUT, to a module that does not contain that function, then an invalid supervisor call (SVC 50) is generated, the job is terminated, and a message is displayed.

Subsetting/Supersetting

Some modules may be subset modules to a superset module. A superset module is one which performs all the functions of its subset or component modules, avoiding duplication and thereby saving storage space. The functions required by several similiar DTFs (that is, several DTFCDs, or several DTFPRs, etc.) are thus available via a single xxMOD macro, even if the DTFs have slightly different parameters. An example is shown in Figure 1-6.

If you do not code the logic modules yourself, IOCS automatically performs all subsetting and supersetting that is possible.

If you code the logic modules yourself, subsetting/supersetting can be achieved by coding a single xxMOD macro containing all of the functions needed by all of the DTFs which use that macro. In this case you may either:

- Not name the module and let IOCS name it for you - that is, specify no name for the xxMOD macro and also no MODNAME operands in the DTFs; or
- Name the module, specifying a name for the xxMOD macro and also specifying the same name in the MODNAME operands of all the DTFs which will use that module.

Subsetting/supersetting cannot be performed if you supply an xxMOD macro for each DTF of a given device type. In this case:

- If you did not name the modules, the assembler program will detect a double declaration error condition, or
- If you did name the modules, they will be generated without any subsetting/supersetting.

Superset Module Functions	Subset Module Functions	Subset Module Functions
Optional use of CNTRL macro	CNTRL macro cannot be used	Optional use of CNTRL macro
Workarea and I/O area processing	Workarea and I/O area processing	I/O area processing only
Support of printer overflow	No printer overflow support	Support of printer overflow
Support of user-specified error actions	Support of user-specified error actions	Support of user-specified error actions

Figure 1-6. Subset and superset module example.

Module Names

As mentioned under "Logic Module Generation Macros," you can have IOCS provide a name for the required logic module, or you can specify that name. Both methods are discussed below.

IOCS Supplies the Name

In order to make use of this facility omit the MODNAME operand from the DTF macro; the IOCS macro will then generate a standard module name as determined by the functions required by the DTF.

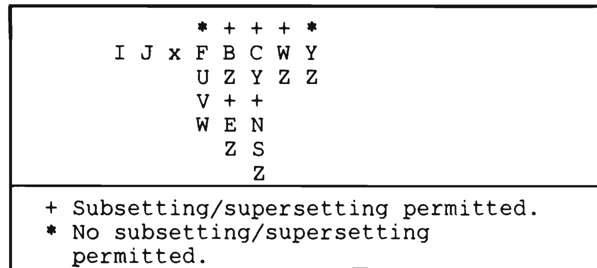
Likewise, if you code your own module, the name should be omitted from the name field, and IOCS will generate a standard module name matching that referenced in the DTF.

Standard module names used by IOCS are given under "Standard CDMOD Names," "Standard DIMOD Names," etc., following the discussing of the appropriate xxMOD macro in *VSE/Advanced Functions Macro Reference.*

For DTFSD, DTFDA, and DTFDI DASD files, the module name, if supplied with the macro, is overridden by OPEN. IBM-supplied superset modules are used for these files. These fully reentrant modules are loaded into the SVA during IPL. One copy of a specific logic module is used for all requestors for the type of file handled by that logic module.

IOCS Subset/Superset Names: The supersetting/subsetting described below does not apply to DTFSD, DTFDA, or DTFDI DASD files. Any user specification is ignored because the logic modules are supplied by the system. OPEN determines the appropriate file type and provides the linkage to the selected module (resident in the SVA). IOCS performs subsetting/supersetting of modules with standard module names by collecting the services required by the DTFs and generating a single module

with different entry points corresponding to the standard module names. If you are interested in seeing how IOCS forms subset/superset names, charts showing the name-building conventions are given throughout the book for the various logic modules under "Subset/Superset CDMOD Names," "Subset/Superset DIMOD Names," etc., following the discussion of the appropriate module in *VSE/Advanced Functions Macro Reference.* Figure 1-7 shows a model for these charts:



```
        * + + + *
  I J x F B C W Y
        U Z Y Z Z
        V + +
        W E N
          Z S
            Z
```

+ Subsetting/supersetting permitted.
* No subsetting/supersetting permitted.

Figure 1-7. Model for a subset/superset naming chart.

The letters indicate functions which can be performed by the logic module (these are fixed for a given module and are explained in the sections "Standard CDMOD Names," "Standard DIMOD Names," etc.). If a module name were composed of letters from the top row exclusively, it could only be a superset name; and names including letters from the second or lower rows would then be subset names to the top-row superset name. For example, the module IJxWESZZ is a subset module to superset module IJxWENZZ. IJxWEZZZ is another subset module to superset module IJxWENZZ. Similarly, IJxWEZZZ is also a subset module to superset module IJxWESZZ.

An asterisk (*) over a column indicates that no subsetting or supersetting is permitted, while a plus (+) sign in a column indicates that both are permitted. Two plus signs in a single column divide that column into mutually exclusive sets. In this example, C is not a superset of N, S, or Z, and conversely, N, S, or Z is not a subset of C.

The vertical arrangement of letters within a column is always in alphabetical order. If a column is divided by plus and/or asterisk signs into sets, then the vertical arrangement of letters within each set of a column is in alphabetical order.

You Supply the Name

For DTFs other than DTFSD, DTFDA, or DTFDI files assigned to DASD devices, specify the logic module name in the MODNAME operand. A module with this name must otherwise be present in your program, or be supplied to your program when it is link-edited. Subsetting/supersetting will occur if one module contains all of the functions needed by all of the DTFs which will use the module (all must reference it by the same name).

Nothing is gained by giving your modules standard IOCS names (see "IOCS Supplies the Name," above), for IOCS will supply the same name for you if you let it name the modules. Should you decide to name your modules, use names which are meaningful to you in the context of your program.

Interrelationship of the Macros

Figure 1-8 shows the relationship between the program, the DTF, and the logic module. Imperative macros initiate the action to be performed by branching to the logic module entry point generated in the DTF table. TAPE is the name of the file. IJFFBCWZ is the name of the logic module.

Linkage between the program, DTF, and logic module is accomplished by the assembler and the linkage editor.

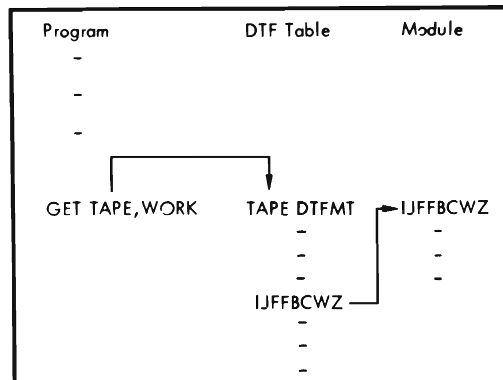


Figure 1-8. Relationship between program, DTF, and logic module.

Link-Editing Logical IOCS Programs

You have the option of assembling your DTFs, and any logic modules which you code yourself, either with your main program or separately for later link-editing with the main program. These possibilities are discussed below and are illustrated in Appendix B.

Program, DTF, and Logic Module Assembled Together

If you assemble DTFs and logic modules with the main program, the linkage editor searches the input stream and resolves the symbolic linkages between tables and modules. This is accomplished by

external-reference information (V-type address constants generated in DTF tables) and the control section definition information (CSECT definitions in logic modules). Further information on link-editing can be found in the section "Linkage Editor" of *VSE/Advanced Functions System Control Statements.*

Program, DTF, and Logic Module Assembled Separately

Specify the operand SEPASMB=YES in the DTF macro or xxMOD macro which is to be separately assembled. For DTFs which are separately assembled, there are some symbolic linkages which you must define yourself in the form of EXTRN and ENTRY symbols. See Appendix B for a full description of which symbolic linkages you must define yourself.

Supplying the SEPASMB=YES operand in a DTF macro causes a CATALR card with the filename to be punched ahead of the object deck and defines the filename as an ENTRY point in the assembly. Specifying the SEPASMB=YES operand in an xxMOD macro causes a CATALR card with the module name to be punched ahead of the object deck and defines the module name as an ENTRY point in the assembly. In either case, a START card must not be used in a separate assembly.

Using the Relocatable Library

As stated earlier, considerable coding effort is saved if logic modules are cataloged in the relocatable library. The same applies to DTFs. Using DTFs cataloged in the relocatable library requires that you take care in naming the DTFs - that is, that you develop a set of standard names and then use them both for your DTFs and in all references your program makes to the DTFs. However, should you decide to name modules yourself, instead of letting IOCS do it, then make sure that you refer to precisely those modules in your DTFs by using their exact names (see "Module Names" above).

If, during generation of your system, a standard set of logic modules needed by the installation has been generated, autolinking the appropriate modules to your DTFs presents no problem. This is particularly true if both the modules and DTF references to them use standard module names.

Using logic modules which you named yourself - as opposed to those named by IOCS - cataloged in the relocatable library requires care. You should verify that the desired modules have been cataloged in the library by consulting a DSERV listing of the library. The linkage editor can perform an autolink only if there is an exact match of module names specified in the DTFs and the names of the modules themselves.

Self-Relocating Programs and IOCS

The Relocating Load feature, standard in VSE (a system generation option in DOS/VS) makes it unnecessary for you to write your own self-relocating programs. If, however, you want to make IOCS imperative macros and control program function macros self-relocating you must do the following:

1. Use the OPENR and CLOSER macro.
2. Use register notation within all your imperative and control program function macros (see "Register Notation" later in this chapter, and "Appendix D. Writing Self-Relocating Programs").

Using DASD Support

For DTFSD, DTFDA, DTFDI files, and ISAM files residing on RPS DASD devices, the DTF in your problem program is linked to a logic module in the shared virtual area (SVA). This linkage is established during open processing of the DTF. The logic modules reside in the core image library and are loaded into the SVA after IPL (or, for ISAM files on RPS DASD devices only, during open processing).

Logic Modules for DTFSD, DTFDA, and DTFDI DASD files are supplied and used as described under "IOCS Supplies the Name" earlier in this chapter.

The following prerequisites must be met before the DTF will be linked to an ISAM RPS or DASD device independent logic module:

1. The DASD must have the RPS feature if an RPS logic module is required.
2. The supervisor must be generated with the RPS=YES option specified (for RPS, only).
3. The modules must be present in the core image library. Logic modules for DTFSD, DTFDA, and DTFDI DASD files are supplied with the system.
4. The logic module must be placed in the SVA when the SVA is being built, or sufficient GETVIS area in the SVA must be available during the open processing to dynamically load the module (for ISAM files on RPS DASD devices, only).
5. Sufficient partition GETVIS area must be available to allow dynamic generation of an extension to the DTF to contain ISAM RPS or DASD device independent channel programs and work areas (partition GETVIS area is made available with the SIZE parameter on the EXEC job control statement).

Since the DTF is linked to a logic module in the SVA, no logic module need to be included with the problem program. However, if the above RPS support prerequisites are not met for ISAM files, open processing bypasses the RPS module linkage and opens the DTF as if a standard non-RPS module linkage exists within the problem program.

Register Usage

Registers for VSE/Advanced Functions Use

General registers 0, 1, 13, 14, and 15 have special uses, and are available to your program only under certain conditions.

The following paragraphs describe the general uses of these registers by IOCS, but the description is not meant to be all inclusive. For more information on subroutine linkage through registers, refer to the "Linkage Registers" under "Requesting Control Functions". In addition, special applications, such as a MICR stacker selection routine, may require different registers.

Registers 0 and 1

Logical IOCS macros, the control program function macros, and other IBM-supplied macros use these registers to pass parameters. Therefore, these registers may be used without restriction only for immediate computations. However, if you use these registers for computations not completed before the system requires them, you must save their contents and reload them later when required.

Register 13

Control program subroutines, including logical IOCS, use this register as a pointer to a 72-byte, doubleword aligned save area. When using the CALL, SAVE, or RETURN macros you can set the address of the save area at the begining of each program phase, and leave it unchanged thereafter. However, when sharing a reentrant (read only) logic module among tasks, each time that module is entered by another task, register 13 must contain the address of another 72-byte save area to be used by that logic module.

Registers 14 and 15

Logical IOCS uses these two registers for linkage. Register 14 contains the return address (to the program) from called programs, DTF routines, and your subroutines. Register 15 contains the entry point into these routines and is used as a base register by the OPEN, CLOSE, and certain DTF macros.

IOCS does not save the contents of these registers before using them. If you use these registers, you must either save their contents yourself (and reload them later) or finish with them before IOCS uses them. Note also that not all logic modules use standard save area conventions. As a result, if you use a read-only logic module (supplying a module savearea) in a subroutine, the savearea back-chain pointer can be lost.

Registers for Your Use

Registers 2 through 12 are available for general usage. There are, however, a few restrictions.

The assembler instructions for the TRT (TRanslate and Test) and the PUTR (PUT with Reply) macros make special use of register 2. It is your responsibility to save the contents of this register for later use in your program if the register contains valuable information (such as pointers or counters) before the TRT or PUTR is executed. After they have been executed, you can then restore the contents of register 2.

If an ISMOD logic module precedes a USING statement or follows your program, the use of registers 2 through 12 remains unrestricted even at assembly time. However, if the ISMOD logic module lies within the problem program, you should issue the same USING statement (which was issued before the logic module) directly following the logic module. This action is necessary because the ISMOD CORDATA logic module uses registers 1 through 3 as base registers. Each time either module is assembled, these registers are dropped.

Macro Format

Macros, like assembler statements, have a name field, operation field, and operand field. Comments can also be included as in assembler statements, although certain macros require a comment to be preceded by a comma if the macro is issued without an operand. These macros are CANCEL, DETACH, FREEVIS, GETIME, GETVIS, TESTT, and TTIMER.

The *name field* in a macro may contain a symbolic name. Some macros require a name; for example, CCB, TECB, DTFxx.

The *operation field* must contain the mnemonic operation code of the macro.

The operand in the *operand field* must be written in either positional, keyword, or mixed formats.

Positional Operands

In this format, the parameter values must be in the exact order shown in the individual macro discussion. Each operand, except the last, must be followed by a comma, and no embedded blanks are allowed. If an operand is to be omitted in the macro

and following operands are included, a comma must be inserted to indicate the omission. No commas need to be included after the last operand. Column 72 must contain a continuation character (any non-blank character) if the operands fill the operand field and overflow onto another line.

For example, GET uses the positional format. A GET for a file named CDFILE using WORK as a work area is written:

```
GET CDFILE, WORK
```

Keyword Operands

An operand written in keyword format has this form, for example:

```
LABADDR=MYLABELS
```

where LABADDR is the keyword, MYLABELS is the specification or value, and LABADDR=MYLABELS is the complete operand.

The keyword operands in the macro may appear in any order, and any that are not required may be omitted. Different keyword operands may be written in the same line, each followed by a comma except for the last operand of the macro. Or they may be written in separate lines as shown in Figure 1-2.

Mixed Format

The operand list contains both positional and keyword operands. The keyword operands can be written in any order, but they must be written to the right of any positional operand in the macro.

For additional information on coding macro statements, see *OS/VS-DOS/VSE-VM/370 Assembler Language,* as listed in the Preface.

Notational Conventions

The following conventions are used in this book to illustrate the format of macros:

1. Uppercase letters and punctuation marks (except as described in these conventions) represent information that must be coded exactly as shown.
2. Lowercase letters and terms represent information which you must supply. More specifically, an n indicates a decimal number, an r indicates a decimal register number, and an x indicates an alphameric character.
3. Information contained within brackets [] represents an optional parameter that can be included or omitted, depending on the requirements of the program.
4. Stacked options contained within brackets represent alternatives, one of which can be chosen, for example:

Options	Description
[name label address]	A name-field symbol in this assembly, or an operand of an EXTRN statement, or * (the location counter).

5. Stacked options contained within braces {} represent alternatives, one of which must be chosen.
6. Items 4 and 5 above may also be shown between brackets and braces, respectively, on one line, that is, unstacked. In that case, the options are separated by OR symbols (|). Examples of this notation are

   {phasename|(1)} [,entrypoint|,(0)]

7. An ellipsis (a series of three periods) indicates that a variable number of items may be included.
8. **filename** — Example of a symbol appearing in the name field of a DTF macro.
9. **n** — Self-defining value, such as 3, X'04', (15), B'010'.
10. **length** — Absolute expression, as defined in *OS/VS-DOS/VSE-VM/370 Assembler Language.*
11. [<u>A</u>|**B**|**C**] — Underlined elements represent an assumed value in the event an operand is omitted.
12. **(r)** — Ordinary register notation. Any register except 0 or 1 to be specified in parentheses.
13. **(0)|(1)** — Special register notation (ordinary register notation can be used.)

Register Notation

Certain operands can be specified in either of two ways:

1. You may specify the operand directly which results in code that, for example, cannot be executed in the SVA because it is not reentrant.
2. You may load the address of the value into a register before issuing the macro. This way the generated code is reentrant and may be executed in the SVA. When using register notation, the register should contain only the specific address; high order bits should be set to 0.

In the latter case, you must specify the register in the macro. (The registers that can be used for this

purpose are discussed under "Register Usage," above.)

When the macro is assembled, instructions are generated to pass the information contained in the specified register to IOCS or to the supervisor. For example, if an operand is written as (8), and if the corresponding parameter is to be passed to the supervisor in register 0, the macro expansion contains the instruction LR 0,8. This is an example of ordinary register notation.

You can save both storage and execution time by using what is known as special register notation. In this method, the operand is shown in the format description of the macro as either (0) or (1), for example. This notation is special because the use of registers 0 and 1 is allowed only for the indicated purpose.

If special register notation is indicated by (0) or (1) in a macro format description and you use ordinary register notation, the macro expansion will contain an extra LR instruction.

The format description for each macro shows whether special register notation can be used, and for which operands. The following example indicates that the filename operand can be written as (1) and the workname operand as (0):

GET {filename|(1)} [,workname|,(0)]

If either of these special register notations is used, your program must load the designated parameter register before executing the macro expansion. Ordinary register notation can also be used.

Operands in (S,address) Notation

Certain system control macros (e.g. ATTACH, GENIORB, GENL, LOAD) allow three notations for an operand:

1. Register notation, as described in the preceding paragraphs.
2. Notation as a relocatable expression which, in the macro expansion, results in an A-type address constant.
3. Notation in the form (S,address). In the macro expansion, an explicit address (that is, an assembler instruction address in base-displacement form) is generated. The address can be specified either as a relocatable expression, for example: (S,RELOC), or as two absolute expressions, the first of which represents the displacement and the second, the base register, for example: (S,512(12)).

You should consider using this notation if your program is to be reenterable. In a reenterable program, macro operands often refer to fields in dynamic storage. The (S,address) format offers an alternative to register notation: if two or more of such operands have to be provided for one macro, there is no need for loading addresses into that many registers.

Declarative Macro Statements

The operands of the DTFxx and the logic module generation macros are written in assembler format statements. Figure 1-2 shows an example of a DTFMT macro. The first statement is the header and the continuations following are the detail statements. The header contains:

- The symbolic name of the file in the *name* field. In a DTF, the symbolic file name may have as many as seven characters. The file name may also be required on standard label job control statements and in certain macros as operands; it must be the same as that used in the DTF header. For a logic module, the name is not usually required. See "Module Names", above.
- The mnemonic operation code of the macro in the *operation* field.
- Keyword operands in the *operand* field, as required.
- A continuation character in column 72, if required.

Note: Avoid using IJ as the first two letters when defining symbols as they may conflict with IOCS symbols beginning with IJ. Avoid symbols that are identical to a filename plus a single character suffix because IOCS generates symbols by concatenating the filename with an additional character. For the filename RECIN, for example, IOCS generates the symbols RECINS, RECINL, etc.

The details follow the header and may be arranged in any convenient order. In Figure 1-2, the programmer has written only one operand on each detail line in order to make it easier to change the DTFMT specifications later, if necessary. If more than one operand is written on a detail line, they must be separated by a comma only. Except for the final detail line, there must be a comma immediately following each operand and have a continuation character in column 72. Each continuation line must begin in or after column 16. You may include a comment on a header or a detail line if there is room between a space following the last operand on a line and column 72.

Chapter 2: File Organization

This chapter discusses data organization for files, explaining how a file can be organized according to the requirements of the user. It serves as a basis for the following chapter, which introduces three access methods that can be used for batch processing and discusses which file organization and access method are best suited to a particular data processing application.

When files are being organized and programs being designed to do a particular job, certain questions must be asked:

What kind of information is available?

How much of the file must be processed in the program?

How many different programs must use the same file?

How often will the information in the file be processed?

When properly asked and answered, these questions will lead to decisions to:

- process all or most of the records of a file sequentially in one program,
- process only selected records of a file randomly in one program,
- process selected records randomly in two or more programs or,
- any combination of the above.

Data organizations that can be chosen to meet these requirements include *sequential* without an index for processing records sequentially, and either *sequential with an index* or *direct* organization for processing records randomly.

If two or more programs process the same file, a more complex data organization, one with a primary index and one or more secondary indexes, for example, may be appropriate. Such a complex data organization, which allows access to its records for different purposes, is commonly referred to as a *data base.* Data base organization is not discussed in this book but is introduced in *DL/I DOS/VS General Information* manual, and is treated in more detail in other DL/I DOS/VS publications.

Sequential Organization without Index

In a file with sequential organization, records are stored adjacent to one another, the physical sequence of the records in the file corresponding to the sequence of the primary keys (see Figure 2-1).

This type of organization requires that, to retrieve a specific record, all the *preceding* records must be read. Therefore, a typical example of a sequentially organized file might be one such as a payroll file that has *all* of its records read at weekly intervals. This is because the advantage of sequential organization is rapid access to the next record in the file and it is best utilized in applications that process all or most of the records in the file.

Sequentially organized files are found most commonly on serial storage media such as tape and punched card, but sequential organization can also be applied efficiently to files on DASD.

Direct Organization

A direct access file is characterized by some predictable relationship between the key of a record and the address of that record on a direct access device. The physical sequence of the records in such a file seems to follow a random order. This explains why such a file is sometimes called a randomly organized file.

Direct organization ignores the physical sequence of the records, and they are accessed on the basis of their physical location on the storage device. The advantage of direct organization is that any record in a file can be accessed without reading all the preceding records. This form of organization is applicable only to direct access devices (DASD) such as disks.

In a direct organization, records are located and identified by means of either a transformation *algorithm* or by *relative record number.*

Direct organization, through an algorithm, means that a formula is used to establish a relationship between the primary key of a record and the address of that record on a DASD (see Figure 2-2).

record 1 prim.key = 000016	record 2 prim.key = 000258	record 3 prim.key = 000783	record 4 prim.key = 006846	record 5 prim.key = 006847	etc.

Figure 2-1. Sequential data organization. Records are usually stored in the sequence of an ascending identifier. They are always accessed in their physical sequence.

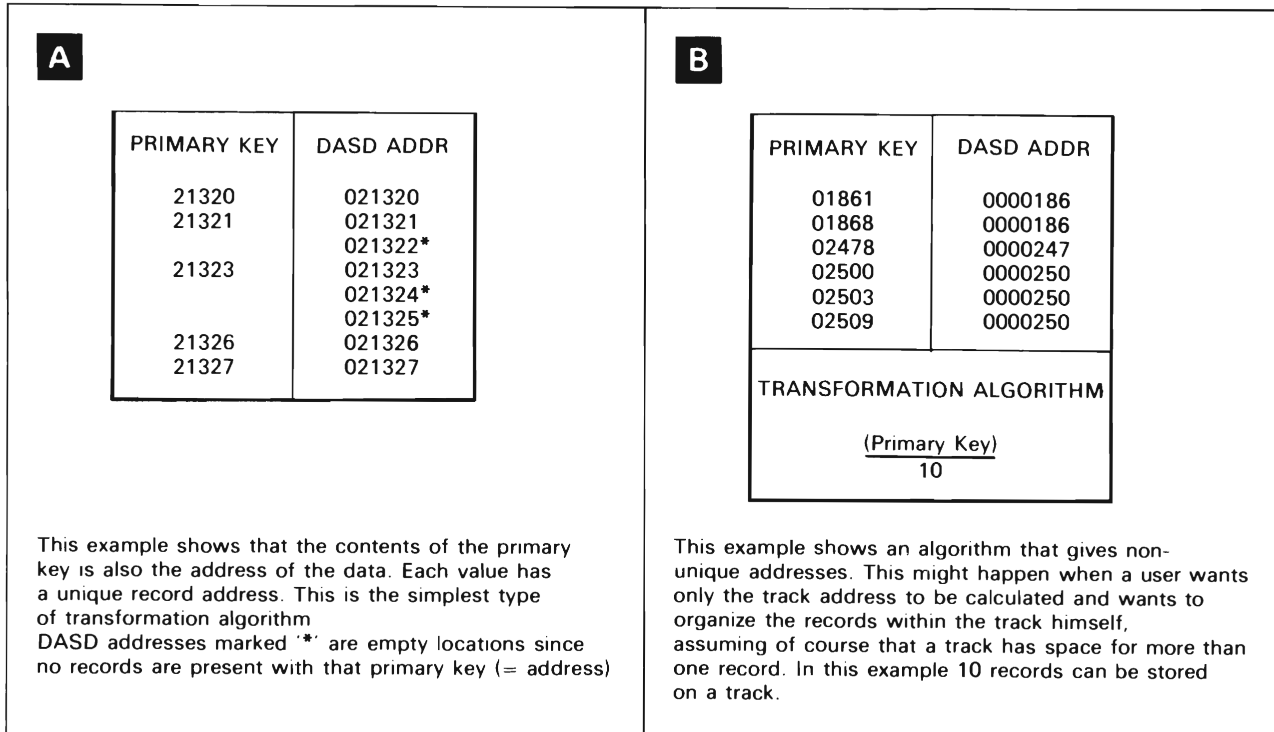


Figure 2-2. Direct organization through an algorithm. In direct organization, an algorithm is used for relating keys to the addresses of data records that need not be in physically sequential locations.

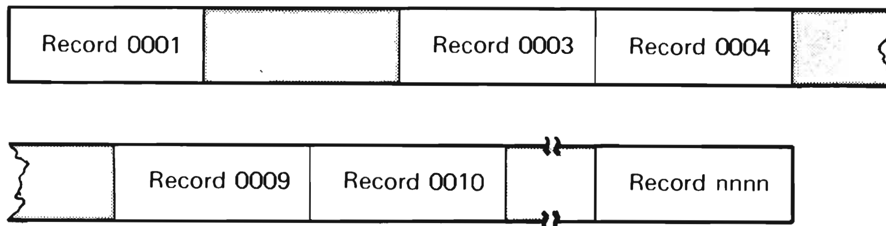


Figure 2-3. Example of the layout of a relative record file.

This technique is often called *randomizing* or *hashing*. The formula, or transformation algorithm, is usually arrived at by trial and error. That is, an algorithm is chosen and all primary keys are processed. Then an analysis of the resulting storage addresses is made and, if the algorithm proves to be inadequate, a new one is developed and tested. An algorithm is considered inadequate if it transforms many different primary keys into the same storage address, or if many addresses are never used.

Direct organization by relative record number is a technique that can be used if (1) all records of a file have the same length and (2) each record of the file is assigned a number within the file. This number is multiplied by the number of bytes in each record; the result is a byte address relative to the beginning of the file. Figure 2-3 shows how the records might be arranged in a relative record file.

Data Areas

A direct access file usually has two types of data areas, the *prime data area* and the *overflow area*. The prime data area is the area in which the records are written when the file is created or subsequently reorganized. Additions to the file may also be written in the prime data area. It may span multiple volumes and may consist of several non-contiguous areas.

The overflow area is used for records that, because of lack of space at a particular address, cannot be placed in the prime data area. There are two kinds of overflow areas (Figure 2-4): a *cylinder overflow area,* where the last tracks of each cylinder are reserved for overflow records, and an *independent overflow area,* where one or more cylinders on either the same or a separate volume are reserved for overflow records. Either or both of these types may be used for a direct access file (see Figure 2-5).

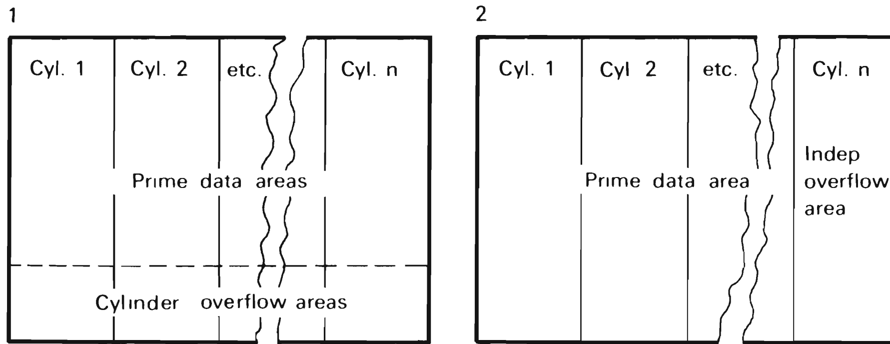


Figure 2-4. Two types of overflow areas.

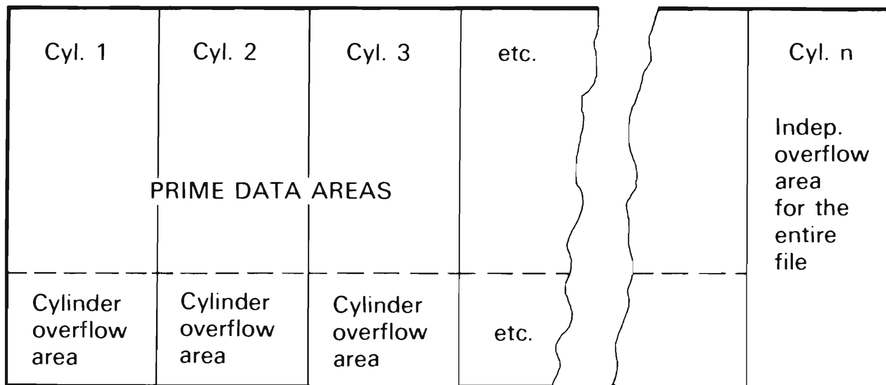


Figure 2-5. Combining the two types of overflow areas.

The independent overflow area may be used as an overflow area without any special structure. If a record does not fit in the prime data area or in the cylinder overflow area because both these areas are full, it may be placed anywhere in the independent overflow area.

The independent overflow area may also be organized as an extension of all prime data cylinders. For example: if a record must be stored in track *xx* of cylinder *yy*, according to the randomizing algorithm, this record can also be placed in track *xx* of the independent overflow area (assuming one separate cylinder), or in track *xx* of any cylinder of the overflow area (assuming several cylinders for the overflow area). This means that you can retrieve a record in an independent overflow cylinder by modifying (with appropriate cylinder addresses) the randomizing algorithm that was used to search the prime data area. Note that when you use only an independent overflow area, more I/O operations are required than when you use it with cylinder overflow areas.

The choice of a good conversion algorithm is very important. A poor algorithm will produce so many synonyms that the overflow areas may prove to be too small and the space in the prime data area is used inefficiently. This means a waste of space. If a good algorithm is used, however, the direct access technique may be the most flexible method of all.

If an algorithm converts record keys to record addresses, only the first record whose key is converted to a particular record address can be placed in the prime data area. Any other record whose key converts to the same address will have to be placed in the overflow area.

The design of the IBM direct access devices allows you to, in certain cases, randomize to a *track* address instead of a record address. In some cases, even a cylinder address may be sufficient. If an algorithm converts record keys to a track address, as many records as fit on the track can be placed in the prime data area. Only when the track is full will a record whose key converts to this track address be placed in the overflow area. Under these conditions, developing a good randomizing algorithm is less urgent.

Sequential Organization with Index

When this data organization is used, there is a separate list or index that contains the primary keys of records in the file. (See Figure 2-6.) Each key is accompanied by a reference to the actual data in

external storage. The index may be part of the file, or may be a separate file itself.

A sequential file with an index is similar to a sequential file in that rapid sequential processing is possible. In addition, the associated index makes it possible to locate individual records for nonsequential processing. That is, you can refer to records at random throughout the file, or to records in their presorted sequence. The organization also provides for additions to the file at a later time, while still maintaining both the random and sequential reference capabilities.

There are three areas in an indexed sequential file. The *prime data area* contains data records and related track indexes. The *index area* contains the cylinder index and, if present, the master index. The *overflow area* is used when records are added.

Prime Data Area

The data records are contained in the prime data area, the starting and ending limits of which are specified by EXTENT job control statements. The records in the prime data area are arranged in physical sequence by key. Each data block is written with count, key, and data areas. The count area specifies the sequence number of the physical block on the track, the length of the key area (which must be constant throughout the file), and the length of the data area (which must also be constant).

Records in the prime data area can be either blocked or unblocked. (See Figure 2-7.) When records are blocked, the key to the highest (last) record in the block is the key for the block and, therefore, is stored in the key of the record.

When an indexed sequential file is created, all data blocks are written in the prime data area, and the whole prime data area is filled with data blocks.

Space for the prime data area is allocated in units of cylinders; it must begin with track 0 of a cylinder and must continue to the last track of the same or a subsequent clinder. Track 0 of every cylinder is used for the track index.

When an indexed sequential file spreads over more than one volume, the prime data area must be continuous from one volume to the next, with the exception of cylinder 0, which is reserved for labels. The space distribution for an ISAM file is illustrated in Figure 2-8.

Note: The prime data area of a multivolume file on a 3340 cannot extend over different types of data modules.

Indexes

As ISAM (Indexed Sequential Access Method) loads a file of records sorted by key, it builds a set of indexes for it. The indexes:

- permit rapid access to individual records for random processing and
- supply the records in key order for sequential processing.

Either two or three indexes are built: a track index and a cylinder index are always built, and a master index is built if you specify the DTFIS MSTIND operand.

Once a file is loaded and the related indexes are built, the ISAM routines search for specified records by referring to the indexes. When a particular record (specified by key) is requested, ISAM searches the master index (if used), then the cylinder index, then the track index, and finally the individual track. Each index narrows the search by pointing to the portion of the next-lower index whose range includes the specified key.

Because of the high speed and efficiency of the direct access devices, a master index should be established only for exceptionally large files, for which the cylinder index occupies several tracks (five or more). That is, it is generally faster to search only the cylinder index (followed by the track index) when the cylinder index occupies four or less tracks.

The indexes are made up of a series of entries, each a separate record composed of both a key area and a data area. The key area contains the highest key on the track or cylinder, and its length is the same as that specified for logical data records in the DTFIS KEYLEN operand. The data area of each in-

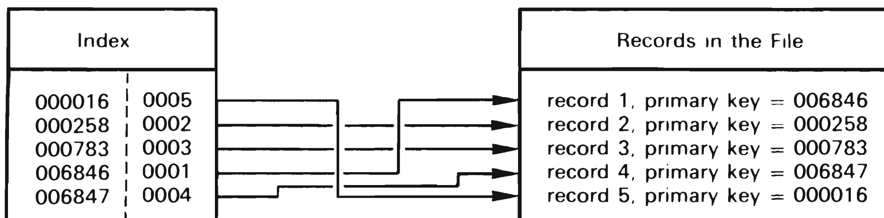


Figure 2-6. Example of a sequential organization with index.

dex is ten bytes long; it contains track information including the track address.

The indexes are terminated by a dummy entry that contains a key of all one bits. Therefore you should not use a key of all one bits for any of your records.

Track Indexes

Track indexes represent the lowest level in the index structure. There is one track index for each cylinder in the prime data area; it is written on track 0 of the cylinder it indexes. Sometimes it needs more than one track and occupies also space on the next track, or tracks.

A track index has one entry for each track of the cylinder. This entry contains information on the prime data track as well as on the records that have been shifted out of it into the overflow area. Actually, the index entry is a double entry, namely a normal entry and an overflow entry (see Figure 2-9).

Figure 2-10 shows part of a track index and its related prime data tracks.

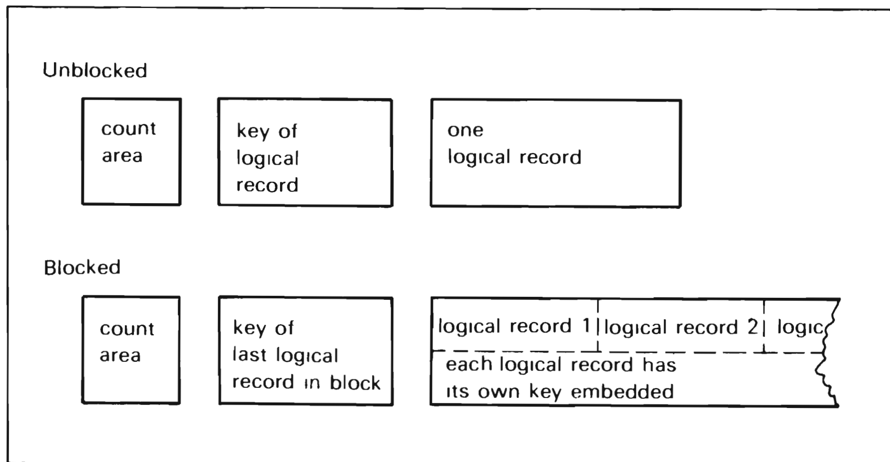


Figure 2-7. Prime data records.

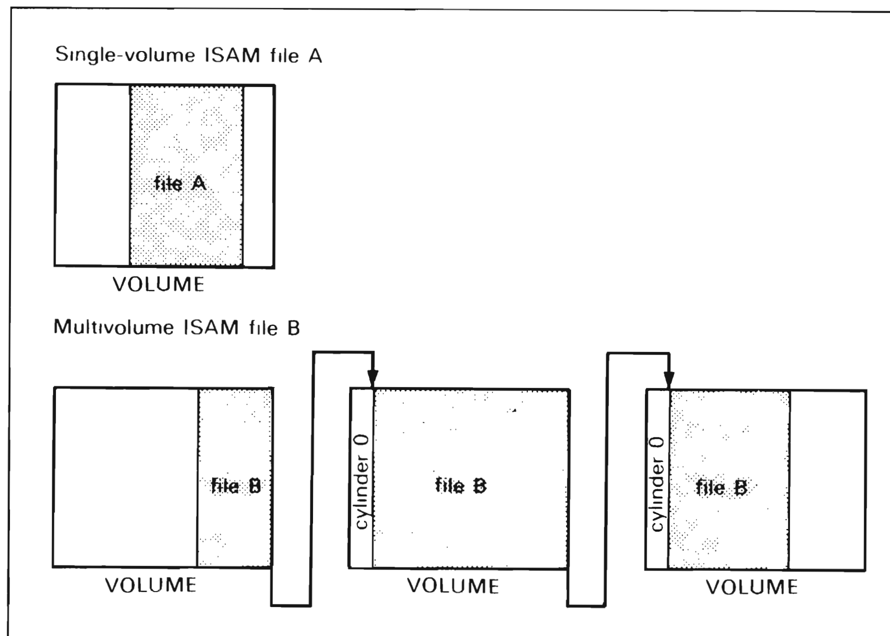


Figure 2-8. The prime data area.

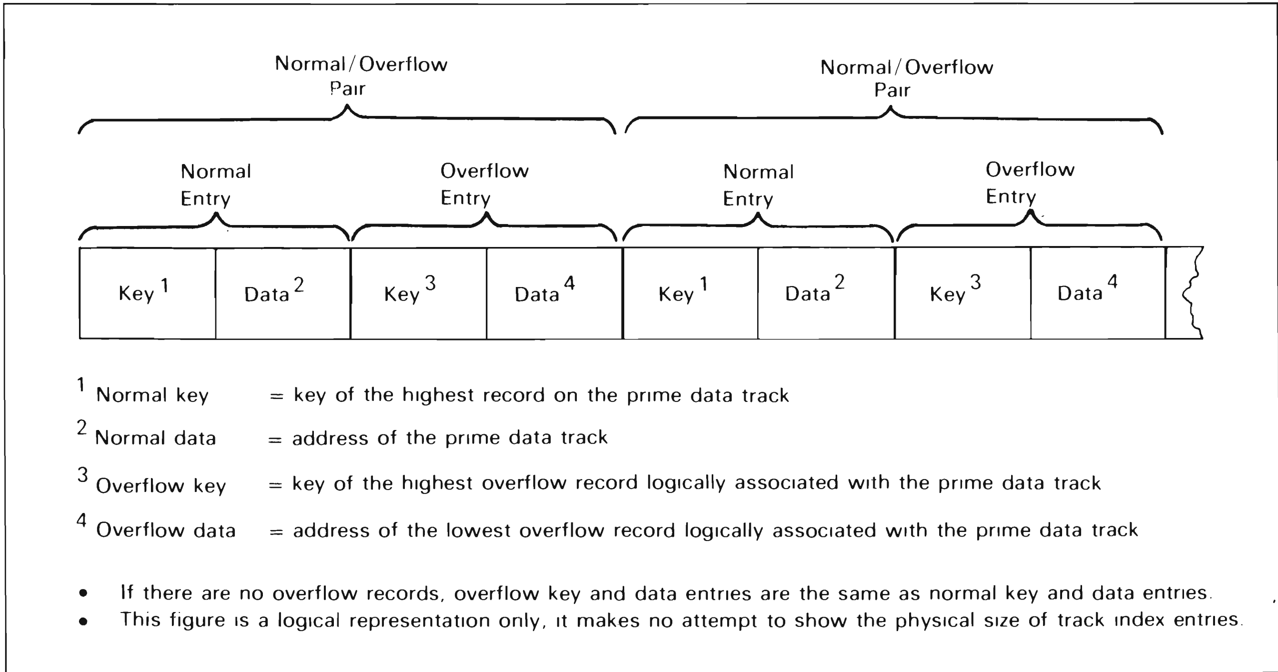


Figure 2-9. Format of track index entries.

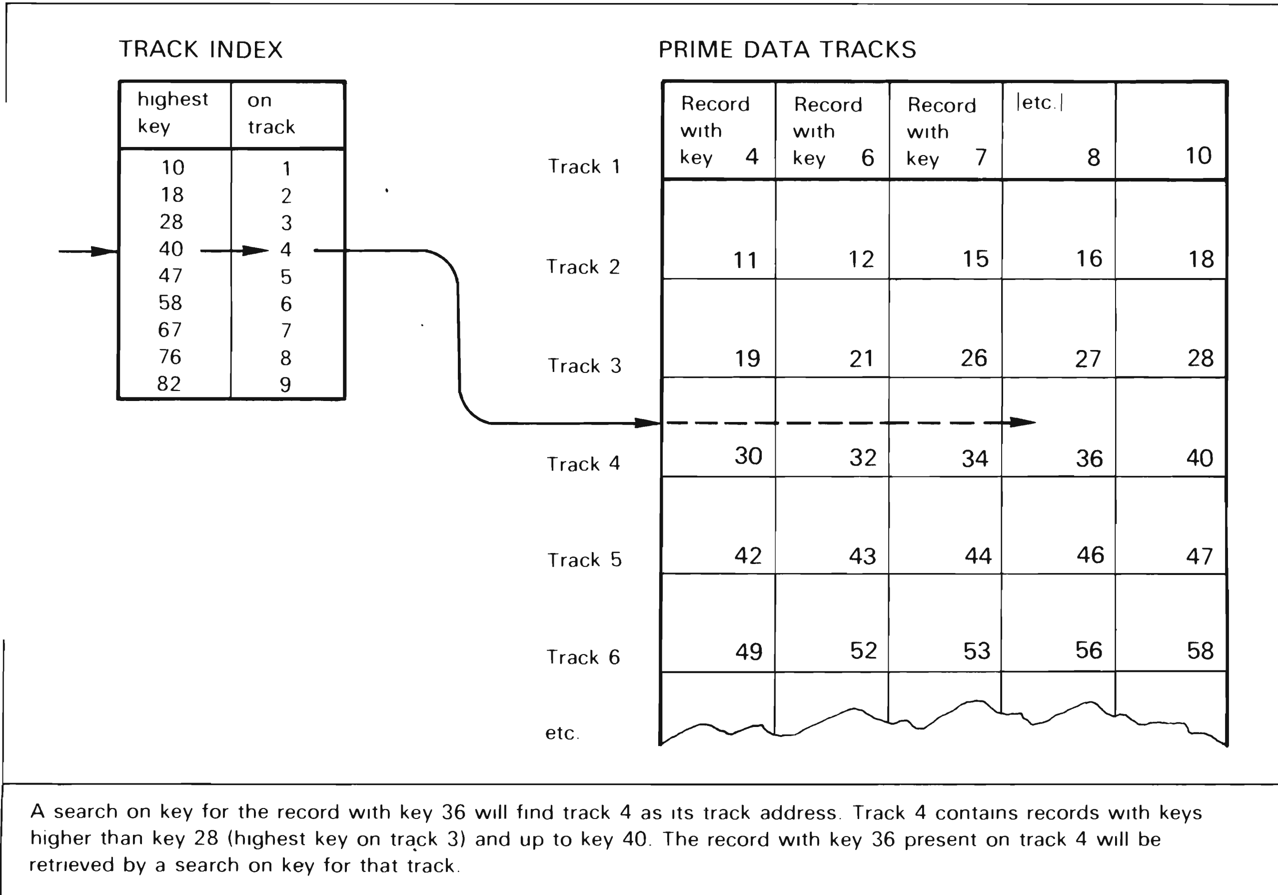


Figure 2-10. Track index and related prime data tracks. The overflow entries in the track index are not shown.

Cylinder Index

The cylinder index is an intermediate level index for the logical file. It contains an index entry for each cylinder occupied by the file. This index is built in the location which you specify in an EXTENT job control statement. You may change the upper extent limit; however, no validity check is performed by the ISMOD and it is therefore your responsibility to make sure the change is correct.

This index contains one entry for each cylinder occupied by the file. The key area contains the highest key associated with the cylinder, and the data area contains the address of the track index for that cylinder. The dummy entry indicates the end of the cylinder index.

The cylinder index may not be built on one of the cylinders that contains prime data records. Also, it should not be built on a cylinder that contains overflow records as this could prevent future expansion of the overflow area. The cylinder index should be on a separate cylinder; it may be on a separate volume that is on-line whenever the logical file is processed.

The cylinder index may be located on one or more successive cylinders. Whenever the index is continued from one clinder to another, the last index entry on the first cylinder contains a linkage field that points to the first track of the next cylinder. A cylinder index may not be continued from one volume to another, however.

Because cylinder index and track index must be searched to locate individual records, it is useful to have the cylinder index resident in virtual storage. You can have all, or part, of the cylinder index reside in virtual storage. If only a part of the cylinder index is kept in virtual storage, it is advisable to presort the input transactions; otherwise, portions of the index might have to be read several times. If all of the cylinder index can reside in virtual storage at the same time, there is of course no need to presort the input transactions.

Master Index

If desired, an even higher index can be created; the master index, which contains one entry for each track of the cylinder index. It is advisable to use a master index if the cylinder index occupies more than four tracks and is not kept in virtual storage permanently. The time to search for a given record will then be significantly shorter.

The optional master index is the highest-level index for a logical file. This index is built only if it is specified by the DTFIS MSTIND operand. A master index is built in the location specified by an EXTENT job control statement. Like the cylinder index, it may be located on the same volume with the data records or on a different volume that is on-line whenever the records are processed.

The master index must immediately precede the cylinder index on a volume, and it may be located on one or more successive cylinders. Whenever it is continued from one cylinder to another, the last index entry on the first cylinder contains a linkage field that points to the first track of the next cylinder. A master index may not be continued from one volume to another.

The master index contains an entry for each track of the cylinder index. The key area contains the highest key on the cylinder index track, and the data area contains the address of that track. The dummy entry indicates the end of the master index.

The complete index structure for an ISAM file, including a master index, is shown in Figure 2-12.

Overflow Area

ISAM uses the prime data area to store records when the file is created. ISAM further uses an overflow area, which is needed only if records are added to the file later. It is provided for those records that are forced off their original tracks by the insertion of additional records. Any new record is placed in the physical sequence exactly where it belongs, according to the value of its key. This implies that existing records with higher keys must be shifted. Shifting all of them would take too much time. Therefore, only the records on the track that is affected will be shifted. The record that is forced off its track because of this shifting is put into the overflow area. The track index is adjusted in the process.

Records that are moved to the overflow area remain associated with their original prime data tracks. They are always unblocked, even when the records in the prime data area are blocked. Each record has an additional link field (10 characters long) in its data area. Figure 2-13 shows the structure of an overflow record.

The overflow entry (in the track index) for a given track references all overflow records associated with that prime data track. They are chained to one another through the information in their link fields, and the overflow entry points to the beginning of that chain.

You may request two types of overflow areas (see Figures 2-4 and 2-5):

- A *cylinder overflow area* for each cylinder, which provides a certain number of tracks on

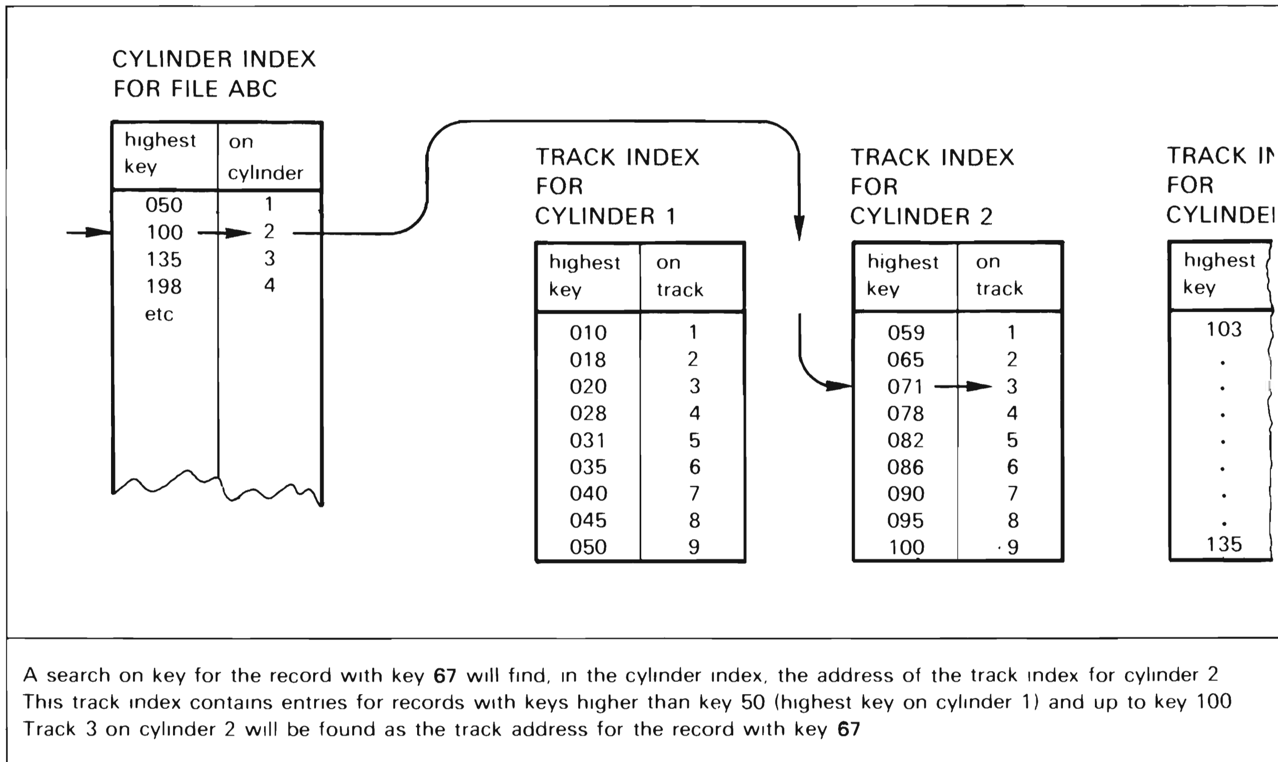


Figure 2-11. Cylinder index and related track indexes. The overflow entries in the track indexes are not shown.

each cylinder to hold the overflow records of that cylinder.

- An *independent overflow area* for the entire file, which provides a certain number of tracks on a separate cylinder, perhaps even on another volume.

You may use either of these types or both of them. If you use them together, the independent overflow area will be used whenever one of the cylinder overflow areas is full.

Example of an ISAM File

Figure 2-14 shows schematically a simplified example of a file organized on a DASD by ISAM. This figure illustrates a file on a 3330, with the last two tracks on each cylinder used for the overflow area. The same file would have similar characteristics if it was created on another DASD type. The assumptions made and the items to be noted are:

1. The track index occupies part of the first track, and prime data records occupy the rest of the track. This is called a shared track.
2. The data records occupy part of track 0 and all of tracks 1-16. Tracks 17 and 18 are used for overflow records in this cylinder.
3. The master index is located on track X on a different cylinder. The cylinder index is located on tracks X+1 through X+20.
4. A dummy entry signals the end of each index.
5. The file was originally organized with records as follows:

Track	Records
0	5 - 75
1	100 - 150
2	.
.	.
16	900 - 980

6. The track index originally had two similar entries for each track. It now shows that overflow records have occurred for tracks 1 and 16.
7. Records 150, 140, and 130 were forced off the track by insertions on the track. Record 135 was added directly in the overflow area.
8. An SL (sequence-link) field was prefixed to each overflow record. The records for track 1 can be searched in sequential order by following the SL fields:

Record	Sequence-Link Field (SL)
130	SL points to record with key 135
135	SL points to record with key 140
140	SL Points to record with key 150
150	End of search. (Key 150 was the highest key on track 2 when the file was loaded.)

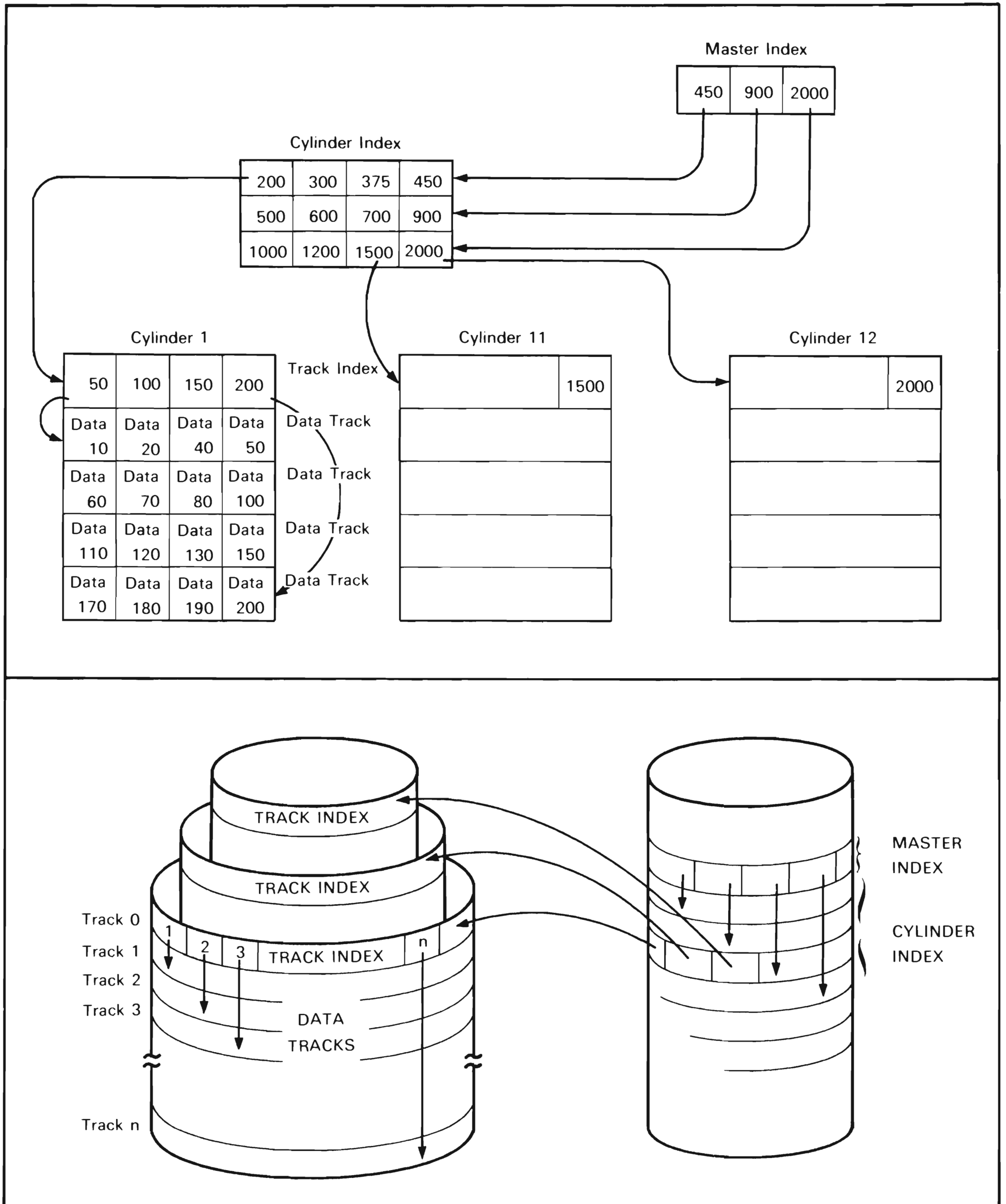


Figure 2-12. Index structure for an ISAM file.

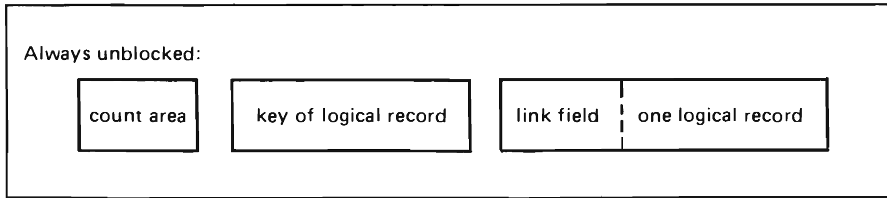


Figure 2-13. ISAM overflow record.

9. When the file was loaded, the last record on cylinder 1 was record 980; on cylinder 2, record 1850; and on cylinder 9, record 4730. This is reflected in the cylinder index. The first entry in the master index is the last entry of the first track of the cylinder index.

10. When cylinder overflow areas are used, the first record (record 0) in the track index for a cylinder is the COCR (Cylinder Overflow Control Record). It contains the address of the last overflow record on the cylinder and the number of tracks remaining in the cylinder overflow area. When the number of remaining tracks is zero, overflow records are written in the independent area. The format of the record zero data field is as follows: hhrbbtxx overflow area.

hh- last cylinder overflow track containing the records.

r- last overflow record on the track.

bb- the number of bytes remaining on the track (this is binary zeros for fixed-length records).

t- the number of remaining tracks available in the cylinder overflow area.

xx- reserved (with binary zeros).

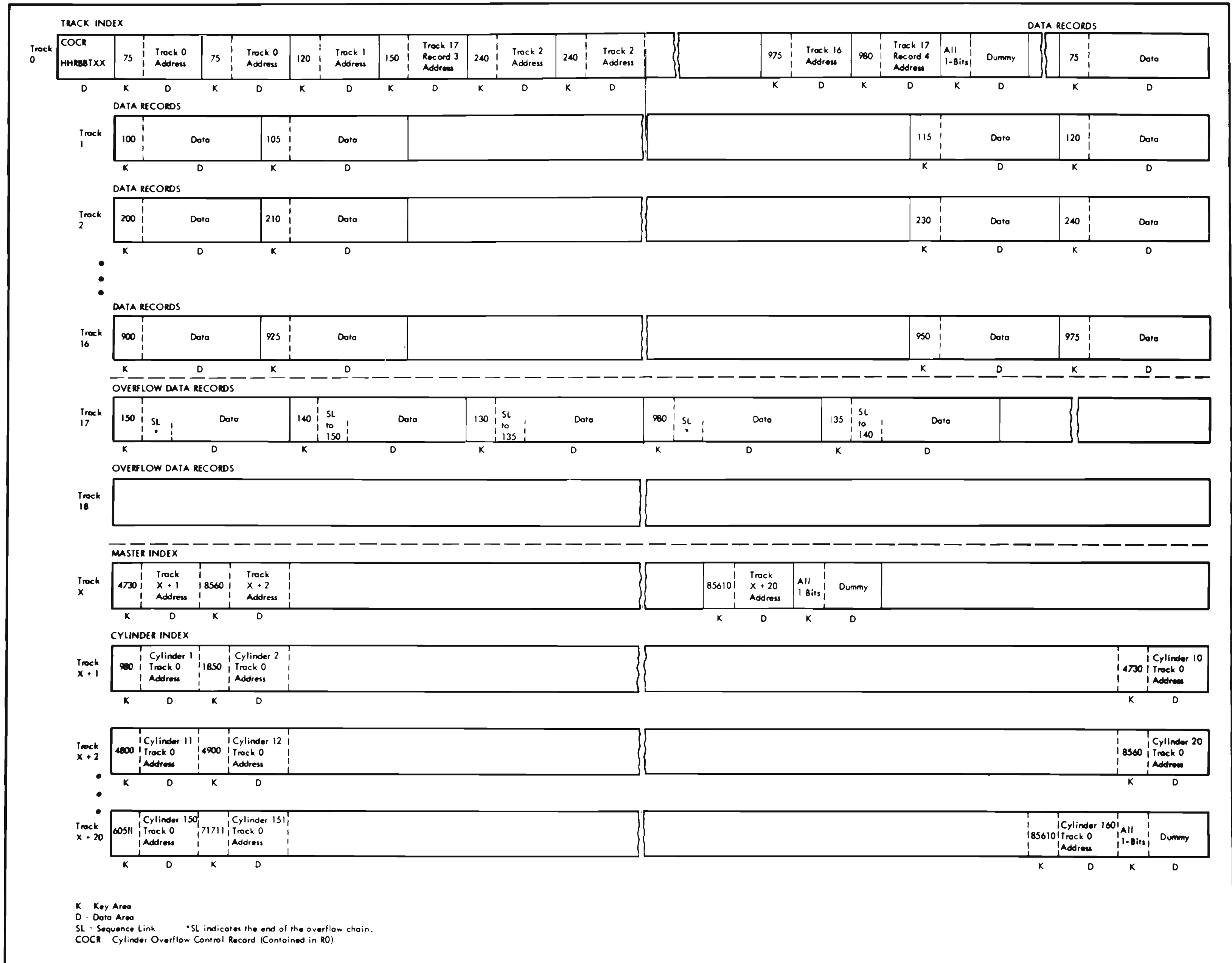


Figure 2-14. Example of a file organized by ISAM.

Chapter 3: Access Methods Concepts

Remember that a file may be organized in various ways and the same file may be subjected to more than one processing method. It is the user's responsibility to choose both an organization and an access method to fit his application.

The Input/Output Control System (IOCS) includes various types of file processing routines; these routines are grouped into three different types of access methods:

- SAM (Sequential Access Method), which allows records to be read and written one after the other.
- DAM (Direct Access Method), which allows records to be read from or written at a DASD address determined by the user.
- ISAM (Indexed Sequential Access Method), which allows records (1) to be read or written in logical sequence determined by keys, one after the other, or (2) to be read from or written at DASD locations determined by keys.

The routines of these three access methods, as a group commonly referred to as logical IOCS (LIOCS), provide the user with a selection from which he can choose the most suitable method to satisfy his specific problem and installation conditions. They provide all the functions necessary to organize a file and to retrieve records from the file for later processing.

SAM (Sequential Access Method)

The Sequential Access Method applies to sequentially-organized files, that is, to files whose records are located adjacent to one another on the storage medium and for which no index exists. An example of this is a deck of punched cards or, even, playing cards. If you are not a magician, you must look at (or 'read') each card in the deck in order to find a particular card. If you want to find several cards in the deck, you may have to read through the deck several times before you locate all of them.

One way to reduce the work involved in locating several cards is to have the cards sorted into a known sequence and to have those cards that you want to find listed in the same sequence. This is, in practice, how sequential files are almost invariably organized: a convenient field in the record is selected and the records in the file are sorted in order, based on this field.

For instance, in a personnel file, the sort field might be either employee number or name; in a parts inventory file, it might be part number. The actual field selected depends, of course, on both the nature of the file itself and the particular data processing applications involved.

The foregoing remarks referred to card files but in fact, you can use the Sequential Access Method for other serial storage media such as magnetic or paper tape, diskette, or sequentially-organized DASD. SAM accepts all record formats supported in VSE, creates sequential files from sorted input records, can update in place sequential DASD records, and with some devices, read a card and punch additional information into the card.

The SAM macros allow you to process a file with a minimum of effort. There are two varieties of processing available to you:

- record processing, using GET/PUT macros;
- block processing, using READ/WRITE macros.

When you deal with logical records, SAM does the blocking and deblocking for you, as needed. You can process the data in either a work area or in one or two I/O areas, and SAM provides for overlapping your processing with physical transfer to or from the storage medium. If you choose to work with blocks instead of logical records, you can, to a certain extent, do direct processing on sequential DASD files. You must, however, code your own blocking and deblocking routines.

The necessity to read all preceding records in a file in order to find a particular record is both the strength and weakness of the Sequential Access Method. It is relatively inefficient when a low percentage of the records in a file is to have transactions posted to it, because of the need to read all records. Conversely, there is no disadvantage when most or all of the records are to be processed. In addition, SAM allows relatively simple record-handling logic to be used.

Control Interval Format

The discussion of the Sequential Access Method deals in generalities to which some exceptions must be made when you use CI (Control Interval) format. Control interval format is required by VSAM (Virtual Sequential Access Method) and by Fixed Block Architecture storage devices such as the IBM 3310. VSE/VSAM, an IBM licensed program, is discussed only incidently in this manual; for information see the VSE/VSAM manuals listed in the Preface of this book. In the context of this manual, control interval format is used only with sequentially organized files

that are stored on FBA (Fixed Block Architecture) devices.

Data storage on FBA devices contrasts with data storage on CKD DASD, which depends on the presence of physical data blocks containing a separate Count area, optional Key area, and Data area to permit data access. Data is stored on FBA devices in fixed-length data blocks called *FBA blocks*, whose length is specific for each FBA device in use. A control interval is the unit of data transfer between an area in virtual storage called the *CI buffer* and an FBA device. The length of a CI is an integral multiple of the FBA block length and may be specified in the DTF (Define The File) macro when a file is defined. The CI size can also be redefined later at execution time by means of the DLBL job control statement.

When CI format is *not* used, the unit of storage and of data transfer between virtual storage and an external storage device is the *physical block*, which is usually composed of several logical records. In its simplest form, with unblocked records such as for a work file, there is only one logical record in each physical block. When the CI format is used, the physical block is referred to as a *logical block*, to emphasize that it is not the unit of data storage and transfer.

A control interval is composed of one or more logical blocks, control information, and usually some free space. The number of logical blocks that make up a CI is determined by SAM, based on the record format and size, blocksize, and CI size that you specify. The relationship between the various record formats, the control intervals, and FBA blocks is illustrated in *VSE System Data Management Concepts*. Data transfer, and blocking/deblocking between your defined I/O area(s) and the FBA block, through the CI buffer, is automatically handled by SAM and the storage device control units and is generally of little concern to the problem program. Depending on the record format you use, however, you can control the number of records or blocks that are blocked into a CI to optimize storage medium efficiency.

Parts of the discussion of the Sequential Access Method that follows are concerned with DASD devices in general and with disk units in particular. If you are using control interval format with an FBA DASD, certain concepts and terms are invalid. For instance, FBA devices store data in fixed-length FBA blocks and SAM accesses the data by referring to the physical block number. Because of this, discussions concerning cylinder and track are meaningless when FBA devices are being used.

When blocking in general is discussed, bear in mind that what is referred to is the user-specified blocking of logical records into physical blocks (logical blocks, with FBA). Any further blocking/deblocking of logical blocks into and out of the CI buffer is specifically mentioned, where appropriate.

Data transfer between virtual storage and an FBA device takes place asynchronously through the CI buffer. That is, a physical write to the file does not necessarily take place when a PUT or WRITE macro is issued, but rather when the CI buffer is filled. Likewise, a physical read takes place when GETs or READs have moved all data from the CI buffer to the user-defined I/O areas.

Finally, logic modules need not be explicitly defined by the programmer for SAM or DAM files stored on DASD devices. Support for these devices includes preassembled logic modules that are loaded into the SVA (system virtual area) at IPL time and are linked to the problem program when a file is opened. To maintain device independence, however, you may choose to specify a logic module for a SAM file; if the file is later assigned to a DASD device, the correct logic module is used by SAM, instead of the one that you defined.

Defining Files and Functions

A file can be described in several ways, based on its organizational and physical characteristics, how it is used, and the device on which it is stored. The DTF macros describe the size and format of your records to SAM by means of operands that you specify. Other operands define actions that you want SAM to take, depending on events that take place or conditions that are met. SAM requires that each file that your program processes be described to it by means of a DTF declarative macro. There are a number of different DTF macros available for your use; the proper one to use depends upon the particular storage medium and the I/O device that you will use to access the file. Refer again to Figure 1-3 for a list of I/O devices and their corresponding SAM DTF macros.

For more details concerning the DTF operands that you must or may specify, see the following sections on the particular devices that you will be working with.

Record Specification

Among characteristics that you can specify are record type, record format, and record size.

Depending on the particular DTF involved, these may be mandatory or optional.

Record Type. Refers to how a file is processed, that is, whether it is used as input, output, or as a work file; whether it is updated in place; or whether it is an associated file. Associated files are concerned with card and printer files and are discussed in "Chapter 7: Processing Unit Record Files". The DTF operands ASOCFLE and FUNC define file association.

If your file is on a direct access device or certain card devices, it can be updated in place; that is, it can be read and then have additional information written back into the original record. For updating card files, specify TYPEFLE=CMBND in the DTFCD; for DASD files, specify TYPEFLE=INPUT or WORK and also UPDATE=YES.

Work File. Is a single volume DASD or magnetic tape file that is used to pass intermediate results between successive phases or job steps. Like an update file, a work file can be read and have information re-written into it without closing and re-opening it. Work files use fixed-length unblocked records or undefined format, and are specified by:

TYPEFLE=WORK.

Certain devices, such as magnetic character readers and optical readers, can only *read* records; specifying the DTFs for these devices defines their files as input. Other DTFs assume that a file is input unless you explicitly specify otherwise, as with:

TYPEFLE=OUTPUT.

Record Format. Refers to whether your records are of a constant ('fixed') or variable length and if they are blocked or not. Records transferred between an I/O device and virtual storage are considered fixed unblocked unless otherwise specified. For more information on record formats and their relationship to control intervals, see the section on the topic in *VSE System Data Management Concepts.*

Record Size. For certain record types and formats, SAM requires that you inform it of the record size by specifying the RECSIZE operand. Its parameter is either the number of bytes in a fixed-length unblocked record or, using register notation for other record formats, the register in which the record size is to be found during execution. For output files, you must load the record length into the register specified by RECSIZE=(r) before issuing a PUT macro. For input files, SAM places the length of the record transferred to virtual storage into the specified register. The RECSIZE operand is invalid for work files.

I/O Area Specification

When SAM reads or writes a file, it transfers data between an I/O device and an I/O area in virtual storage. The I/O area must be specified by name with the IOAREA1 operand. When you use control interval format, however, the primary I/O area is the CI buffer and the use of the (secondary) IOAREA1-named area is sometimes optional. To obtain speedier overlapped I/O processing, you may use a second I/O area by also specifying IOAREA2. The length of these I/O areas is specified by the BLKSIZE operand and depends on a number of factors such as record format and blocking and the device upon which the data is stored. Consult the discussions of the various devices in later sections of this book for permitted BLKSIZE ranges.

Note: If you do not specify an I/O area, the system will issue a GETVIS macro to obtain the area for you. If you specify a larger blocksize (for DTFSD data files only) than the previously specified blocksize, the system will dynamically allocate the larger I/O area required.

You may wish to actually process your records in a work area separate from the I/O area, especially if you are using blocked records. You can specify this by including the WORKA=YES operand in some DTFs and including the work area address in your imperative macros. If you do not use a work area, you need a register that points within an I/O area or CI buffer to either the next input record available or to the address where you can build an output record. This register is specified by IOREG; if omitted, it is assumed to be in register 2.

A work area cannot be used with paper tape or the 3881 optical mark reader.

Processing Specifications

The symbolic address (SYSxxx) and actual model number of the device that you are working with are specified by the operands DEVADDR and DEVICE. The addresses of your routines that process end-of-file and error conditions are specified by operands such as EOFADDR, ERROPT, ERREXT, and WLRERR.

Because these operands are required for some DTFs, optional for others, and invalid for yet others, you should consult for details the later chapters on the devices that you will be using.

Activating a File For Processing

To activate, or make ready a file for processing, you normally use the initialization macro OPEN.

The OPEN macro associates the logical file declared in your program with a specific physical file on an I/O device. Thus an OPEN macro must be issued for any file declared in your program with a specific physical file before processing is attempted;

an exception is that an OPEN need not be issued for DTFCN and DTFPT files in a non-self-relocating environment. The association of your logical file with a physical file remains in effect throughout your program until you issue a completion macro (see the section "Deactivating a File After Processing".)

For an output file with two I/O areas, OPEN loads your IOREG with the address of an I/O area. OPEN also checks or writes standard or non-standard DASD or magnetic tape labels.

If you prefer to do your own label checking and writing, specify another initialization macro, LBRET. This macro is discussed, along with other aspects of label processing, in the chapters "Processing DASD Files" and "Processing Magnetic Tape Files."

A maximum of 16 files may be activated with one OPEN by entering additional filenames.

Whenever an input/output DASD or magnetic tape file is opened and you plan to process user-standard labels (UHL or UTL) or non-standard tape labels, you must provide the information for checking or building the labels. If this information is obtained from another input file, that file must be opened ahead of the DASD or tape file. To do this, specify the input file ahead of the DASD or tape file in the same OPEN, or issue a separate OPEN for the file.

Alternatively, you can load the address of the DTF filename into a register and specify the register using ordinary register notation. The high-order 8 bits of this register must be zeros or unpredictable results may occur. The address of the filename may be preloaded into any of the registers 2 through 15 although practice usually restricts the choice to registers 2 through 12.

If OPEN attempts to activate a file whose device is unassigned, the job is terminated. If the device is assigned IGN, OPEN does not activate the file but turns on the DTF byte 16, bit 2, to indicate that the file is not activated. If DTF byte 16, bit 2 is on after issuing an OPEN, I/O operations should not be attempted for the file, as unpredictable results may occur.

Self-relocating programs (see Appendix D) *must* use OPENR for file activation. OPEN and OPENR perform essentially the same functions with the exception that when OPENR is specified, the symbolic addresses that are generated are self-relocating while, with OPEN, the addresses are not self-relocating. Throughout the manual, the term OPEN refers also to OPENR, unless stated otherwise.

Processing Data Files with SAM

SAM permits you to store and retrieve data records without coding your own blocking and deblocking routines, allowing you to concentrate on processing your data rather than processing your files. A major feature is the ability to use one or two I/O areas and to process records either in a work area or an I/O area.

The SAM routines provide for overlapping the physical transfer of data with processing. The amount of overlapping actually achieved is governed by your program through the assignment of I/O areas and work areas, and by the implementation of specific logic modules. An I/O area is that area of virtual storage to or from which a block of data is transferred by SAM. A work area is an area used for processing an individual record. For spanned records, the work area contains the entire record. This may consist of more than one block. The address of an I/O area is specified in the DTF macro, while the address of a work area is specified in the processing macro.

The following combinations of I/O areas and work areas are possible (except in the cases of spanned records and associated DTFCD files):

1. One I/O area without a work area
2. One I/O area with a work area
3. Two I/O areas without a work area
4. Two I/O areas with a work area.

When processing spanned records, you may use either one or two I/O areas with a work area. Although two I/O areas are permitted, normal overlap is curtailed because each imperative macro that you issue may require multiple I/O operations by the logic module.

When processing associated DTFCD files, you may use one I/O area, either with or without a work area. Although two I/O areas are not permitted for associated files, a type of overlapped processing can be achieved: see *VSE System Data Management Concepts* for details.

When processing other SAM files on FBA devices, all of the above combinations of I/O areas and work files are valid but the utilization of the area differs from the CKD devices. That is, with FBA devices, physical I/O transfers data between virtual storage and the FBA device through a single control interval (CI) buffer. When a work area is specified, read requests for a logical block result in data moving from the CI buffer directly to the work area to avoid an extra data move to the I/O area. This means that your specified I/O area(s) is ignored when a work

area is used. When no work area is used, data is transferred form the CI buffer to the I/O area(s). Since logical blocks are retrieved asynchronously (that is, not necessarily at the same time that retrieval requests are made), overlap can be achieved for output files and non-update input files if two I/O areas are provided, even though only one CI buffer is used.

Required DTF Macro Entries

You must specify an I/O area by means of the IOAREA1 operand of the DTF macro if you are *not* using the CI format with:

- a WORKA work area, or
- an IOREG I/O register with *input* files, or
- an IOREG register with fixed-length *output* records without truncation.

Note: If you do not specify an I/O area, the system will issue a GETVIS macro to obtain the area for you. If you specify a larger blocksize (for DTFSD data files only) than the previously specified blocksize, the system will dynamically allocate the larger I/O area required.

For a file other than a combined file or an associated file, two areas may be used to permit overlapping of data transfer and processing operations. The second area is specified as IOAREA2. Whenever two I/O areas are specified, the SAM routines transfer records alternately to or from each area. They completely handle this flip-flop so that the next sequential record or the proper output area is always available to your program for processing.

For a combined file, the input area is specified in IOAREA1 and the output area is specified in IOAREA2. If the same area is used for both input and output, IOAREA2 is omitted.

For associated files, only one input area may be specified.

Note: Combined and associated files pertain only to unit record devices.

When records are processed in the I/O area(s), a register must be specified in the IOREG operand of the DTF macro if:

1. Two I/O areas are used, or
2. Records are blocked (chained on diskette), or
3. Undefined or variable-length magnetic tape records are read backwards, or
4. Neither BUFOFF=0 nor WORKS=YES is specified for ASCII files.

The IOREG-specified register identifies the next single record to be processed. It always contains the absolute address of the currently available record or output record area. The GET and PUT routines place the proper address in the register. You should always address the I/O areas by using the IOREG as a base register and should not make any assumptions about which I/O area is presently being used. If a work area is used, WORKA=YES must be specified and IOREG must *not* be specified.

For output, if blocked records of variable length are built in the I/O area(s), an additional register must be specified with the VARBLD operand. SAM stores the number of bytes remaining in the output area in the VARBLD register each time a PUT macro is executed.

Obtaining a Record for Processing

The GET macro makes the next sequential logical record from an input file available for processing in either an input area or a specified work area, passing the data through an intermediate control interval buffer, if CI format is used. It is used for any input file in the system, and for any type of record. If GET is used with a file containing checkpoint records, they are automatically bypassed.

If a work area is specified, *all* GETs for the named file must use a register or workname. This causes GET to move each individual record from the input area or CI buffer to a work area. If you use a work area, WORKA=YES must also be specified in the DTF but IOREG must not.

All records from a logical file may be processed in the same work area, or different records from the same logical file may be processed in different work areas. In the first case, each GET for a file specifies the same work area. In the second case, different GET macros specify different work areas. If it is advantageous to use two work areas, remember that only one work area can be specified in any one GET macro: you must specify the alternate areas in alternate GET macros.

When records are *unblocked* and only one input area is used, each GET transfers a single record from an I/O device to the input area (or CI buffer). The record is then transferred to the work area, if one is specified in the GET macro. If two input areas are specified, each GET makes the last record that was transferred to virtual storage available for processing in the input area or work area.

When *blocked* records are specified for DASD or magnetic tape with the DTF RECFORM operand, each individual record must be located for processing (that is, deblocked). Therefore, blocked records are handled as follows:

1. The first GET macro transfers a block of records from DASD or tape to the input area or CI buff-

er. It also initializes the specified register to the address of the first data record, or it transfers the first record to the specified work area.

2. Subsequent GETs either add an indexing factor to the register or move the proper record to the specified work area, until all records in the block are processed.
3. Then, the next GET makes a new block of records available in virtual storage and either initializes the register or moves the first record.

When *spanned* records are processed, the operand RECFORM=SPNUNB or SPNBLK must be included in the DTF and in the appropriate logic module macro (MTMOD), if one is used. GET assembles spanned record segments into logical records in your work area. Null segments are recognized and not assembled into logical records, but skipped. The length of the logical record is passed to you in the register specified in the DTF RECSIZE operand.

If you choose to update logical records, the pointer to the physical record in which a logical record starts is saved on each GET so that the device may be repositioned. The extent sequence number (in the DTF) is also saved in case the logical record spans disk extents.

When *undefined* records are processed, the operand RECFORM=UNDEF must be included in the DTF macro. GET treats undefined records as unblocked, and you must locate (deblock) individual records and fields. If a RECSIZE register is specified, SAM stores in that register the length of the read record. SAM considers undefined records to be variable in length. No other characteristics of the record are known or assumed by SAM. Determining these characteristics is your responsibility.

An example of GET processing is shown in Figure 3-1. The operand IOAREA1 points to the first I/O area for this file, while IOAREA2 points to the second. The operands of the GET macro point to the DTF and to the work area A3, to which logical records are moved from areas A1 and A2 by SAM.

```
Name      Operation   Operand          Column 72
FNAME     DTFMT                             X
          •
          •
          •
                      IOAREA1=A1,           X
                      IOAREA2=A2,           X
                      WORKA=YES,            X
          •
          •
          •
A1        DS          500C
A2        DS          500C
          •
          •
          •
          GET         FNAME,A3
          •
          •
          •
A3        DS          100C
          •
```

Figure 3-1. GET macro processing example.

Filing a Record After Processing

The PUT macro writes, prints, or punches logical records that have been built in either an output area or in a specified work area. When control interval format is used, as with an FBA DASD, the PUT transfers data from the output area through an intermediate CI buffer. Blocking between storage and the CI buffer, and deblocking between the buffer and the device is performed by the operating system and the device control unit and is generally of no concern to the problem program. The PUT macro is used by any output file defined by a DTF, and for any record type. It operates much like the GET macro, but in reverse; it is issued after a record has been built.

Records may be built in a work area that you define in your program. PUT then moves each record from the work area to the (output) I/O area or CI buffer. If a work area is specified, all other PUTs to the named file must also specify it.

Individual records for a logical file may be built in the same work area or in different work areas. Each PUT macro specifies the work area where the completed record was built. However, only one work area can be specified in any one PUT macro.

Whenever a PUT macro transfers an output data record from an output (or work) area to an I/O device, the data remains in the area until it is either cleared or replaced by other data. SAM does not clear the data. Therefore, if you plan to build another record whose data does not use every position of the output or work area, you must clear that area before you build the record. If this is not done, the new record will contain interspersed characters from the previous record. If you specify a work area in your program, you should use only that area to build your records, and never change the contents of the I/O area.

When records are *unblocked*, each PUT transfers a single record from the output area (or input area if updating is specified) to the file. If a work area is specified in the PUT macro, the record is first moved from the work area to the I/O area or CI buffer and then to the file.

When *blocked* records are written on DASD or magnetic tape, the individually built records must be

formed into a block in the output area before it can be transferred to the output file. Fixed length blocked records can be built directly in an output area or in a work area. Each PUT macro for these records either adds an indexing factor to the register (IOREG), or moves the completed record from the specified work area to the proper location in the output area or CI buffer. When an output block of records is complete, a PUT macro causes the block to be transferred to the output file and initializes the register, if one is used.

Variable-length blocked records can also be built in either an output area or in a work area. The length of each variable-length record must be determined by your program and included in the output record as it is built. Your program can calculate the length of the output record from the length of the corresponding input records. That is, variable-length output records are generally developed from previously written variable-length input records. Each variable-length input record must include the field that contains the length of the record.

When variable-length blocked records are built in a work area, the PUT macro performs the same functions as it does for fixed-length blocked records. The PUT routines check the length of each output record to determine if the record fits in the remaining portion of the output area or CI buffer. If the record fits, PUT immediately moves the record. If it does not fit, PUT causes the completed block to be written and then moves the record from the work area.

However, if variable-length blocked records are built directly in the output area, the DTF VARBLD operand, the TRUNC macro, and additional programming are required. Your program must determine whether each record built will fit in the remaining position of the output area. This must be known before record processing for a subsequent record begins, so that the completed block can be written. Thus, the length of the record must be precalculated and compared with the amount of remaining space.

The amount of space available in the output area at any time is supplied to your program in a register by the IOCS routines if VARBLD is specified in the DTF macro. This register is in addition to the register specified in IOREG. Each time a PUT macro is executed, IOCS loads into the specified register the number of bytes remaining in the output area. Your program uses this to determine whether the next variable-length record will fit. If it will not fit, a TRUNC macro must be issued to transfer the block of records to the output file, or CI buffer. The entire output area is then available for building the next block.

Note: When end of track or CI overflow occurs, the logic module truncates the last variable-length blocked record to fit on the track. The records that did not fit on the track are moved to the beginning of the I/O area.

When PUT handles unblocked or blocked *spanned* records, the operand RECFORM=SPNBLK or RECFORM=SPNUNB (whichever applies) must be included in the file definition (DTFMT, DTFSD) macro and in the appropriate logic module definition (MTMOD) macro (if one is used.) Records in your work area are divided into spanned record segments according to the length specified in the BLKSIZE operand. In constructing the segments, full use is made of the space available in each physical record or logical block and extent unit. For disk output, spanned records do not span volumes. If there is not enough space on the current volume to contain a spanned record, the logic module:

1. Rereads the last block of the previous spanned record.
2. Rewrites the last block (truncated to the last segment of the previous spanned record, if necessary) to erase the remainder (if any) of the track or control interval.
3. Writes an eight-byte record-block descriptor word and one null segment on each remaining track or control interval on the current volume.
4. Attempts to put the entire spanned record on the next volume.

For *update* files, the logic module repositions the device to the first block of the logical record by using the pointer saved in GET processing. If the logical record spans extents, the extent sequence number that was also saved in GET processing is used to ensure that updating starts in the proper extent; that is, from the beginning of the record.

When *undefined* records are processed, PUT treats them as unblocked. You must provide any blocking desired. You must also determine the length of each record (in bytes) and load it in a register before issuing the PUT macro for that record. The register used for this purpose must be specified in the DTF RECSIZE operand.

An *update* DASD record may be read, modified, and written back to the same DASD location from which it was read. This is possible with all DASD devices. A card record may, with some devices, be read and then have additional information punched back into the same card.

When updating a file, one I/O area can be specified (using the IOAREA1 operand) for both the input

and output of a card record. If a second I/O area is required, it is specified with the IOAREA2 operand. For associated DTFCD files, however, two I/O areas are not allowed.

A PUT for a DASD file or for a combined card file must always be followed by a GET before another PUT is issued: GETs, however, can be issued as many times in succession as desired. When updating a disk file, the record is not actually transferred with the PUT but with the next GET for the file.

In the combined card file example of Figure 3-2 data is punched into the same card that was read. Information from each card is read, processed, and then punched into the same card to produce an updated record.

Processing Blocked Records

The GET/PUT macros allow you to store and retrieve logical records of a file without the need for coding blocking/deblocking routines; these functions are performed automatically by LIOCS whenever necessary. For example, each time you issue a GET macro in your program, the next logical record is made available for processing. Actual physical input/output is performed only when the next logical record must be obtained from the next block or control interval.

The GET macro is used to obtain logical records in physical sequence from a file on any device. Automatic record deblocking is included. As required, the system schedules the filling of input areas, deblocks records, and directs error recovery procedures.

Issuing a GET macro after the last record of an input file has been processed results in an end-of-file condition. The system also checks for end-of-volume conditions, and initiates automatic volume switching if an input file extends over more than one

```
Name     Operation   Operand                Column 72
FILEC    DTFCD                                      X
                     TYPEFLE=CMBND,                 X
                     IOAREA1=AREA,                  X
                     DEVADDR=SYS005,                X
                     RECFORM=FIXUNB,                X
                     IOAREA2=AREA2
         •
         •
         •
         GET         FILEC
         •
         •
         •
         PUT         FILEC
         •
         •
```

Figure 3-2. Combined card file example.

Record Format	Number of I/O Areas	Separate Work Area	Amount of Maximum Achievable Overlap
Unblocked	1	no	Overlap of the device operation only for buffered devices such as 1403, 1443, 2540. No overlap of magnetic tape, disk or unbuffered unit record devices.
		yes	Overlap processing of each record.
	2	no	Overlap processing of each record.
		yes	Overlap processing of each record.
Blocked	1	no	No overlap.
		yes	Overlap processing of first record of a block.
	2	no	Overlap processing of full block.
		yes	Overlap processing of full block.

Note 1: If UPDATE=YES is specified, no overlap occurs with DTFSD DASD files.
Note 2: The amount of effective overlap depends on the workload of the system.

Figure 3-3. Overlap of processing and I/O.

volume. When a file occupies more than one area on a DASD volume, automatic switching from one extent to the next is also performed.

The PUT macro releases logical records to the system for output, in physically sequential order. Automatic record blocking is included. As required, the system blocks records, schedules the emptying of output areas, and handles output error correction procedures where possible. The system checks for end-of-volume condition and performs automatic volume switching and label creation. References to other DASD extents are resolved.

A major feature of this level of sequential processing is the choice of using one or two input or output areas per file, with or without a separate workarea. Figure 3-3 indicates the amount of overlap that can be achieved with various combinations.

In a multiprogramming environment, the amount of overlap is impossible to predict because the program may loose control after I/O begins. Processing, however, may be faster in a "no overlap" situation because the path length of the logic module is shorter.

The processing of the logical record can be done in either a workarea or in an I/O area. Figure 3-4 contrasts the processing of a logical record with and

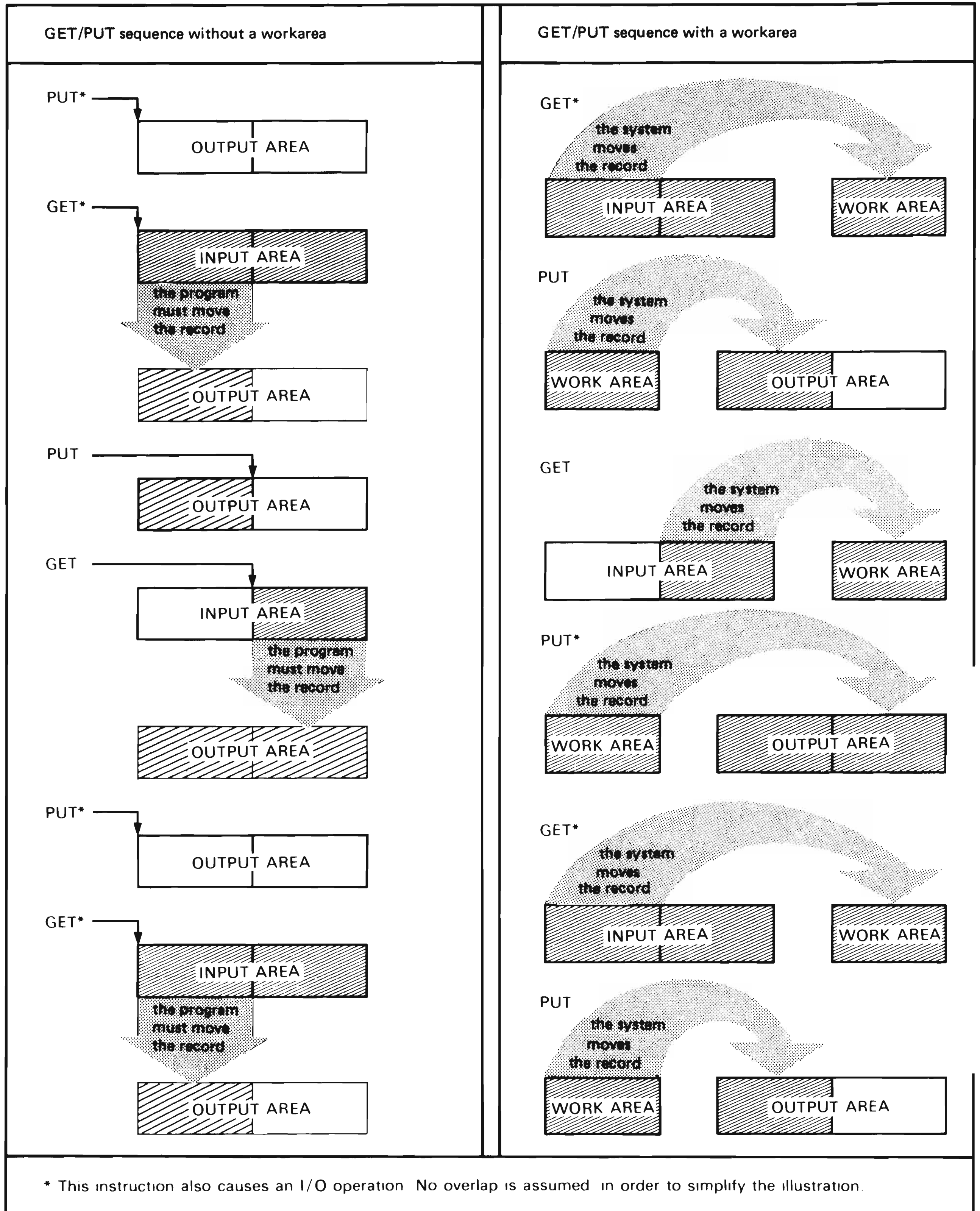


Figure 3-4. GET/PUT sequence with and without a workarea.

without a workarea; it shows a sequence of GET and PUT macros with and without a workarea.

If you issue GET or PUT with the filename as the only operand, GET provides the address of the logical record available in the input area; PUT provides the address of that part of the output area where the next logical record may be built.

If you specify, in addition to the filename, the workname operand, the logical record is actually transferred from the input area for GET to an area specified by the workname operand; with PUT, it is transferred from the specified area to the output area. The area referred to by the workname operand can be a separate workarea or part of an I/O area used as such.

Operating with GET and PUT using a separate workarea offers ease in coding, in maintaining the code, and in debugging. Operating with GET and PUT without a separate workarea provides faster performance, since there is no need for data transfer between the workarea and the I/O areas. A combination of GET with a workarea, PUT without a workarea can also be used. In this case, the workname operand of the GET macro specifies a register. The register contains a pointer, provided by the preceding PUT, to a location within the output area.

The following text points out some of the implications of your decision to use one or two input or output areas for a file, to use a workarea or not, and to process the records in the workarea or in the I/O area. Inasmuch as the following discussion deals with overlap, it is primarily significant for one-partition operation. When, in a multiple-partition system, a program in one partition must wait for an I/O operation to be completed, control is given to a program in another partition.

Our example assumes an input file whose records are to be processed and written to an output file. The records are assumed to be blocked. Actual input is performed only when the record requested belongs to a new block; actual output is performed only after completion of a whole block. I/O requests for all other records of the block do not result in actual I/O; they merely cause the pointer to the next logical record to be updated. If a workarea is used, the logical record is transferred from the workarea to the output area or from the input area to the workarea.

Recall the extra level of blocking present when CI format is used and that actual I/O operations take place into and out of the CI buffer. Successive GETs transfer logical blocks from a CI buffer to the input area, as required, meanwhile updating the pointer to the next logical record. Actual read operations take place only when the CI buffer, not the logical block, is depleted. Likewise, PUTs update the logical record pointer, transfer logical blocks to the output CI buffer, and write to the output file when the buffer is filled, all asynchronously and as required.

If you decide that you want only one input and one output area and no workarea, no overlap at all occurs. In this case, it would be advisable to process the records in the output area. This allows data transfer from the device to the input area to be started right after the last record of the block is moved to the output area and before it is processed.

The I/O time per record can be slightly reduced by increasing the blocking factor. The blocksize can be dynamically assigned during OPEN, via the DLBL statement, for blocked records.

Establishing a second input area and a second output area in our example makes an overlap possible. The amount of the effective overlap depends on the workload of the system. One block or control interval is processed while the transfer of another block between device and virtual storage is taking place.

Using a separate workarea together with one input and one output area has the great advantage that lies in the fact that writing and maintaining the code is less complicated, and debugging easier. You would issue GET, process the record in the workarea, and issue PUT. The input and the output files are connected to the same workarea. The name of the workarea (or, for self-relocating programs, a register containing the address of the workarea) would be specified as the workname operand with both macros.

Selective Processing of Blocked Records

Mostly, a program will process a file starting with the first logical record and proceed until end-of-file is signalled by IOCS. In these cases, processing blocked records or unblocked records is equally suitable to the application, especially since blocking or deblocking is performed automatically with the GET and PUT macros. For some special situations, however, the use of blocked records offers possibilities that do not exist when records are unblocked.

When processing a logical record of a block, you can have the system ignore the remaining records of that block and obtain the first logical record of the next block. For output, you can skip the remainder of the current block and place the next logical record as the first of the next block. When blocked spanned records are processed, you can bypass all subsequent records of the block being processed,

and obtain the first segment of the next logical record in the new block.

A case in which the application could benefit from these possibilities is, for example, a file that consists of several major groups of logical records. If each category started on a new block, it would be easy to locate any of the categories for selective processing. Only the first record of each block would have to be checked. To achieve this, you would use the RELSE (release) macro with input, and the TRUNC (truncate) macro with output.

The RELSE macro causes the following GET to ignore any logical records remaining in the current block and to obtain the first logical record of the following block. When spanned records are processed, RELSE causes the following GET to skip any subsequent records of the current block and makes the first record of the next block available.

The RELSE macro is used with blocked input records read from a DASD device, or with blocked spanned records read from, or updated on, a DASD device. This macro is also used with blocked input records read from magnetic tape.

If RELSE immediately precedes a CNTRL macro with the codes FSL or BSL (tape spacing for spanned records), then the FSL or BSL logical record spacing is ignored.

The symbolic name of the file, specified in the DTF header entry, is the only parameter required for the RELSE macro. It can be specified as a symbol or in register notation.

RELSE discontinues the deblocking of the present block of records, which may be of either fixed or variable length. RELSE causes the next GET macro to transfer a new block to the input area, or switch I/O areas, and make the first record of the next block available for processing. GET initializes the register or moves the first record to a work area.

For example, RELSE may be used in a job in which records on DASD or tape are categorized. Each category (perhaps a major grouping) is planned to start as the first record in a block. For selective reports, specified categories can be located readily by checking only the first record in each block.

The TRUNC macro is used with blocked output records written on DASD or magnetic tape. It allows you to write a short block of records. Blocks do not include padding. The TRUNC macro causes the following PUT macro to regard the output area as full and, subsequently, the next logical record to be placed into the following block. Thus, the TRUNC macro can be used for a function similiar to that of the RELSE macro for input records. That is, when the end of a category of records is reached, the last block can be written and the new category can be started at the beginning of a new block. IOCS provides for reading truncated blocks, so that reading a short block will not result in an error condition on input. The CLOSE macro truncates the last block of a file.

If the TRUNC macro is issued for fixed-length blocked DASD records, the DTF entry TRUNCS must be included in the file definition.

When the TRUNC is issued, the short block is usually written (on DASD or tape), and the output area is made available to build the next block. If the file resides on an FBA device, however, TRUNC will not necessarily cause a physical write to the FBA DASD unless PWRITE=YES is specified on the DTF. The last record written in the short block is the record that was built and included in the output block by the last PUT preceding the execution of the TRUNC macro. Therefore, if records are built in a work area and the program determines that a record belongs in a new block, TRUNC should be issued first to write the block. This should be followed by a PUT for this particular record to move the record into the new block. If records are built in the output area, however, you must determine if a record belongs in the block before you build the record.

Whenever variable-length blocked records are built directly in the output area, the TRUNC macro must be used to write a complete block of records. When a PUT is issued after each variable-length record is built, the output routines supply you with the space (number of bytes) remaining in the output area. From this, you determine whether your next variable-length record fits in the block. If it does not fit, issue the TRUNC macro to transfer the block and make the entire output area available to build the next record. The amount of remaining space is supplied in the register specified in the DTF VARBLD operand.

The name of the file must be specified as an operand for both the RELSE and the TRUNC macros.

Multivolume File Processing - Forcing End of Volume: Both the GET and the PUT macros check for the normal end-of-volume condition. If such a condition occurs, the system performs automatic volume switching.

You may also deliberately stop processing a file on a volume before the normal end-of-volume occurs, and resume processing the same file on the next volume. You can force end-of-volume and cause automatic volume switching by using the FEOV or FEOVD macro.

The FEOV macro forces the system to assume an end-of-volume condition on either an input or output *magnetic tape* file, thereby causing automatic volume switching.

When FEOV is issued for an input file, trailer labels are not checked. The header labels of the next volume, however, are verified. When FEOV is issued for an output file, trailer labels are created as required.

The FEOVD macro forces the system to assume an end-of-volume condition on either an input or output *DASD* file, thereby causing automatic volume switching. The operation is the same as for the FEOV macro, except that trailer labels are also processed for input.

The name of the file is required as an operand for both the FEOV and the FEOVD macros.

Processing Update Files: Files that are processed sequentially are normally either input or output files. With certain devices it is also possible to use the same file as both input and output file. In this case, you obtain a logical record from the file and, after processing, write the updated version of the record back into the original location of the file.

The devices that can have input and output file combined are:

- All types of DASD. Specify UPDATE=YES in the DTFSD declarative macro.
- IBM 1442 Card Read Punch, IBM 2520 Card Read Punch, IBM 2540 Card Read Punch equipped with the Punch-Feed-Read special feature. Specify TYPEFLE=COMBND in the DTFCD declarative macro.
- IBM 2560 Multifunction Card Machine, IBM 3525 Card Punch equipped with the Card Read special feature, IBM 5424/5425 Multifunction Card Unit. Specify the ASOCFLE and the FUNC operands in the DTFCD declarative macro.

Records are obtained from the file as usual by a GET macro. After the record has been processed, the next PUT causes the record to be returned to its original location in the file (DASD), or to be punched into the same card from which it was read.

Processing is done in the input area. After processing, the records are returned to the file from the input area. For card devices, the records are returned to the file by PUT; for DASD, the PUT sets an indicator which is used by the next GET (or CLOSE) to accomplish the transfer. The input area must not be modified between a PUT and the next GET.

If a workarea is used for the file, the records are returned by PUT from the workarea to the input area and then from the input area to the file. When control interval format is used, of course, the CI buffer is the primary input area and any user-specified input area is bypassed when a workarea is specified. For spanned records, you must have a workarea that is sufficiently large to hold the entire spanned record.

If a particular record does not require updating, a subsequent PUT may be omitted, except for the 2540, 2560, 3525, or 5424/5425.

Processing Work Files with SAM

A *work file* is a file used for both input and output. Typically, it is used to pass intermediate results between successive phases or job steps, but it can also be written, read, and rewritten within a single phase without being closed and reopened. By using the READ and WRITE work file macros, you can perform overlapped processing. That is, while your program is waiting for an I/O operation on a record to be completed, it can perform other operations that do not depend upon the presence of the record. You must use the CHECK macro to ensure that a READ or WRITE has been completed before allowing the program to continue.

By using the NOTE, POINTR, POINTS, and POINTW work file macros, you can do a certain amount of direct processing. That is, you can position the file to a specific point within the file and continue sequential processing from that point.

There are certain restrictions on the use of work files; for instance, they may be specified only as having unblocked records of fixed lengths or of undefined format. If you want to use blocked records with a work file, you must either code your own blocking and deblocking routines, or use CI format with an FBA device, or both. Automatic I/O area switching is not provided; your program must supply the address of your I/O area each time it issues a READ or WRITE macro. A work file must be contained on a single volume. You may not use magnetic tapes written in ASCII mode for work files. Both normal extents (type 1) and split extents (type 8) are supported for CKD disks, but only type 1 extents for FBA DASD.

In addition, if you use CI format, the work file logic module limits the number of logical blocks per control interval to no more than 255. This means that an error condition may occur if you attempt to use a CI-format work file that has been created or modified by other than SAM with the DTFSD TYPEFLE=WORK operand specified, as the creation

of the file may have caused more than 255 blocks to be put into a single CI.

Required DTF Macro Entries

Work files are defined by specifying the TYPEFLE=WORK operand on the DTF. The RECFORM operand must have either UNDEF or FIXUNB specified to define the file as 'undefined' or 'fixed-length unblocked'. (If you omit the RECFORM operand, FIXUNB is assumed.)

Retaining and Deleting a Work File

If you want to retain a DASD work file for later use, you must specify the DTFSD operand DELETFL=NO and make sure that the expiration date is not the current date. When this is done, the CLOSE routines do not delete the format-1 file extent label created by the open routines, and the file can be saved until the expiration date is reached.

To delete the file after use, *do not* specify the DELETFL=NO operand of DTFSD; the close routines will delete the format-1 label that was created when the file was opened. This allows another job requiring a work file to use the same extents and file name.

Opening a Work File

When a work file is opened, it is opened as an output file and the OPEN routines determine if standard labels are present. If the file is on DASD, file protection is ensured only if the labels are unexpired; you must supply label information with the job or by means of standard labels.

Because a work file is always opened as an output file, a DASD work file that is being reopened (as when you are using it to pass information to a second job step) causes an overlapping extent message to be printed to the operator. The operator can then delete the format-1 label, after which the open routines create a new label for the file, and the job continues.

If the work file is on tape, however, label information job control statements are not needed, and the DTFMT FILABL operand is ignored. If the tape does not contain standard labels, no labels are created for it. Trailer labels are not processed. If standard labels are present and the date has expired, a new label is created, consisting of a HDR1 followed by 76 blanks, which marks the file as a work file. If OPEN determines from the HDR1 label that the file already is a work file, the label is not rewritten. Trailer labels are not processed.

Sequential Processing of Work Files

Using READ and WRITE macros permits you to do sequential processing of work file records. Since work files are considered by SAM to be unblocked, when you issue a READ (for instance) for a logical record, you actually obtain a physical block from a magnetic tape or a CKD disk. If the records are in fact blocked, you are generally responsible for deblocking them. The reverse is true for the WRITE macro, of course: if you do your own blocking, the WRITE transfers the block to the I/O device; unblocked logical records are transferred to the tape or disk as single physical blocks.

If you use control interval format and select a CI size that will hold an integral multiple of your logical record size (plus the needed control information), SAM will block (and deblock) your logical records through a logical block in the CI buffer, and the FBA DASD control unit will transfer the logical block to (or from) the FBA DASD as one or more physical FBA blocks.

Use the CHECK macro to halt processing until a READ or WRITE I/O transfer is complete. By deferring a CHECK, you can overlap processing that is unrelated to a record with the I/O transfer of the record. Depending on circumstances, this can significantly increase throughput efficiency.

The format of the READ macro is as follows:

Name	Operation	Operand
[name]	READ	{filename|(1)},SQ,{area|(0)} [,length|,(r)|,S]

filename|(1): Specifies the name of the file from which the record is to be read and is always required. This name is the same as the name specified in the DTFMT or DTFSD header entry for the file. The name can be specified as a symbol or in register notation.

SQ (for sequential): Is always required.

area|(0): Specifies the name (as a symbol or in register notation) of the input area used by the file. If tape is to be read backwards, area must be the address of the rightmost byte of the input area.

length|(r)|S: Is used only for records of undefined format (RECFORM=UNDEF). To read only a portion of a record, specify the actual number (or a register containing the number) of bytes. Or, specify an S to indicate that the entire physical record should be read.

If the work file records are fixed length unblocked records (RECFORM=FIXUNB), this parameter must not be specified in the READ macro. In this case, the number of characters to be read is specified in the BLKSIZE operand. You can change this number (which is stored in the DTF table) at any time by referencing the halfword *filenameL.*

Before processing this data, you must make sure that the input operation for that block is complete. To do so, issue a CHECK macro after each READ macro.

After READ has retrieved all blocks of a file and discovers that no more data is available for processing, IOCS passes control to your end-of-file routine, whose address is specified in the DTFxx macro.

Because the READ macro has been designed for workfiles, multiple-volume support is not available. That is to say, a file to be processed by means of the READ macro must be contained on one volume. READ can also be used to read backwards from magnetic tape.

The WRITE macro requests that a block of data be transferred from an area in virtual storage to a file. It operates in the same fashion as READ, but in reverse order. Each WRITE must be followed by a CHECK, and the file to be processed must be contained on one volume.

The format of the WRITE macro is as follows:

Name	Operation	Operand
[name]	WRITE	{filename|(1)} ,{SQ|UPDATE},{area|(0)} [,length|,(r)]

filename|(1): Specifies the name of the file to which the record is to be written and is always required. This name is the same as the name specified in the DTFMT or DTFSD header entry for this file. The name can be written as a symbol or in register notation.

SQ|UPDATE: Specifies the type of WRITE to be executed. Always specify SQ for a magnetic tape file. For DASD work files, SQ produces a *formatting* write, UPDATE a *non-formatting* write.

When you are using control interval (CI) format, as with an FBA DASD, a non-formatting WRITE (with UPDATE) writes the current CI, while a formatting WRITE (with SQ) writes the CI and follows it immediately with a Software-End-Of-File (SEOF). When not writing in CI format, as with CKD disk, a formatting WRITE writes count, key, and data, while a non-formatting WRITE writes only data. An update WRITE should be preceded by a READ, WRITE, UPDATE, POINTR, or POINTW macro. A CLOSE macro (following an update write) protects the updated file by not writing an end-of-file record. If SQ is specified and a CLOSE immediately follows an OPEN (no formatting WRITE commands were issued), an end-of-file record is not written.

area|(0): Specifies the name, as a symbol or in register notation, of the output area used by the file.

length|(r): Is used only for records of undefined format (RECFORM=UNDEF). Length specifies the actual number (or register containing the number) of bytes to be written. If fixed-length unblocked records (RECFORM=FIXUNB) are written, length is not used in the WRITE macro.

The number of characters to be written is specified in the BLKSIZE entry. You can change this number, which is stored in the DTF table, at any time by referencing the halfword *filenameL.* For disk, the BLKSIZE entry in the DTF itself should not include eight bytes for the length of a count field.

The CHECK macro prevents data requested by a READ or a WRITE macro from being processed before its transfer is completed. In addition, it tests for errors and exceptional conditions that may have occurred during the data transfer. When necessary, control is passed to the appropriate exits (for error analysis and end-of-file) specified in the DTFxx macro for the file. Use the CHECK macro after each READ or WRITE before you issue any other macro for the same file, or before the contents of the input or output area in virtual storage are altered.

If the data transfer is completed without any error or other exceptional condition, CHECK returns control to the next instruction. If the operation results in a read error, CHECK processes the option specified in ERROPT. If CHECK finds an end-of-file condition, control is passed to the routine specified in EOFADDR.

Both READ and WRITE operate in a strictly sequential manner, starting either at the beginning of a file or at a given point, to which the file can be positioned by one of the POINTx macros (see below).

Selective Processing of Work Files

You can position a sequentially organized file to a specific block within the file and start sequential processing from this point. When you want to do this, be prepared to identify this block, that is, you must be able to indicate its position in the file.

During processing you may issue the NOTE macro. It returns information about the position of the

block just read or written. The format of the NOTE macro is the same as that of the CHECK macro. That is, the only operand is the name of the file being processed, which may be specified as a symbolic name or in register notation.

NOTE can be issued after a READ or a WRITE macro and after the I/O operation was checked for completion through use of a CHECK macro. It identifies a record on magnetic tape by its physical record number counted from the load point of the file. It identifies a record on CKD disk by its physical record number counted from the beginning of the track, and on FBA DASD by counting from the beginning of the file. For an output file on DASD, NOTE also returns the number of bytes of space left on the track or in the control interval.

To NOTE a desired record successfully, the POINTR, POINTS, or POINTW macro must not be issued between the CHECK and NOTE.

For magnetic tape, the last record read or written is identified by the number of physical records read or written in the specified file from the load point. The physical record number is returned in binary in the three low-order bytes of register 1. The high-order byte contains binary zero.

You must store the identification so that it can be used later in either a POINTR or POINTW macro.

For disk, if a READ precedes the NOTE, the record identified is the last record read. If a WRITE precedes the NOTE, the record just written is the identified record.

For CKD DASD, the identification is returned in register 1 in the form cchr, where

cc= cylinder number,

h= track number,

r= record number within the track.

cc, h, and r are binary numbers. If NOTE follows a READ or WRITE to a disk file, the unused space remaining on the track following the end of the identified record is returned in register 0 as the binary number 00nn.

For FBA DASD, because the unit of data transfer is a control interval (CI) instead of a physical block, information returned when a NOTE macro is issued is different.

Register 0 contains the length of the longest logical block that can be guaranteed to fit in the CI following the NOTEd logical block. A logical block three bytes longer than the returned value will fit in the CI if it is of the same length as both the NOTEd block and the block preceding the NOTEd block. (This means that if the CI were exactly filled when the NOTE was issued, a value of -3 would be passed back in register 0.)

Register 1 contains a record identifier that is an address relative to the beginning of the file. The first three bytes contain the relative CI number of the current CI within the file (with origin 0). The fourth byte is the relative block number of the logical block within the current CI (with origin 1).

You must construct a six-byte field and store in it the identification of the record (from register 1 after NOTE) and the remaining capacity (from the low-order two bytes of register 0 after NOTE) so that it can be used later in a POINTR or POINTW macro to find the NOTEd record again. The remaining capacity is required only if the POINTR or POINTW will be followed by a WRITE SQ or another NOTE while still positioned on the same CI or track that was pointed to.

The POINTS macro causes a file to be repositioned to the beginning. For magnetic tape files, POINTS causes a rewind of the tape to the load point and the positioning of the tape to the first data block; labels are bypassed. For DASD files, POINTS positions the file to the lower limit of the first extent.

On DASD, a POINTS macro followed by a WRITE SQ causes the new record to be written and the remainder of the track or control interval to be erased. POINTS should not be followed by a WRITE UPDATE.

An example of POINTS with workfile processing is given in Figure 3-5.

```
      L          12,LENGTH            A
X     WRITE      F,SQ,OUT,(12)        B
      •
      •                               C
      •
      CHECK      F                    D
      •
      •
      •
      BNZ        X
      POINTS     F                    E
Y     READ       F,SQ,IN,S            F
      •
      •
      •
      CHECK      F                    G
      •
      •
      BNZ        Y
      EOJ
```

```
A   Load the length of the record
B   Write a record
C   Process data unrelated to OUT
D   Wait until the record is written
E   Reposition to the beginning of the
    file
F   Read physical record 1
G   Wait until the record is read
```

Figure 3-5. Example of POINTS macro with workfile processing.

The POINTR macro is used to position the file to a specific block. The block can then be read by a subsequent READ macro. A series of READ macros following a POINTR will pick up blocks sequentially, starting with the block specified in the POINTR macro. The address to be specified in the POINTR macro can be obtained from the result of a previously issued NOTE macro.

For magnetic tape, POINTR repositions the file to read the record that was read or written immediately before the NOTE that was used to create the record identification field. For magnetic tape, a WRITE must not follow POINTR.

For disk, POINTR repositions the file to read the record identification returned when a previous NOTE macro was issued. If a WRITE UPDATE follows the POINTR macro, the noted record is overwritten. If a WRITE SQ follows the POINTR macro, the record after the noted record is written, and the remainder of the track or CI is erased. On FBA devices only, an SEOF is written immediately after the current CI.

Some programs using disk work files may include multiple WRITE macros following a NOTE macro. If a POINTR macro is used with a DASD and the work file records are in undefined format, there may be occasions when a replacement record longer than the original record remains as the last record on the track or control interval when the next WRITE is performed. The replacement record is written as the first record on the next track or control interval of the file.

The POINTW macro is used to position the file to the block following the one specified. A block can then be written to that location by a subsequent WRITE macro. A series of WRITE macros following a POINTW will write blocks sequentially, starting at the location following the block specified in the POINTW macro. The address to be specified can be obtained from the result of a previously issued NOTE macro.

For magnetic tape, POINTW repositions the file to read or write a record after the one previously identified by the NOTE.

For disk, POINTW repositions the file to write at the record location that was read or written immediately before the last NOTE macro was issued. If a WRITE UPDATE is issued, the noted record is overwritten. If a WRITE SQ is issued, the record following the noted record is written and the remainder of the track or CI is erased. On FBA devices only, a SEOF is written immediately after the current CI. A READ macro can follow the POINTW macro, in which case the record identified by the NOTE is the record read.

Some programs using disk work files may include multiple WRITE macros following a NOTE macro. If a POINTW macro is issued to a CKD DASD and the work file records are in undefined format, there may be occasions when a replacement record longer than the original record cannot be written in the space available on the track or CI. In this case, when the next WRITE is performed, the original record remains as the last record on the track. The replacement record is written as the first record on the next track or CI of the file.

For files on DASD, the WRITE macro can be used after either POINTW or POINTR. If you specify WRITE it will be executed as a WRITE UPDATE, and the block specified in the POINTx macro will be overwritten. Otherwise WRITE is considered a WRITE SQ (sequential), and the block will be written after the one specified in the POINTx macro, and the remainder of the track will be erased.

You will have gathered from the preceding text that SAM allows for direct processing through these macros. This is true to a certain extent, provided the following limitations are recognized:

In SAM, the READ and WRITE macros can be applied only to files that are completely contained on one single volume.

Direct processing on magnetic tape, although possible, is inefficient, since many blocks may have to be bypassed before a block wanted for processing is reached. Furthermore, on magnetic tape, direct processing is restricted to reading. Writing should be done sequentially, because it takes some time before actual writing starts; when actual writing stops, it takes some time before the medium comes to a complete standstill. The interrecord gaps between blocks allow for this, but frequent overwriting of blocks may cause an interrecord gap to be too short or too long, and may even affect a following block.

Deactivating a File After Processing

The FEOV and FEOVD macros force an end-of-volume condition before it actually occurs. Use an FEOV/D macro to indicate to SAM that processing on one volume is finished, but that reading or writing will take place on the following volume.

The completion macro CLOSE must be used after processing has been completed. CLOSE ends the association of the logical file declared in your program with a specific file on an I/O device.

Forcing End-Of-Volume

Force end-of-volume by using an FEOV macro for magnetic tape files or FEOVD for DASD files.

When SAM macros are used for a file, FEOV initiates the same functions that occur at a normal end-

of-volume condition, except for checking of trailer labels.

For an input tape, FEOV immediately rewinds the tape (as specified by REWIND) and provides for a volume change (as specified by the ASSGN cards). Trailer labels are not checked. FEOV then checks the standard header label on the new volume and allows you to check any user-standard header labels if LABADDR is specified. If nonstandard labels are specified (FILABL=NSTD), FEOV allows you to check these labels as well.

For an output tape, FEOV writes

- A tapemark (two tapemarks for ASCII files.)
- A standard trailer label and user-standard labels (if any).
- A tapemark.

If the volume is changed, FEOV then writes the header label(s) on the new volume (as specified in the DTFMT REWIND, FILABL, LABADDR operands, and the ASSGN cards). If nonstandard labels are specified, FEOV allows you to write trailer labels on the completed volume, and header labels on the new volume, if desired.

When FEOVD is issued for an output file assigned to a CKD DASD, a short last block is written, if necessary, with an end-of-file record containing a key length of 0 (indicating end of volume). If the output file is assigned to an FBA device, SAM writes a Software End-Of-File (SEOF), which is a control interval containing only zeros. An end-of-extent condition is posted in the DTF. When the next PUT is issued for the file, all remaining extents on the current volume are bypassed. The first extent on the next volume is then opened, and normal processing continues on the new volume.

If the FEOVD macro is followed immediately by the CLOSE macro, the end-of-volume marker is rewritten as an end-of-file marker, and the file is closed as usual.

Closing a File

CLOSE must be issued to deactivate all files previously activated by means of an OPEN macro, with the exception of console (DTFCN) files. No deactivation of console files is necessary, and the CLOSE macro must not be issued for them.

After you issue a CLOSE macro, no further commands can be issued to the file unless it is reopened. Sequential DASD files cannot be successfully reopened for output unless the DTFSD table is saved before the file is first opened and then restored between closing the file and reopening it again as an output file.

A CLOSE normally deactivates an output file by writing an EOF record and output trailer labels, if any. CLOSE sets a bit in the format-1 label to indicate the last volume of the file. A file may be deactivated at any time by issuing CLOSE. Up to 16 files may be deactivated with one CLOSE.

Note that if you issue CLOSE to a magnetic tape input file that has not been opened, the option specified in the DTF REWIND operand is performed. If you issue CLOSE to an unopened *output* magnetic tape file, no tapemark or labels are written, and no REWIND options are performed.

As with an OPENR macro, you must use the CLOSER macro if your program is to be self-relocating. The CLOSE and the CLOSER macros are essentially the same with the exception that when CLOSER is specified, symbolic addresses that are generated are self-relocating. Throughout the manual the term CLOSE refers also to CLOSER, unless stated otherwise.

Non-Data Device Operations

The CNTRL (control) macro provides commands for magnetic tape units, card devices, printers, and optical readers. For DTFSD files, CNTRL is treated as a No-Op.

Commands apply to physical nondata operations of a unit and are specific to the unit involved. They specify such functions as rewinding tape, card stacker selection, and line spacing on a printer. For optical readers, commands specify marking error lines, correcting a line for journal tapes, document stacker selecting, or ejection and incrementing documents. The CNTRL macro does not wait for completion of the command before returning control to you, except when certain mnemonic codes are specified for optical readers. The permitted mnemonic codes are device-dependent and are discussed in later sections of this manual.

The CNTRL macro must not be used for printer or punch files if the data records contain control characters and the entry CTLCHR is included in the file definition. Whenever CNTRL is issued in your program, the DTF CONTROL operand must be included (except for DTFMT and DTFDR) and CTLCHR must be omitted. If control characters are used when CONTROL is specified, the control characters are ignored and treated as data.

Logic Modules for SAM

Unless a file is assigned to a DASD device, or unless it uses extended printer buffering for the IBM 3800 printer, a logic module must be assembled by the programmer. The logic modules for DASD and for

extended printer buffering are automatically loaded into the SVA (system virtual area) at IPL time and are linked to the problem program when the file is opened. The logic modules for the files assigned to other than DASD devices or the IBM 3800 must be assembled by the programmer from a source statement library, supplied by IBM. This is a one-time process. Once assembled, the logic modules can be stored in the relocatable library.

The logic modules can be link-edited with any problem program that requires them. If preferable, however, they can also be assembled along with the user's program and included in the same object module.

The logic module for a specific type of file in a particular problem program is assembled on a selective basis, according to the requirements specified by the user. The requirements vary depending on the characteristics of the file, the type of device on which the file resides, and the operations to be performed on the file.

The functions that a particular module is to provide are specified in the parameters of the xxMOD macro. There are different xxMOD macros for different device types (see Figure 3-6). For console files, no logic module is required.

The characteristics of the file are specified as parameters of the DTFxx macro, which generates a DTF table, serving as a link between the user's program and the logic module for a certain file. There are different DTFxx macros for different device types (see Figure 3-6).

Device Type	xxMOD Macro	DTFxx Macro
Card, and 3881 Optical Mark Reader	CDMOD	DTFCD
Console	—	DTFCN
Device Independent	DIMOD[2]	DTFDI
3886 Optical Character Reader	DRMOD DFR DLINT	DTFDR
Diskette	DUMODFx	DTFDU
Magnetic Character Reader, Optical Character Reader	MRMOD	DTFMR
Magnetic Tape	MTMOD	DTFMT
Optical Reader, Optical Page Reader	ORMOD	DTFOR
Printer	PRMOD[1]	DTFPR
Paper Tape	PTMOD	DTFPT
Sequential DASD	—	DTFSD

1 Not needed for 3800 with extended printer buffering.

2 Not needed for DASD devices.

Figure 3-6. Declarative macro instructions for SAM.

DAM (Direct Access Method)

The Direct Access Method is a flexible access method provided specifically for use with CKD direct access storage devices. Some of the features of these devices are:

- Flexible record referencing, either to physical track and record address (record ID) or to record key (control field of the physical block).
- Ability to search sequentially through an area for a physical block, using a minimum of central processing unit time.

DAM does not include elaborate routines for handling file maintenance functions such as adding records to existing files, handling overflow records, locating synonym records, and deleting records. These functions are entirely the user's responsibility. This may seem a disadvantage, but, once a user has determined the way he will handle his data, DAM will prove to be a flexible access method. High-level programming languages, on the other hand, may not be able to support the devices fully, because of the restricted nature of the languages themselves; high-level language programmers should consult their language reference manual in order to learn about the device features that are under their control.

With the Direct Access Method you can process records in random order. The records may be read or written; they may be updated, or they may be replaced.

DAM supports the following DASD equipment:

IBM 2311 Disk Storage Drive
IBM 2314 Direct Access Storage Facility
IBM 2319 Disk Storage
IBM 3330 Disk Storage
IBM 3340 Disk Storage
IBM 3344 Direct Access Storage
IBM 3350 Direct Access Storage

For programming purposes, the 3344 can be regarded as being identical to the 3340. The 3350 can be regarded as either a 3350 (operating in 'native' mode) or a 3330 (in '3330-compatible' mode). For information about the device characteristics of all the DASDs listed above, see the appropriate component description manuals.

DAM can be applied to all record formats of VSE. When record spanning is used, the segmentation of the logical records and their reassembly is performed by LIOCS routines whenever necessary. The records can be written with or without a key area. When records in a file have keys that are to be processed, every record must have a key, and all keys must be of the same length. Records without keys

are identified by their position in the physical sequence of a given track.

DAM uses one I/O area for a file. To determine the size of the I/O area, the length of the data area and the use of the count and key areas as well as the control information must be taken into account.

DAM processes only unblocked records; that is, a physical block is regarded by IOCS as containing one logical record. If blocking is desired, the blocking/deblocking must be done by the programmer in the problem program.

Blocking records can hardly be recommended for DAM. Blocking logical records without keys amounts to greater difficulty in establishing an efficient randomizing algorithm. Although it may seem advantageous as far as storage utilization is concerned, it may have an adverse effect on the time needed to locate a specific record. On the other hand, blocking records with keys makes sense only if the key sequence and the physical sequence of the logical records coincide. Such an organizational requirement goes beyond the requirements met by an ordinary direct access file. As a matter of fact, this kind of organization is employed more adequately in ISAM, to be described later.

With DAM you can process DASD records in random order. You specify the address of the record to IOCS and issue a READ or WRITE macro to transfer the specified record.

Variations in the parameters of the READ or WRITE macros permit records to be read, written, updated, or replaced in a file.

Whenever DAM is used, the file must be defined by the declarative macro DTFDA (Define The File for Direct Access). The detail entries for this macro are described later in this book. In order to understand the use of some of these entries, however, it is necessary to provide information about how DAM processing uses them.

Record Types

DASD records that will be processed by DAM can exist on the DASD in either of two formats: with a key area, or without.

With key area:

Count	Key	Data

Without key area:

Count	Data

When processing spanned records, this format applies only to the first segment. For additional information on spanned records, see *VSE System Data Management Concepts,* as listed in the Preface.

Whenever records in a file have keys that are to be processed, every record must have a key and all keys must be the same length.

When the DTFDA KEYLEN operand is not specified for a file, IOCS ignores keys, and the DASD records may or may not contain key areas. A WRITE ID or READ ID reads or writes the data portion of the record. However, when KEYLEN is not specified in the DTF for a file, WRITE AFTER cannot be used to extend a file that has keys.

IOCS considers all records as unblocked; if you want blocked records, you must perform your own blocking and deblocking. Records are also considered to be either fixed, variable, or undefined length. A spanned record indicates variable blocks where the size of each segment is a function of the track size and record size. The record size is set by a formatting WRITE macro (WRITE AFTER). All the variable record segments of a given spanned record are logically contiguous. The type of records in the file must be specified in the DTFDA RECFORM operand. Whenever records specified as undefined are written, you must determine the length of each record and load the length in a register (specified by the DTFDA RECSIZE operand) before issuing the WRITE macro for that record.

I/O Area Specification

The DTFDA IOAREA1 operand is available to define an area of virtual storage in which records are read on input or built on output.

Format

The format of the I/O area is determined at assembly time by the following DTFDA operands: AFTER, KEYLEN, READID, WRITEID, READKEY, and WRITEKY. Figure 3-7 shows the DTFDA macro entries and the I/O areas that they define. The information in this figure should be used to determine the length of the I/O area specified in the BLKSIZE operand. The I/O area must be large enough to contain the largest record in the file. If the DTF used requires it, the I/O area must include room for an 8-byte count field. The count is provided by IOCS.

Contents

Figures 3-7 and 3-8 give a summary of what the contents of IOAREA1 are for the various types of DTFDA macros. These contents are provided by, or to, IOCS when an imperative macro is issued. When you build a record, you must place the contents

DTFDA MACRO ENTRIES						I/O AREA DEFINED
AFTER	KEYLEN	READID	WRITEID	READKEY	WRITEKY	
X	X	•	•	•	•	COUNT / KEY / DATA; Length (Bytes): 8, KEYLEN=n, Largest Record; BLKSIZE = n; IOAREA1
X		•	•			COUNT / DATA; Length (Bytes): 8, Largest Record; BLKSIZE = n; IOAREA1
	X	□	□	•	•	KEY / DATA; Length (Bytes): KEYLEN=n, Largest Record; BLKSIZE = n; IOAREA1
		□	□			DATA; Length (Bytes): Largest Record; BLKSIZE = n; IOAREA1
	X			□	□	

X = Specified

● = May also be specified

□ = Of two entries, one and/or the other is specified

Figure 3-7. I/O area for different DTFDA macro operands for fixed unblocked and undefined record formats.

shown in Figures 3-7 and 3-8 in the appropriate field of the I/O area. For example, if the DTF used for the file resulted in the uppermost format shown in Figure 3-7, the data would be located to the right of the count or key area.

As opposed to fixed unblocked and undefined records, the IOAREA1 for variable length and spanned unblocked records is independent of the DTFDA macro entries. If you specify the KEYLEN entry of the DTFDA macro, the key is transmitted to or from the field you specified on the KEYARG entry. The count field, if desired, is taken from an area reserved automatically by logical IOCS.

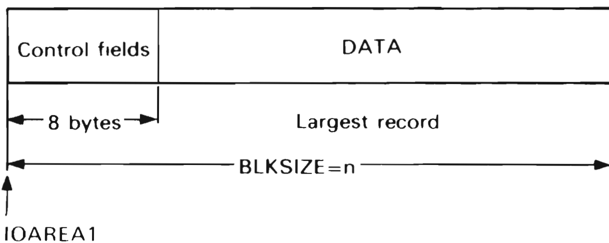


Figure 3-8. I/O areas for variable length and spanned unblocked DTFDA record format.

The control fields are built by logical IOCS except for the case when you create your file or add records to it by using the WRITE AFTER macro. You must, in that case, insert the data length of the record (plus four) into the 5th and 6th bytes of the control fields. When you read a variable length or spanned unblocked record these bytes will contain the length of the record. When updating records, you should not change any parts of the control fields.

The maximum length of a logical record plus its key and control fields, if any, is shown in Figure 3-9.

Device	RECFORM	
	FIXUNB VARUNB UNDEF	SPNUNB
2311	3625	32767
2314, 2319	7294	32767
3330/3333	13,030	32767
3340	8,535	32767
3350	19,069	32767

Figure 3-9. Maximum length of DTFDA records including key and control fields.

Creating a File or Adding Records

Your program can preformat a file or an extension to an existing file in one of two ways depending on the type of processing to be done. If the WRITE AFTER macro is used exclusively, the WRITE RZERO macro is enough for preformatting the tracks. If nonformatting functions of the WRITE macro are used, the tracks should be preformatted with the IBM-supplied Clear Disk utility program. The Clear Disk utility also resets the capacity record to reflect an empty track.

In addition to reading, writing, and updating records randomly, DAM permits you to create a file or write new records on a file. When this is done, all three areas of a DASD record are written: the count area, the key area (if present), and the data area. The new record is written after the last record previously written on a specified track. The remainder of the track is erased. This method is specified in a WRITE macro by the parameter AFTER.

IOCS ensures that each record fits on the track specified for it. If the record fits, IOCS writes the record. If it does not fit, IOCS sets a no-room-found indication in your error/status byte specified by the DTFDA ERRBYTE operand. If WRITE AFTER is specified, IOCS also determines (from the capacity record) the location where the record is to be written.

Whenever the AFTER option is specified, IOCS uses the first record on each track (R0) to maintain updated information about the data records on the track. Record 0 (Figure 3-10) has a count area and a data area, and contains the following:

Count Area:

- Flag (normally not transferred to virtual storage)
- Physical Identifier
- Key Length (KL)
- Data Length (DL)

Data Area:

- Physical ID of last record written on track (cchhr)
- Number of unused bytes remaining on track
- Flag (used by operating systems other than VSE)

Each time a WRITE AFTER macro is executed, IOCS updates the data area of this record.

Locating Data: Reference Methods

DAM requires two references for all read or write operations, the *track* reference and the *record* reference. The track reference may be either the actual *physical* DASD address, which specifies the location of the track, or the *relative* track address, which specifies the position of the track in relation to the beginning of the file. The record reference may be either the record key (if the records contain key areas) or the record identifier (ID).

IOCS seeks the specified track, searches it for the individual record, and reads or writes the record as indicated by the macro. If a specified record is not found, IOCS sets a no-record-found indication in your error/status byte specified by the DTFDA ERRBYTE operand. This indication can be tested and appropriate action can be taken to suit your requirements.

Multiple tracks can be searched for a record specified by key (SRCHM). If a record is not found after an entire cylinder (or the remainder of a cylinder) is searched, an end-of-cylinder bit is turned on instead of the no-record-found bit in ERRBYTE.

When an I/O operation is started, control returns immediately to your program. Therefore, when the program is ready to process the input record, or build the succeeding output record for the same file, a test must be made to ensure that the previous transfer of data is complete. Do this by issuing a WAITF macro.

After a READ or WRITE macro for a specified record has been executed, IOCS can make the ID of the next record available to your program. The WAITF macro should be issued to assure that the data transfer is complete. You must set up a field (in which IOCS can store the ID) to request that IOCS supply the ID. You must also specify the symbolic address of this field in the DTFDA IDLOC operand.

When record reference is by key and multiple tracks are searched, the ID of the specified record (rather than the next record) is supplied. The function of supplying the ID is useful for a random updating operation, or for the processing of successive DASD records. If you are processing consecutively on the basis of the next ID and do not have an end-of-file record, you can check the ID supplied by IOCS against your file limits to determine when the end of the file has been reached.

Track Reference

To provide IOCS with the track reference, you set up a track reference field in virtual storage, assign a name in the DTFDA SEEKADR operand, and determine by DTFDA operand specifications which type of addressing to use. Before issuing any READ or WRITE macro for a record, you must store the proper track identifier in either the first seven hexadecimal bytes (mbbcchh), or the first three hexadecimal bytes (ttt), or the first eight zoned decimal (tttttttt)

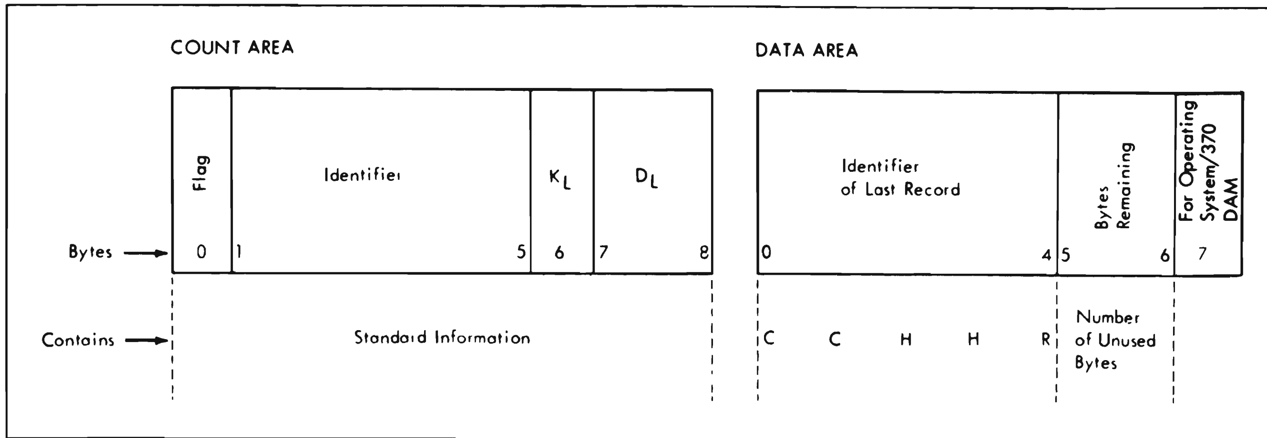


Figure 3-10. Contents of record 0 for capacity-record option.

bytes of this field. The latter two track references, along with the DSKXTNT and RELTYPE operands, indicate that relative addressing is to be performed. Thus, instead of providing the exact physical track location (mbbcch), only the track number relative to the starting track of the file need be provided. If these operands are omitted, the physical track location is assumed.

The fields for each of these track reference methods are shown in Figure 3-11. For reference to records by record number, r of rr is used (see "Identifier (ID) Reference," below). When the READ or WRITE is executed, IOCS refers to this field to select the specific track on the appropriate DASD.

Physical Track Addressing: An actual physical DASD address can be shown as an 8-byte binary address in the form **mbbcchhr**.

m identifies the volume. If a single file extends over more than one volume, the physical units must be assigned (in EXTENT job control statements) to a sequential set of symbolic unit numbers. The value of **m** is always 0 for the first volume, 1 for the second, 2 for the third, etc.

For example, a single logical file located on three volumes could be assigned to the logical unit numbers SYS002, SYS003, and SYS004. Here, **m**=0 refers to SYS002, **m**=1 refers to SYS003, and **m**=2 refers to SYS004.

The value of **m** is never actually read or written on the storage device. It references the proper element in the LUB table.

bb is a 2-byte reserved field; it is set to zeros.

cc is a 2-byte field that contains the cylinder number in binary form.

hh is a 2-byte field containing the head number in binary form in the second byte. The first byte is reserved.

r is the record number within a track. This 1-byte field can contain a binary value of 0 to 255 to identify the physical location of a record on the track. This field is not used when records are referenced by record key.

The 8-byte DASD addresses described above are used either as a starting point for a search on record key (control field) or as the actual address for a read or write operation. When searching for a key, you have the option of specifying that the search be only within the specified track (**hh**) or from track to track starting at the address given and continuing either until the record is found or until the end of the cylinder (**cc**) is reached.

Relative Track Addressing: The required DASD address may also be given as a relative address, which is then converted by IOCS to an actual address. Relative track addressing is more convenient to use than the physical address for the following reasons:

- The data file is treated logically as if it were located in one continuous area, although it may occupy several non-adjacent areas.
- You need to know only the relative position of the data within the file; its actual physical address is not required. This is especially advantageous if you plan to move the file from one location to another. In such cases the relative addressing scheme remains the same, and the actual addresses are automatically converted by IOCS.

You may specify the relative address either in hexadecimal (in the form **tttr**), or in zoned decimal (in the form **tttttttrr**). The hexadecimal notation

Bytes	Decimal Identifier	Contents in Zoned Decimal	Information
0-7	ttttttt	0-16,777,215	Track number relative to the first track of the file.
8-9	rr	0-99	Record number relative to the first record on the track. If reference is by key, rr should be zero.

Bytes	Hexadecimal Identifier	Contents in Hexadecimal	Information
0-2	ttt	0-FFFFFF	Track number relative to the first track of the file.
3	r	0-FF	Record number relative to the first record on the track. If reference is by key, r should be zero.

Bytes	Physical Identifier	Contents in Hexadecimal	Information
0	m	00-FF	Number of the volume on which the record is located. Volumes and their symbolic units for a file must be numbered consecutively. The first volume number for each file must be zero, but the first symbolic unit may be any SYSnnn number. The system references the volume by adding its number to the first symbolic unit number. **Example:** The first extent statement // EXTENT SYS005,... and m=0 result in the system referencing SYS005.
1-2	bb	0000	
3-4	cc	0000-00C7 (2311,2314,2319) 0000-0193 (3330-1,-2,3333) 0000-0327 (3330-11) 0000-015B (3348 model 35) 0000-02B7 (3348 model 70) 0000-022A (3350)	The maximum number of the cylinder in which the record can be located is: for 2311, 2314, 2319: 199 for 3330-1,-2, 3333: 403 for 3330-11: 807 for 3340 with 3348 model 35: 347 for 3340 with 3348 model 70: 695 for 3350: 554 These two bytes (cc), together with the next two (hh), provide the track identification. DAM does not permit the use of different data module sizes in a multivolume file on a 3340.
5-6	hh	0000-0009 (2311) 0000-0013 (2314) 0000-0012 (3330) 0000-0012 (3333) 0000-000B (3340) 0000-001D (3350)	The number of the read/write head that applies to the record. The first byte is always zero and the second byte specifies one of the disk surfaces in a disk pack.
7	r	0-FF	Sequential number of the record on the track. **Note:** r=0 if reference is by key.

Figure 3-11. Types of track reference fields.

requires four bytes, while the zoned decimal notation requires ten bytes.

ttt or **ttttttt** represents the track number relative to the beginning of the file.

r or **rr** represents the record number on the track.

Please note that the addressing techniques described above are used by the system, and can be applied in assembler language. Addressing in a high-level programming language, such as COBOL or PL/I, may be different. Information about DASD addressing in a high-level programming language should be obtained from the appropriate language reference manuals.

For certain types of operations, you can request the system to return the actual record address (ID) of the block read or written, or of the block following the one just read or written. This returned ID can be used to either read or write a new record, or to update the one just read and write the updated record back to the same location.

The format of the returned ID is the same as the format of the DASD address used for locating data, namely **mbbcchhr**, **tttr**, or **tttttttrr**.

Record Reference

DAM allows records to be specified by either record key or record identifier.

Key Reference: If records contain key areas, the records on a particular track can be randomly

searched by their keys. This allows you to refer to records by the logical control information associated with the records, such as an employee number, a part number, a customer number, etc.

For this type of reference you must specify the name of a key field in virtual storage in the DTFDA KEYARG operand. You then store each desired key in this field.

Identifier (ID) Reference: Records on a particular track can be randomly searched by their position on the track, rather than by control information (key). To do this, use the record identifier. The physical record identifier, which is part of the count area of the DASD record, consists of five bytes (cchhr). The first four bytes (cylinder and head) refer to the location of the track, and the fifth byte (record) uniquely identifies the particular record on the track. You may, however, use the relative track notation instead of cylinder and head notation if specified in the DSKXTNT and RELTYPE operands. When records are specified by ID, they should be numbered in succession (without missing numbers) on each track. The first data record on a track should be record number 1, the second number 2, etc.

Whenever records are identified by a record ID, the r-byte of the track-reference field (see Figure 3-10) must contain the number of the desired record. When a READ or WRITE macro that searches by ID is executed, IOCS refers to the track-reference field to determine which record is requested by the program. The number in this field is compared with the corresponding fields in the count areas of the disk records. The r-byte (or bytes) specifies the particular record on the track.

Locating Free Space

DASD design allows the operating system to locate track space that has not yet been used. For this purpose, DAM maintains a capacity record as a part of record zero on each track.

When a record must be written, the system will do this:

- It reads the data portion of record zero (the capacity record).
- It determines whether or not there is space on the track for the record.
- If the new record fits, it writes it onto the track as a new last record and updates the capacity record.
- If there is not enough space on the track, it notifies the problem program. An overflow routine in the problem program may then become active.

This design makes a randomizing problem less critical than in the past, when every single record was supposed to have its unique address. Each synonym resulting from a conversion algorithm resulted in an overflow record. Now the conversion algorithm may randomize to a track address, and more than one record may have the same track address assigned by the algorithm.

The capacity record is not always used. The description of the WRITE macro explains when it is used.

The capacity record is updated for each record that fills empty space on a track. When a record is deleted, however, the capacity record does not show it as empty space. Space that is 'free' because a record has been deleted can be recognized as such only by the user. You may, for example, do this: When you delete a record from a direct access file that has records with key areas, you may search on key to read the block and request that the record ID be returned. You then use this ID to write blanks or zeros (or whatever unique identification is acceptable to mark the deleted key and data area) to this location. If you later want to re-use deleted blocks for data, you may randomize the key of a new logical record to a starting location, make a search for a blank or zeroed key, and use the ID returned to write the new record with the new key into the same location.

Logic Modules for DAM

Four preassembled superset logic modules, supplied by IBM and loaded into the SVA during IPL, will be linked to the DTF when the file is OPENed. These logic modules are fully reentrant so that one copy of a logic module can be used by all requestors having the type of file for which the logic module was generated. Any other logic module referenced by the DTF will be ignored.

ISAM (Indexed Sequential Access Method)

The Indexed Sequential Access Method is a *file management system* developed for use with CKD DASD; logical records are organized by ISAM on the basis of a collating sequence determined by their keys.

ISAM supports the following devices:

IBM 2311 Disk Storage Drive
IBM 2314 Direct Access Storage Facility
IBM 2319 Disk Storage
IBM 3330 Disk Storage
IBM 3340 Disk Storage
IBM 3344 Direct Access Storage
IBM 3350 Direct Access Storage when operating in 3330-1 compatibility mode

For programming purposes, the 3344 can be regarded as being identical to the 3340. The 3350 is not supported by ISAM when operating in either 3350 mode or 3330-11 compatibility mode, nor is the 3330-11 supported in native mode. For information about the device characteristics of all the DASDs listed above, see the appropriate component description manuals.

With ISAM, you can process DASD records in either random or sequential order. For random processing, you supply the key (control information) of the desired record to ISAM and issue a READ or WRITE macro to transfer the specified record. For sequential processing, you specify the first record to be processed and then issue GET or PUT macros until all desired sequential records are processed. The successive records are made available in sequential order by key. Variations in macros permit:

- Creating a DASD file.
- Reading, adding to, or updating a DASD file.

Whenever ISAM is used, the file must be defined by the declarative macro DTFIS (Define The File for Indexed Sequential organization). The detail entries for this macro are described later in the book. In order to understand the use of some of these entries, however, it is necessary to provide information about how ISAM processing uses them.

As a file management system, ISAM takes care of the data organization where SAM and DAM do not. For instance, handling overflow when inserting records on an existing file and retrieving those records later is managed by ISAM routines. As a result, the management of ISAM data requires little I/O programming.

ISAM offers the programmer flexibility in the operations he can perform on a file. He has the ability to:

- Read or write logical records whose keys are in ascending collating sequence.
- Read or write individual records randomly, on the basis of the primary keys. If a large portion of a file is being processed, reading records in this manner is somewhat slower than reading according to a collating sequence. A search through indexes is required for each logical record.
- Add logical records with new keys to the existing file. The file management routines of ISAM find proper locations in the file for the new records and make all necessary adjustments to the indexes so that the new records may be retrieved easily. New logical records are physically stored in a separate *overflow area*; the logical sequence to other logical records in the file is maintained through the indexes. As new records are added, the performance of ISAM decreases slightly, until it becomes advisable to reorganize the file (see below).

ISAM has the following restrictions:

- Data records must be of fixed length.
- All *physical* blocks must contain key areas, all of the same length.
- For multivolume files, all volumes must be online for any function to be performed.
- ISAM uses three types of data areas in auxiliary storage: prime data area, overflow area, and indexes. The prime data area must be allocated in one continuous area which may be over more than one volume; it must begin on the first track (track 0) of a cylinder and it must end on the last track of a cylinder. For a multivolume file, the prime data area must continue from the last track of the last cylinder on one volume to the first track of cylinder 1 of the next volume so that the area is considered continuous by ISAM (cylinder 0 is reserved for labels). The overflow area and the indexes may be located on separate volumes.
- An ISAM file cannot be used as input for sort/merge programs. The data is organized on a logical basis: the logical records are sequenced logically, according to the keys of the records. Should a user attempt to re-sort this file, the indexes would no longer be an interface between the user's problem program and the data records, and individual records can no longer be located. It is possible, of course, to create another (sequential) file from the contents of an ISAM file, and then to re-sort this newly created file. If this is needed, VSE/VSAM alternate index support should be considered.
- Once a file has been created as an ISAM file, it should be processed and updated by means of ISAM only. SAM or DAM must never be used to process an ISAM file; doing so might cause serious problems with data integrity. ISAM manages the data completely, and this management function might be made impossible if a user were to destroy an index/data relationship as established by ISAM.
- ISAM does not actually delete logical records from an ISAM file. However, since a user may update any logical record, he can 'delete' a logical record by overwriting the data portion with, for example, binary zeros, decimal zeros, or blanks. However, he must make sure not to

overwrite the key field. He may also enter a special field in his logical record which contains the status of the data: current data or deleted data (see note, below). The main issue is that the user himself must distinguish deleted data from current data, and choose some satisfactory way of doing so.

A disadvantage of this restriction is that an ISAM file increases in size as many logical records are tagged as deleted, and many new records are added. Eventually, the file must be reorganized in order to obtain a 'clean' file. During this reorganization, the file is read sequentially (logically, according to the primary keys) and written (loaded) to a new file; logical records that have been tagged as deleted are then ignored by the user, and not written to the new file.

Note: Under OS/VS1 or OS/VS2, deleted records are flagged by placing the hexadecimal value "FF" in the first byte. It is recommended that VSE users who plan to use VSE ISAM data under OS/VS follow this procedure.

Record Types

When an ISAM file is originally organized, it is loaded onto the volume(s) from presorted input records. These records must be sorted by key and all records in the file must contain key areas:

Count	Key	Data

All keys must be the same length and this length must be specified in the DTFIS KEYLEN operand.

The logical records must be fixed length, and the length must be specified in the DTFIS RECSIZE operand. Logical records may be either blocked or unblocked, and this is specified in the DTFIS RECFORM operand. When blocked records are specified, the key of the highest (last) record in the block is the key for the block and, therefore, ISAM stores it in the key area of the record. The number of records in a block must be specified in the DTFIS NRECDS operand.

Storage Areas

Records of one logical file are transferred to, or from, one or more I/O areas in virtual storage. The areas must always be large enough to contain the key area and a block of records, or a single record if unblocked records are specified. Also, space must be allowed for the count area when a file is loaded, or when records are added to a file. For the functions of adding or retrieving records, the I/O area must also provide space for a sequence-link field used with overflow records (see "Addition of Records and Overflow Areas," below). When an overflow record is brought into the I/O area, you should not alter the sequence-link field. The I/O area requirements are illustrated in Figure 4-16 and described in detail in the discussions of the DTFIS IOAREAL, IOAREAR, IOAREAS, and IOAREA2 operands.

Records may be processed directly in the I/O area or in a work area for either random or sequential retrieval. If the records are processed in the I/O area, a register must be specified in the DTFIS IOREG operand. This register is used for indexing, and points to the beginning of each record.

If the records are processed in a work area, the DTFIS WORKL, WORKR, or WORKS operand must be specified (WORKL must always be specified when creating or adding records to a file). ISAM moves each individual input record from the I/O area to the work area where it is available to your program for processing. Similarly, on output, ISAM moves the completed record from the work area to the I/O area where it is available for transfer to DASD storage. Whenever a work area is used, no register is required.

Activating (Opening) a File for Processing

All ISAM files must be activated or opened before any processing can take place. The OPEN macro makes files available for processing by associating a logical file declared in your program with a specific physical file on a DASD unit. This association remains in effect throughout your program until you deactivate, or close, the file by issuing a CLOSE macro.

Self-relocating programs (see Appendix D) *must* use OPENR instead of OPEN to activate a file.

Processing an ISAM File

This section describes in general how the following functions are performed with ISAM:

- creating (loading) a file
- extending a file
- adding records
- sequential retrieval and update
- direct retrieval and update
- deleting records
- reorganizing a file

Figure 3-12 summarizes which macros apply to the individual processing functions. ISAM provides for processing with both the GET/PUT and the READ/WRITE macros; their purpose and effect vary, however, depending on the logic modules used and the conditions set by preceding macros.

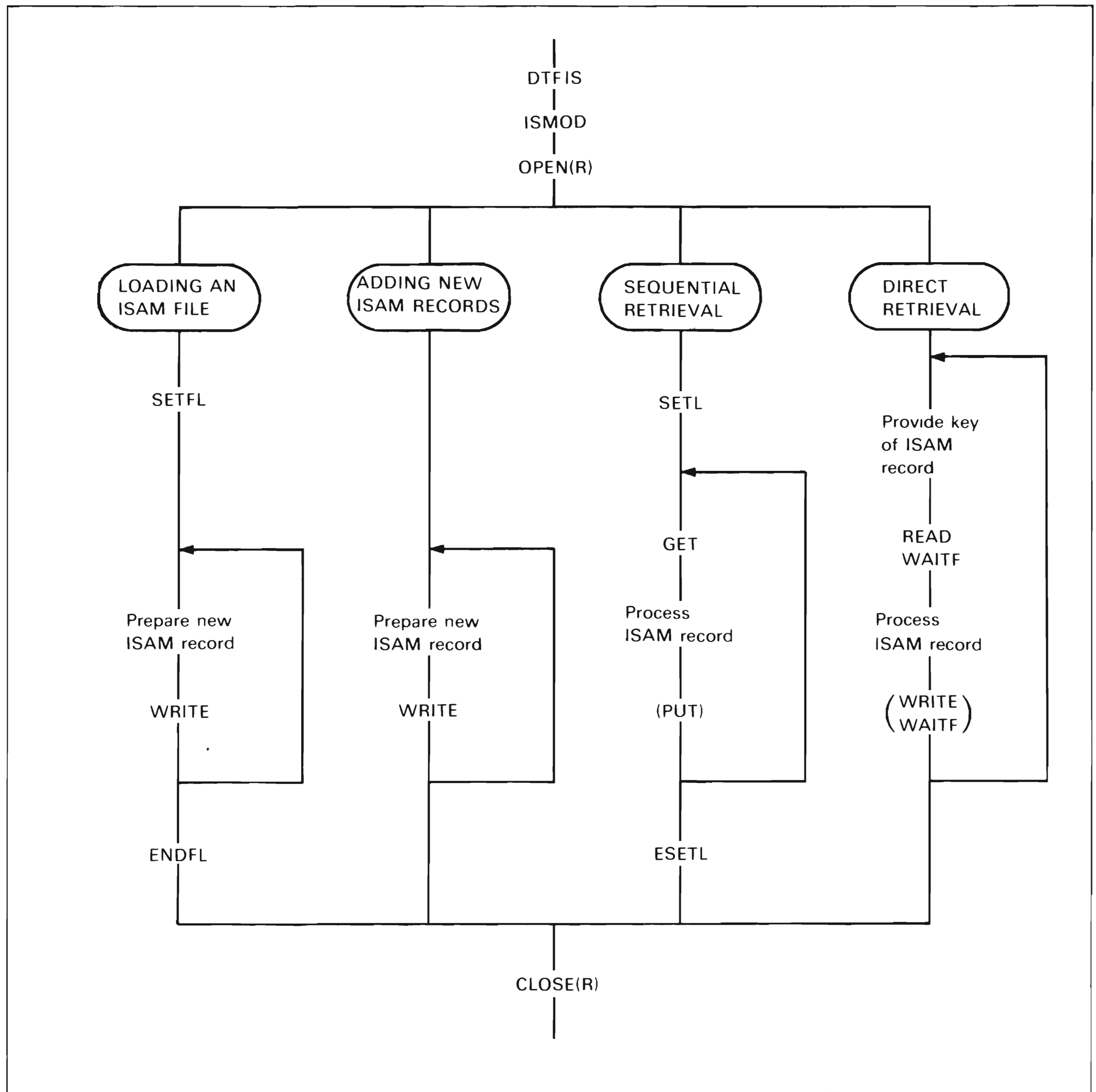


Figure 3-12. Controlling ISAM functions in assembler language.

Creating an ISAM File

An ISAM file is created by a load routine. You specify the characteristics of the output ISAM file in your problem program. The format-2 DASD label used for an ISAM file contains pre-recorded information about the record format, such as record length, block length, and key length. This implies that these must be fixed for a given file.

Before the actual loading begins, a SETFL macro is issued, which sets up the file and initializes the indexes. Each record is presented to the load routine separately, in ascending key sequence, by means of a WRITE macro. (The WAITF macro is not used when a file is loaded or extended.) Before transferring a record, ISAM performs a sequence check and a duplicate-record check. When all records have been loaded, an ENDFL macro terminates the loading process. It writes the last block of records followed by and end-of-file record; it completes the index and creates dummy index entries for the unused portion of the prime data extent.

The input may have a different format and be described separately as another file, or even more than one file. It may be provided by any access me-

thod, from any device, in any format suitable for that access method or device. The problem program constructs an ISAM record from the input; it presents the logical records to the ISAM load routine, one by one in ascending order by key. If requested by the problem program, the load routine performs record blocking. It also builds the indexes.

Creating the file cannot be combined with any other processing functions.

Extending an ISAM File

The function of extending an ISAM file by adding presorted records with keys higher than those already present is the same as creating an ISAM file from presorted records. Both are considered a load operation and should be defined as such in the DTFIS declarative macro by specifying IOROUT=LOAD. Both functions also use the same macros and the same program that initially sets up an ISAM file can also be used to extend the file.

Addition of Records and Overflow Areas

After a logical file is organized on a DASD, it may subsequently become necessary to add records. These records may contain keys that are above the highest key presently in the file and thus constitute an extension of the file. Or, these records may contain keys that fall between keys already in the file and therefore require insertion in the proper sequence in the file.

If all records to be added have keys that are higher than the highest key in the file, the upper limit of the prime area of the file can be adjusted (if necessary) with an EXTENT job control statement. The new records can then be added by presorting them and loading them into the file. No overflow area is required, and the file is merely extended further on the volume. However, new records can be batched with the normal additions and added to the end of the file.

If records must be inserted among those already in the file, an overflow area is required. ISAM uses the overflow area to permit the insertion of records without necessitating a complete reorganization of the established file. The fast random and sequential retrieval of records is maintained by inserting references to the overflow chains in the track indexes, and by using a chaining technique for the overflow records. For chaining, a sequence-linked field is prefixed to your data record in the overflow area that has the next-higher key. Thus a chain of sequential records can be followed when searching for a particular record. The sequence-link field of the highest record in the chain indicates the end of the chain. All records in the overflow area are unblocked, regardless of the specification in the DTFIS RECFORM operand for the data records in the file.

An example of the addition of records to an ISAM file using an overflow area is shown in Figure 3-13.

You may request two types of overflow areas;

- A cylinder overflow area for each cylinder. Specify the number of tracks to be reserved for each cylinder overflow area with the DTFIS CYLOFL operand when a file is loaded or when records are added to an existing file.
- An independent overflow area for the entire file, specified with an EXTENT job control statement. This area may be on the same volume as the file or on a different (on-line) volume of the same device type. An independent overflow area may be added to a file originally created without it when the DTFIS IOROUT operand specifies LOAD, ADD, or ADDRTR.

The independent overflow area may be used either in addition to cylinder overflow areas or without them. When used in addition to cylinder overflow areas, it is used whenever one of the cylinder overflow areas is filled.

There must always be one prime data track available for a DASD EOF record when additions are made to the last track in the prime data area containing records.

Sequential Retrieval and Update

A sequential processing routine retrieves all records in ascending sequence by key, starting with a specified record somewhere in the file and continuing to the point where the program decides to break the sequence. The program can then choose another starting point to process another sequential set of records. The SETL macro specifies the first record of the series of records to be retrieved sequentially. The actual retrieval of these records in logically sequential order is accomplished by GET macros. The ESETL macro ends the sequential mode initiated by the SETL macro. You may resume sequential processing at some other point in the file by issuing another SETL.

Updating is possible, since each record that is retrieved by the sequential processing routine can be processed and returned to its original location in the file. After a record has been processed, a PUT macro must follow to restore the record to its original location. If you have some records that do not need updating, GET macros can be issued without any PUT between.

When unblocked records are processed, the key areas are not read. Information regarding record,

Initial status of ISAM file

	Normal Entry		Overflow Entry					
Track Index Track 0	100	Track 1	100	Track 1	200	Track 2	200	Track 2
Prime Data Track 1	10		20		40		100	
Prime Data Track 2	150		175		190		200	
Overflow Track 3								

Status after inserting records with keys 25 and 101

Track Index	40	Track 1	100	Track 3 Record 1	190	Track 2	200	Track 3 Record 2
Prime Data	10		20		25		40	
Prime Data	101		150		175		190	
Overflow	100	Track 1	200	Track 2				

Status after inserting records with keys 26 and 199

Track Index	26	Track 1	100	Track 3 Record 3	190	Track 2	200	Track 3 Record 4
Prime Data	10		20		25		26	
Prime Data	101		150		175		190	
Overflow	100	Track 1	200	Track 2	40	Track 3 Record 1	199	Track 3 Record 2

Figure 3-13. Adding records to an ISAM file.

block, and key sizes is obtained from the format-2 DASD label. When records are blocked, the complete block is restored, but only if a PUT has been issued for any of its records.

In a program that processes an ISAM file sequentially, new records can be added. The sequential processing must then be terminated properly by an ESETL macro before you issue a WRITE macro to add a new record. After the addition of one or more records, sequential processing can be resumed by another SETL macro.

Direct Retrieval and Update

To retrieve a record directly from a file, place the record key in the key field specified in the DTFIS macro. A READ macro returns the specified record from the file by searching the indexes, reading the record, and presenting the data portion of the record to the problem program. Information regarding the record, block, and key sizes is obtained from the format-2 DASD label.

If the data is to be updated, you must also issue a WRITE macro to write the updated record back in its original positions. A WAIT macro must follow each READ or WRITE to suspend record processing until the data transfer is complete.

Combining Sequential and Direct Retrieval

It is possible to perform both sequential and direct processing in one problem program. Figure 3-14 illustrates how sequential and direct processing can be combined. You may, for example, process records sequentially and yet update some record in the file directly. Or you may, in a program that processes records sequentially, also add some records. Adding records and direct updating should not, however, be interspersed. Whether you are updating directly or adding records, the direct processing and the sequential processing must be kept separate. The sequential processing begins at a specified point and must be explicitly terminated before any direct processing can start.

Deleting Records in an ISAM File

ISAM provides no facility for actually deleting records. A programmer may, however, tag a record for deletion by any method desired as long as it does not alter the logical record sequence in the file. The data portion of a record may, for example, be overwritten with zeros, or a special field in a record may indicate that the record is deleted. If you plan to interchange data between VSE and OS/VS, you should use the hexadecimal value X'FF' in the first byte of the record for that purpose.

Records that are tagged for deletion can be eliminated when the file is reorganized.

Deactivating (Closing) an ISAM File After Processing

A CLOSE completion macro must be issued after the processing of an ISAM file has been completed in order to deactivate or close any file that was previously opened. Closing a file ends the association of a logical file with a specific physical file on a DASD unit.

Self-relocating programs (see Appendix D) *must* use CLOSER instead of CLOSE to deactivate a file.

See the section on "Label Processing" in "Appendix C" for information on label processing done by the CLOSE macro.

Reorganizing an ISAM File

As new records are added to an ISAM file, existing records are moved to the overflow areas. The access time for retrieving a record from an overflow area is greater than that for retrieving one from the prime data area. This is because prime data records are located by a device search over a track, whereas overflow records are found by scanning, record by record, through a chain of records. Therefore, an increasing number of overflow records reduces performance. For this reason, you should reorganize ISAM files as soon as you recognize the need for it.

The system maintains statistics to assist you in determining when reorganization is required. These statistics are maintained, in binary form, in the format-2 DASD label recorded with the file:

- Non-first-overflow reference count
  A 3-byte count of the number of times a reference (direct retrieval) is made to records that are second or higher links in an overflow chain.
- Prime record count
  A 4-byte count of the number of records in the prime data area.
- Number of independent overflow tracks
  A 2-byte count of the number of tracks that are still available in the independent overflow area (if used).
- Overflow record count
  A 2-byte count of the number of records in the overflow area.
- Cylinder overflow area count
  A 2-byte count of the number of cylinder overflow areas that are full.

These fields are maintained automatically by ISAM. In addition, there is another field that can contain statistics: tag deletion count. It is, however,

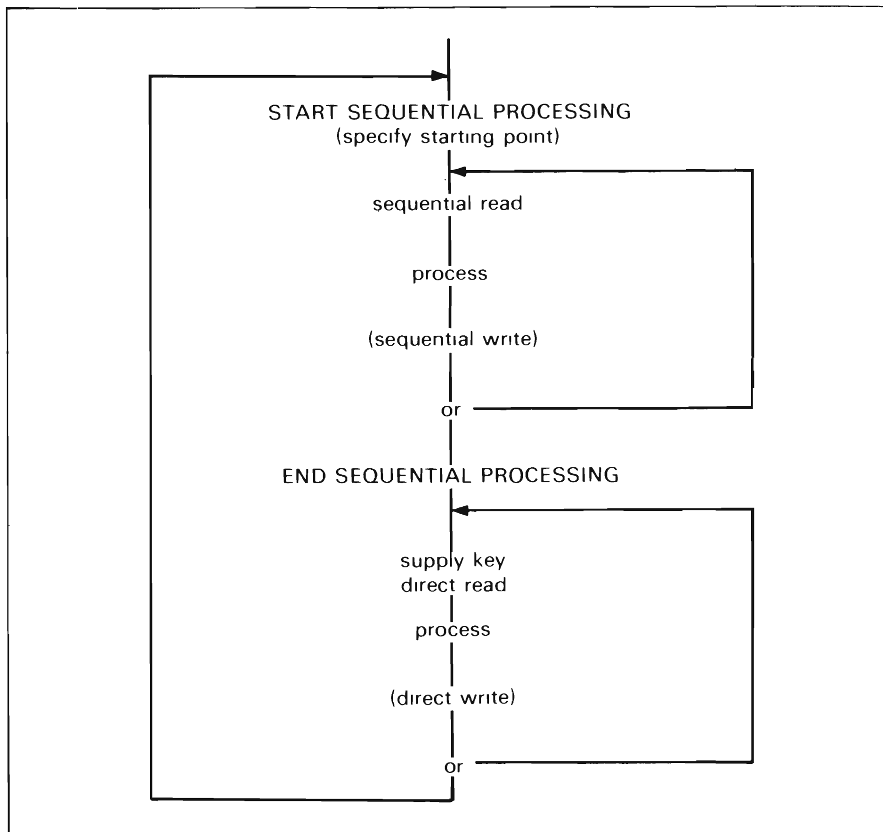


Figure 3-14. Combining sequential and direct processing.

not maintained by the ISAM routines, but the problem program can maintain it. The contents of this field are retrieved by the OPEN routines of ISAM and placed in virtual storage. After processing, the CLOSE routines of ISAM return this field to the format-2 DASD label. Before ISAM returns the field, you may use it for counting the number of records that have been tagged for deletion.

Reorganization is accomplished by creating a new version of the file, using the existing version as input. Two ISAM files are defined: the existing file as sequential input and the new version as load output. As far as the system is concerned, these two files are not related. It is the problem program that establishes a relationship by reading the existing version and loading the new version. Therefore, the records of the new version may be given quite another format provided that the problem programs that will process the new version are designed accordingly.

Depending on the capacity of the system, the reorganization is done in one step or in two steps. For a reorganization to be done in one step, the capacity of the system must be large enough to hold both files online. Otherwise the reorganization must be done in two steps: the first reads the existing version in key sequence and writes it to magnetic tape; the second then reads the magnetic tape and creates a new version. Records that are tagged for deletion may be eliminated in either step.

Figure 3-15 indicates how the reorganization process can be accomplished, in one or in two steps. When in two steps, an intermediate file is created in step 1 and processed as input in step 2. This file is processed in both steps by means of the Sequential Access Method. It may be written on magnetic tape or on DASD. Since this file need not be completely online, a single DASD or tape drive is sufficient.

Logic Modules for ISAM

You must assemble the logic modules available with ISAM from a source statement library supplied by IBM. This is a one-time process. Once assembled, the modules can be stored in the relocatable library.

The logic modules can be link-edited with any problem program that requires them. If preferable, however, they can also be assembled along with your program and included in the same output object module.

Four basic types of routines are available with ISAM:

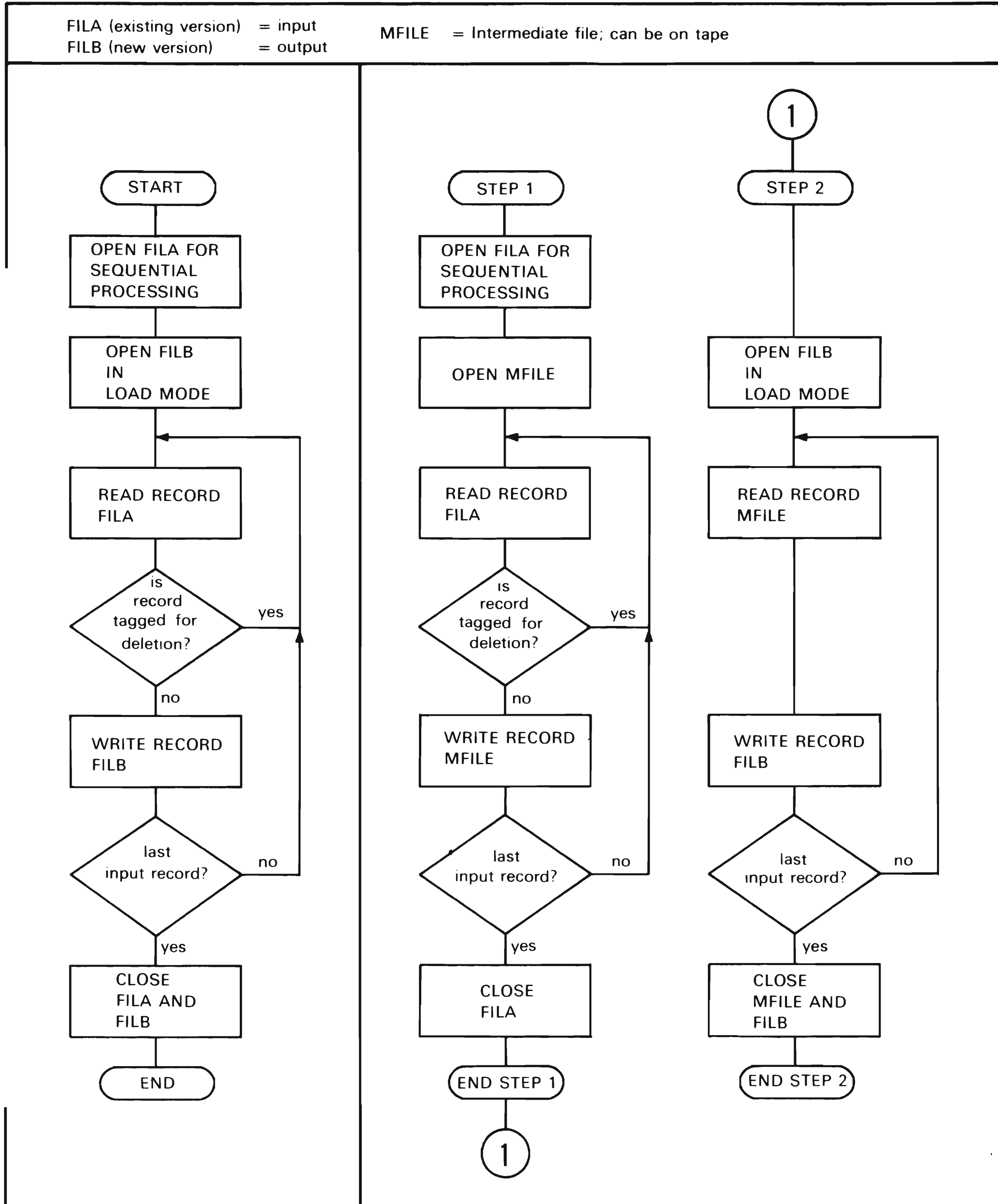


Figure 3-15. Reorganizing an ISAM file in one step or in two steps.

LOAD
: to create or extend an ISAM file,

ADD
: to insert additional records in an ISAM file,

SEQUENTIAL RETRIEVAL
: to retrieve records in logical sequence from an ISAM file,

DIRECT RETRIEVAL
: to retrieve individual records by key from any point in an ISAM file.

The load routine is always separate; no other functions can be performed at the same time. The add and retrieval functions can be used in combination. Futhermore, the retrieval routines have an updating capability, which allows you to write updated records back into their original location in the file.

The logic module for a specific file in a given problem program is assembled on a selective basis according to the requirements you specify in the DTFIS and the ISMOD declarative macros. The DTFIS macro generates a DTF table, which serves as a link between the problem program and the logic module.

If the DTFIS and the ISMOD macros specify that two I/O areas are used, overlap of the physical transfer of data with processing can take place in the load and the sequential retrieval routines.

PIOCS (Physical IOCS)

Records can be transferred to or from an input or output device by issuing physical IOCS macros. These macros relate directly to the physical IOCS routines and are distinct from the logical IOCS macro described in the SAM, DAM, and ISAM sections of this book. For more information on the distinction between the physical and logical IOCS, see *VSE System Data Management Concepts,* as listed in the Preface.

When using physical IOCS macros, you must provide for such functions as the blocking or deblocking of records, performing programmed wrong-length record checks, testing the CCB for certain errors, switching I/O areas when two areas are used, and setting up CCWs. You must also recognize and bypass checkpoint records if they are interspersed with data records on an input tape.

Physical IOCS routines control the transfer of data to or from the external device. These routines perform the following:

- Starting I/O operations
- I/O interrupt handling
- Channel scheduling
- Device error handling.

Thus physical IOCS macros provide you with the capability of obtaining data and performing nondata operations with I/O devices using exactly the CCWs you request. For example, if your program handles only physical records, you do not need the logical IOCS routines for blocking and deblocking logical records.

The macros available for direct communication with physical IOCS are CCB (command control block) or IORB (I/O request block); EXCP (execute channel program); WAIT and, if Rotational Position Sensing is used in your installation, SECTVAL.

Whenever physical IOCS macros are used, you must provide the CCWs for input and output operations in your program. Use the assembler instruction CCW statement to do this. A detailed technical description of the CCW can be found in *IBM System/370 Principles of Operation* or the *IBM 4300 Processors Principles of Operation,* as listed in the Preface. Considerations for CCW programming are given in Chapter 9.

Macros normally used with files processed by logical IOCS are necessary in addition to the macros provided by PIOCS when standard DASD or magnetic tape labels are processed, or when DASD file protect is present. The DTFPH, OPEN, CLOSE, LBRET, FEOV and SEOV macros can be used in this processing. The OPEN and the DTFPH macros are also necessary when a disk is used for a checkpoint file.

DASD Capacities

Capacity is a physical characteristic — for different types of DASD volumes the characteristics differ. Figures 4-1, 4-2, and 4-3 give capacity characteristics for count-key-data and fixed block architecture DASDs to help you to determine the number of volumes needed to contain your files.

Maximum number of bytes per physical record formatted without keys					Records per track	Maximum number of bytes per physical record formatted with keys				
2311	2314 2319	3330	3340	3350		2311	2314 2319	3330	3340	3350
3625	7294	13030	8368	19069	1	3605	7249	12974	8293	18987
1740	3520	6447	4100	9442	2	1720	3476	6391	4025	9360
1131	2298	4253	2678	6233	3	1111	2254	4197	2603	6151
830	1693	3156	1966	4628	4	811	1649	3100	1891	4546
651	1332	2498	1540	3665	5	632	1288	2442	1465	3583
532	1092	2059	1255	3024	6	512	1049	2003	1180	2942
447	921	1745	1052	2565	7	428	877	1689	977	2483
384	793	1510	899	2221	8	364	750	1454	824	2139
334	694	1327	781	1954	9	315	650	1271	706	1872
295	615	1181	686	1740	10	275	571	1125	611	1658
263	550	1061	608	1565	11	244	506	1005	533	1483
236	496	962	544	1419	12	217	452	906	469	1337
213	450	877	489	1296	13	194	407	821	414	1214
193	411	805	442	1190	14	174	368	749	367	1108
177	377	742	402	1098	15	158	333	686	327	1016
162	347	687	366	1018	16	143	304	631	291	936
149	321	639	335	947	17	130	277	583	260	865
138	298	596	307	884	18	119	254	540	232	802
127	276	557	282	828	19	108	233	501	207	746
118	258	523	259	777	20	99	215	467	184	695
109	241	491	239	731	21	90	198	435	164	649
102	226	463	220	690	22	82	183	407	145	608
95	211	437	204	652	23	76	168	381	129	570
88	199	413	188	617	24	69	156	357	113	535
82	187	391	174	585	25	63	144	335	99	503
77	176	371	161	555	26	58	133	315	86	473
72	166	352	149	528	27	53	123	296	74	446
67	157	335	137	502	28	48	114	279	62	420
63	148	318	127	478	29	44	105	262	52	396
59	139	303	117	456	30	40	96	247	42	374

Figure 4-1. Record capacities of selected DASDs.

Storage Device	Track Capacity in Bytes When R0 is Used as Specified by IBM Programming Systems	Bytes Required for Data Records			
		Data Records (except for last record)		Last Record	
		Without key	With key	Without key	With key
2311	3625	$61+\frac{537 \cdot D}{512}$	$81+\frac{537 \cdot (K+D)}{512}$	D	20+K+D
2314 2319	7294	$101+\frac{2137 \cdot D}{2048}$	$146+\frac{2137 \cdot (K+D)}{2048}$	D	45+K+D
3330 3333	13165	135+D	191+K+D	135+D	191+K+D
3340	8535	167+D	242+K+D	167+D	242+K+D
3350	19254	185+D	267+K+D	185+D	267+K+D
D = data length K = key length					

Figure 4-2. Track capacities of selected DASDs.

Count-Key-Data (CKD) Devices

Storage Device ▶	2311	2314* 2319*	3330-1* 3330-2* 3333*	3330-11*	3340**	3350*
Volumes per device	1	8	8	8	2	8
Cylinders per volume	200	200	404	808	696	555
Cylinders per device	200	1,600	3,232	6,464	1,392	4,440
Tracks per cylinder	10	20	19	19	12	30
Tracks per volume	2,000	4,000	7,676	15,352	8,352	16,650
Tracks per device	2,000	32,000	61,408	122,816	16,704	133,200
Sectors per track (RPS)	—	—	128	128	64	—
Bytes per track	3,625	7,294	13,165	13,165	8,535	19,254
Bytes per cylinder	36,250	145,880	247,570	247,570	100,416	572,070
Bytes per volume	7,250,000	29,176,000	100,018,280	200,036,560	69,889,536	317,498,850
Bytes per device	7,250,000	233,408,000	800,146,240	1,600,292,480	139,779,072	2,539,990,800

* The table shows the maximum data capacity of an installation with eight disk drives.

** The table shows the data capacity of a model A2 with two 3348 Model 70 or Model 70 F data modules.

Fixed Block Architecture (FBA) Devices

Storage Device ▶	3310	3370
Actuators per spindle	1	2
Blocks per actuator*	126,016	558,000
Blocks per spindle*	126,016	1,116,000
Bytes per actuator	64,520,192	285,696,000
Bytes per spindle	64,520,192	571,392,000

* An FBA block = 512 bytes

Figure 4-3. Disk storage capacity table.

Processing with SAM

Before any processing can be done on a sequentially organized file by SAM, the file must be defined by the DTFSD macro. The DTFSD operands are listed in Figure 4-4.

SAM DASD logic modules are pre-assembled, and loaded into the SVA at IPL time. When a SAM DASD file is OPENed, the proper logic module is automatically selected and connected to the DTF. Because of this, problem programs no longer need to specify an SDMODxx macro to obtain a SAM DASD logic module.

Programs that have an assembled SDMODxx will run as if SDMODxx were not there at all because OPEN routes control to the logic module in the SVA.

FBA (Fixed Block Architecture) DASD Processing

An FBA (Fixed Block Architecture) DASD is a direct access device on which data resides in blocks of a fixed size that are addressed by relative block number. These blocks are known as *FBA blocks* and their number and size depends upon the individual DASD device, as shown in Figure 4-3.

A sequentially-organized data set on an FBA DASD is an ordered set of units of data transfer called CI's (Control Intervals). In recording the data on an FBA device, SAM writes each CI over an integral number of FBA blocks. A control interval is composed of one or more *logical blocks,* which correspond to the physical blocks that were discussed earlier. That is, a logical block, whose length is specified by the DTF BLKSIZE operand, is made up of one or more logical records. The control interval size is determined by OPEN routines, based upon the DTF parameters CISIZE, BLKSIZE, and RECSIZE, and the DLBL statement parameter CISIZE.

For more information about the FBA CI concept, see *VSE System Data Management Concepts.*

Most programs that use SAM (that is, non-EXCP) imperative macros to access data can run unchanged with FBA DASD. Exceptions are programs containing elements that are sensitive to I/O synchronization, such as error exits and logging. These programs will have to be reevaluated and may require programming changes.

In the discussions that follow, bear in mind that FBA DASDs use control intervals as the unit of data transfer, rather than the physical block. Since these need not be the same size as logical blocks, issuing a macro that usually causes a block transfer need not cause an actual CI transfer.

For instance, issuing a WRITE macro to an FBA file transfers a logical record from the output area to the CI buffer. When the CI buffer is filled, it is transferred to the DASD asynchronously with the WRITE. For applications that require physical writes to be done when a WRITE or PUT is issued, the force write (PWRITE=YES) operand must be specified on the DTFSD macro.

SAM automatically reformats CIs into logical blocks and vice versa, as it automatically blocks and deblocks, and should be of no concern to the programmer.

Proper selection of CISIZE for FBA devices can affect overall throughput. For example, if the CI size is such that only one logical block and attendant control information fits into the CI, the number of physical I/O operations required to access a file is essentially the same as is required for CKD devices. However, if the CI size is large enough to contain two or more logical blocks plus the needed control information, throughput is improved because fewer physical I/O operations are required to access the same file.

Indiscriminate increasing of the CISIZE must be avoided in a heavily multi-programming system or system throughput may be adversely affected by excessive paging activity.

Opening a File

In order to make a file available for processing, your program must issue an OPEN macro.

DASD Input

In a multi-volume file only one extent is processed at a time, and thus only one pack need be mounted at a time. When processing on a volume is completed, message

4n55A WRONG PACK, MOUNT nnnnnn

will be issued so that the next volume may be mounted.

When a volume is opened, OPEN checks the standard VOL1 label and goes to the VTOC to check the file label(s). OPEN checks the specified extents in the extent statements against the extents in the labels to make sure the extents exist. If LABADDR is specified, OPEN makes the user standard header labels (UHL) available to you one at a time for checking.

After the labels are checked, the first extent of the file is ready to be processed. The extents are made available in the order of the sequence number on the extent statements. The same extent statements that were used to build the file can be used when the file is used as input, and if your specifications fall within these limits, IOCS makes the area that you specified available for processing.

Note: If EXTENT cards with specified limits are included in the job stream, or if an extent was created by replying with an extent to message

4450A NO MORE AVAILABLE EXTENTS

when the file was built, then an additional EXTENT card must be submitted on input to process that extent. If no EXTENT cards are submitted, however, this additional extent is processed normally.

When the last extent on the mounted volume is processed, the user standard trailer labels are made available for checking one at a time. The next volume is then opened.

For DASD devices that are file protected, when OPEN makes the first extent of the new volume

Applies to					
Input	Output	Work			
X	X	X	M	BLKSIZE=nnnn	Length of one I/O area, in bytes. Can be overridden at OPEN time via the DLBL statement.
X		X	M	EOFADDR=xxxxxxxx	Name of your end-of-file routine.
X	X	X	O	CISIZE=nnnn	Size of FBA Control Interval. If omitted for an FBA device, the default is 0, and the CISIZE will be set dynamically by OPEN.
		X	O	DELETFL=NO	CLOSE is not to delete format-1 and format-3 labels for work file.
X	X	X	O	DEVADDR=SYSnnn	Symbolic unit required only when not provided on an EXTENT statement.
X	X	X	O	ERROPT=xxxxxxxx	(IGNORE, SKIP, or name of error routine). Prevents job termination on error records. Do not use SKIP for output files.
X	X		O	FEOVD=YES	Forced end of volume for disk is desired.
X		X	O	HOLD=YES	Employ the track or block hold function.
X	X		O	IOAREA1=xxxxxxxx	Name of first I/O area. Optional for FBA files. If not specified, it will be GETVISed by OPEN. If the blocksize is increased, the IOAREA(s) is GETVISed to the larger size.
X	X		O	IOAREA2=xxxxxxxx	If two I/O areas are used, name of second area.
X	X		O	IOREG=(nn)	Register number. Use only if GET or PUT does not specify work area or if two I/O areas are used. Omit WORKA.
X	X		O	LABADDR=xxxxxxxx	Name of your routine to check/write user-standard labels.
	X	X	O	PWRITE=YES	For FBA files only, specify for a physical write of each logical block.
X	X	X	O	RECFORM=xxxxxx	(FIXUNB, FIXBLK, VARUNB, VARBLK, SPNUNB, SPNBLK, or UNDEF). For work files use FIXUNB or UNDEF. If omitted, FIXUNB is assumed.
X	X		O	RECSIZE=nnnnn	If RECFORM=FIXBLK, number of characters in record. If RECFORM=SPNUNB, SPNBLK, or UNDEF, register number. Not required for other records.
X	X		O	SEPASMB=YES	DTFSD is to be assembled separately.
X	X		O	TRUNCS=YES	RECFORM=FIXBLK or TRUNC macro used for this file.
X	X	X	O	TYPEFLE=xxxxxx	(INPUT, OUTPUT, or WORK). If omitted, INPUT is assumed.
X		X	O	UPDATE=YES	Input file or work file is to be updated.
	X		O	VARBLD=(nn)	Register number if RECFORM=VARBLK and records are built in the output area. Omit if WORKA=YES.
	X	X	O	VERIFY=YES	Check disk records after they are written.
X			O	WLRERR=xxxxxxxx	Name of your wrong-length-record routine.
X	X		O	WORKA=YES	GET or PUT specifies work area. Omit IOREG. Required for RECFORM=SPNUNB or SPNBLK.

M = Mandatory
O = Optional

Note: With Release 2 of VSE/Advanced Functions, some operands of the DTFSD macro are no longer required since support for them is always included. These operands are:

CONTROL=YES
DEVICE={2311|2314|3330|3340|3350|FBA}. The actual device type is determined by OPEN.
ERREXT=YES
MODNAME=name. Open will always load an IBM-supplied logic module and link it to the DTF.
NOTEPNT={POINTRW|YES}. NOTEPNT=YES is always assumed.
RDONLY=YES.

No compilation problems will occur when submitting the obsolete operands if properly specified; they will simply be ignored by OPEN.

Figure 4-4. DTFSD macro operands. For details of these operands, see *VSE/Advanced Functions Macro Reference.*

available, it makes the extent(s) from the previous volume unavailable.

DASD Output

When a multi-volume DASD file is created using SAM, only one extent is processed at a time. Therefore, only one pack need be mounted at a time. When processing on a volume is completed, message

```
4n55A WRONG PACK, MOUNT nnnnnn
```

will be issued so that the next volume may be mounted.

When a file is opened, OPEN checks the standard VOL1 label and the specified extents:

1. The extents must not overlap each other.
2. The first extent must be at least two tracks long (for CKD) if user standard labels are created.
3. Only extent types 1 and 8 are valid.

The data extents of a sequential CKD DASD file can be type 1, type 8 or both. Type 8 extents are called split cylinder extents and use only a portion of each cylinder in the extent. The portion of the cylinder used must be within the head limits of the cylinder and within the range of the defined extent limits. For example, two files can share three cylinders — one file occupying the first two tracks of each cylinder and the other file occupying the remaining tracks. In some applications, the use of split cylinder files reduces the access time.

For an FBA DASD, only type 1 extents are valid. Split cylinder extents are not valid on FBA devices as the addressing scheme does not use cylinders.

When OPENing a file, the FBA control interval size is stored in the format-1 label when the file is created, and retrieved from it when the file is re-OPENed as an input file.

OPEN checks all the labels in the VTOC to ensure that the file to be created does not destroy an existing file whose expiration date is still pending. It also checks to determine that the extents do not overlap existing extents. After having checked the VTOC, OPEN creates the standard label(s) for the file and writes the label(s) in the VTOC.

If you wish to create your own user standard labels (UHL or UTL) for the file, include the DTFxx LABADDR operand. OPEN reserves the first track of the first extent or a sufficient number of FBA blocks for the user header and trailer labels. Then, your label routine is given control at the address specified in LABADDR.

After the header labels are built, the first extent of the file is ready to be used. The extents are made available in the order of the sequence numbers on the actual extent statements. When the last extent on the mounted volume is filled, your LABADDR routine is given control and user standard trailer labels can be built. Then, the next specified volume in the extent statements is mounted and opened.

For a file-protected DASD, when OPEN makes the first extent of the new volume available, it makes the extent(s) from the previous volume unavailable. When the last extent on the final volume of the file is processed, OPEN issues an operator message. The operator has the option of canceling the job or typing in an extent on the console printer-keyboard or the operator display console and continuing the job.

Label Processing

If your program has specified a label processing routine (with LABADDR), the LBRET macro will cause SAM to write or check standard DASD labels. The LBRET macro has only one operand, the numerals 1, 2, or 3. Use one of these to specify which function you want SAM to perform:

Checking User Standard DASD Labels: IOCS passes the labels to you one at a time until the maximum allowable number is read (and updated), or until you signify you want no more. In the label routine, use LBRET 3 if you want IOCS to update (rewrite) the label just read and pass you the next label. Use LBRET 2 if you simply want IOCS to read and pass the next label. If an end-of-file record is read when LBRET 2 or LBRET 3 is used, label checking is automatically ended. If you want to eliminate the checking of one or more remaining labels, use LBRET 1.

Writing User Standard DASD Labels: Build the labels one at a time and use LBRET to return to IOCS, which writes the labels. Use LBRET 2 if you want control returned to you after IOCS writes the label. If, however, IOCS determines that the maximum number of labels has already been written, label processing is terminated. Use LBRET 1 if you wish to stop writing labels before the maximum number of labels is written.

See also the section on "Label Processing" in Appendix C.

Error Handling

Certain DTFSD macro operands are provided to assist you in processing I/O and record-length errors. Before discussing the use of these macro operands in detail, it is important that you understand the basic alternatives open to you regarding the handling of SAM file errors.

For example, the first decision you must make is whether or not you want to code your own error processing routine for the file and have LIOCS exit to it when an error condition occurs.

The alternative to doing your own error processing is to rely on LIOCS to satisfy the handling of error conditions in a more general and limited way. However, even if you choose this alternative, you still have some options open to you and these will be discussed more fully when the use of a specific macro operand is described.

The DTFSD macro operands which you may use to achieve the desired error processing are: WLRERR and ERROPT.

Control can be returned from your WLRERR or ERROPT error processing routine by means of the ERET imperative macro. Nonrecoverable I/O errors (such as 'no record found') occurring before data transfer takes place are indicated to your error processing routine.

To take full advantage of the error processing capabilities, you must include the ERROPT=name operand.

When an error condition occurs, register 1 will contain the address of a 2-part parameter list. The first four bytes of the list is the address of the DTF table and the second four bytes is the address of the physical record in error. Logic modules provide the error exit parameter list. You can make use of both addresses in your error routine, the first address to interrogate specific indicators in the CCB (the first 16 bytes of the DTF table — see Figures 9-3 and 9-4), and the second address to access the record for error processing.

The WLRERR=name operand (only in the DTF macro) is used only in conjunction with input files. With it you identify your routine for processing wrong-length records. If you omit this operand, one of the following actions will occur if a wrong-length record is detected:

- If the ERROPT operand is also omitted, the wrong-length record condition is ignored, or
- If the ERROPT operand is included for this file, the wrong-length record is treated as an error block and it is handled according to your specification for an error (see the discussion of the ERROPT operand which follows).

The ERROPT operand is used to indicate your choice of action if an error block is encountered. It has three valid parameters — IGNORE, SKIP, or the *name* of your error routine. However, for an output file, the only acceptable ERROPT parameters are IGNORE or name. On an UPDATE=YES file, the parameter SKIP causes write errors to be ignored. The functions of these parameters are:

IGNORE

For input files, the error condition is completely ignored and records are made available for processing by your main program. When reading spanned records, the entire spanned record or block of spanned records is returned to you rather than just the one physical record in which the error occurred.

When writing spanned records, the error is ignored and the physical record containing the error is treated as a valid record. If possible, any remaining spanned record segments are written.

SKIP

For input files, no records in the error block are made available for processing by your main program. The next block is read from the disk and processing continues with the first record of that block. When reading spanned records, the entire spanned record or block of spanned records is skipped rather than just the one physical record in which the error occurred.

On an UPDATE=YES file, the physical record in which the error occurred is ignored as if it were written correctly. If possible, any remaining spanned record segments are written.

name

IOCS branches to your error processing routine named by this parameter. In your routine, you process or make note of the error condition as you wish.

Programming Your Error Processing Routines

You may perform any kind of error processing you want in your error routine; however, you must abide by certain rules and restrictions.

- For a file assigned to a DASD device, the use of any LIOCS macro other than ERET in your error processing routine may cause termination of the task.

- Register 1 contains the address of the 2-part parameter list; otherwise, register 1 contains the address of the physical record in error. You must access the error block, or the records in the error block, by the address in the parameter list or in register 1. (The content of the IOREG register or work area — if either is specified — is unpredictable and, therefore, should not be used for error processing.) When spanned records are processed, the address is that of the whole blocked or unblocked spanned record.
- The data transfer bit (byte 2, bit 2) of the DTF table (CCB) should be tested to determine whether a nonrecoverable I/O error has occurred. If the bit is on, the physical record in error has not been read or written. If the bit is off, data was transferred and your routine must address the physical record in error to determine the action to be taken.
- At the conclusion of error processing, your routine must return control to LIOCS either:
  - by the ERET macro.

    For an input file:

    The program skips the block in error and reads the next block with an ERET SKIP.

    Or, it ignores the error with an ERET IGNORE.

    Or, it makes another attempt to read the block with an ERET RETRY.

    For an output file:

    The program ignores the error condition with an ERET IGNORE or ERET SKIP.

    Or, attempts to write the block with an ERET RETRY.
  - by branching to the address in register 14. For a read error, IOCS skips the error block and makes the first record of the next block available for processing in your main program.

Note: You cannot use the ERET RETRY option in your WLRERR error routine when processing record length errors. For this condition, ERET RETRY results in job termination.

Wrong-length Error Processing Considerations: If the block read is shorter than the length specified in the BLKSIZE operand, the first two bytes of the DTF table (CCB) contain the number of bytes left to be read (residual count). Therefore, the size of the actual block is equal to the specified block size minus the residual count. In this case there is no wrong-length error condition for variable and spanned records. If the block read is longer than the length specified in the BLKSIZE operand, the residual count is zero, and there is no way to compute the actual size of the block. In this latter case, the number of bytes transferred is equal to the length specified in the BLKSIZE operand, and the remainder of the block is lost (truncated).

Undefined records are not checked for incorrect record length. The record is truncated if its length exceeds the length specified in the BLKSIZE operand.

You issue an ERET macro in your error processing routine to skip or ignore the wrong-length error. SKIP causes the logic module to skip the block that contained the error and to read the next block. IGNORE causes control to be passed back to the logic module to ignore the error and to continue processing the block in error. (RETRY is invalid for use in WLRERR routines for sequential DASD files. Its use causes the job to be canceled with an invalid SVC message.)

Other Error Processing Considerations:

- When a parity error is encountered, attempts are made (by device ERPs) to reread or rewrite the erring block of records. If the attempts are unsuccessful, the job is terminated unless the ERROPT operand in the DTFSD macro is included.
- If an error occurs while rereading the physical block when updating spanned records, and neither WLRERR nor ERROPT operands are included, the entire logical record is skipped. Likewise, if an error occurs when rereading the physical block that contains the last segment of a blocked spanned record, the next entire logical record is skipped. If WLRERR and/or ERROPT has been included, the error recovery procedure is the same as for nonspanned records.
- A sequence error may occur if LIOCS is searching for a first segment of a logical spanned record and fails to find it. If you have specified either WLRERR=name or ERROPT=name, the error recovery procedure is the same as for wrong-length record errors. If you have specified neither WLRERR=name nor ERROPT=name, LIOCS ignores the sequence error and searches for the next first segment. Write errors are ignored.

Figure 4-5 summarizes the DTFSD error options for various combinations of error specifications and errors.

To Terminate the job,	specify nothing;
Skip the error record,	specify ERROPT=SKIP;
Ignore the error record,	specify ERROPT=IGNORE;
Process the error record,	specify ERROPT=name,
and/or (for wrong-length record error)	specify WLRERR=name.
After processing the record, to leave the error-processing routine and	
Skip the record (input only),	execute ERET SKIP;
Ignore the record,	execute ERET IGNORE;
Retry the record,	execute ERET RETRY.

Figure 4-5. DTFSD error options.

Deactivating a Sequential DASD File

To *force end-of-volume* on a sequential DASD file means that your program has finished processing records on one volume, but that more records in the same logical file are to be processed on the following volume. Issuing the FEOVD macro allows you to force end-of-volume before it actually occurs. If extents are not available on the new volume, or if the format-1 label is posted as the last volume of the file, control is passed to the EOF address specified in the DTF.

When FEOVD is issued to an input file, an end of extent is posted in the DTF. When the next GET is issued for this file, any remaining extents on the current volume are bypassed, and the first extent on the next volume is opened. Normal processing is then continued on the new volume.

When FEOVD is issued for an output file assigned to a CKD DASD, a short last block is written, if necessary, with a standard end-of-file record containing a data length of 0 (indicating end of volume). The end-of-volume indicator is not written on an FBA DASD, however, because an SEOF has already been written (if there was room for it) following the last data CI. An end-of-extent condition is posted in the DTF. When the next PUT is issued for the file, all remaining extents on the current volume are bypassed. The first extent on the next volume is then opened, and normal processing continues on the new volume. The end-of-volume marker written by VSE when an FEOVD macro is issued is compatible with the end-of-volume marker that OS/VS writes when an EOV macro is issued.

If the FEOVD macro is followed immediately by the CLOSE macro, the end-of-volume marker is rewritten as an end-of-file marker, and the file is closed as usual.

A CLOSE normally deactivates an output file by writing an EOF record and output trailer labels, if any, after writing any outstanding data, for example the last block. CLOSE sets a bit in the format-1 label to indicate the last volume of the file. A file may be closed at any time by issuing this macro.

Because, for FBA DASD, the unit of data transfer is the control interval instead of the physical block, SAM will (if necessary) automatically write the last CI when a CLOSE is issued. The program must always issue a CLOSE, for both FBA and CKD devices, to insure that all processing for the file has been completed.

If there is no room in the last CI to hold an SEOF, the data set will be considered delimited by end-of-last-extent.

After a CLOSE, no further commands can be issued for the file unless it is reopened. Sequential DASD files cannot be successfully reopened for output unless the DTFSD table is saved before the file is first opened, and restored between closing the file and reopening it again as an output file.

Processing with DAM

Before any processing can be done on DAM files, they must be defined by the declarative macro DTFDA. Figure 4-6 shows the operands permitted for the DTFDA macro.

DAM DASD logic modules are pre-assembled, and loaded into the SVA at IPL time. When a DAM DASD file is OPENed, the proper logic module is automatically selected and connected to the DTF. Because of this, problem programs no longer need to specify a DAMOD or DAMODV macro to obtain a DAM DASD logic module.

Programs that have an assembled DAMOD or DAMODV will run as if they were not there at all because OPEN routes control to the logic module in the SVA.

DTFDA should not be used to define SYSIPT if the program may be invoked by a catalogued procedure and if SYSIPT contains data. In this case, the program must process the data sequentially, and the DTFDI macro should be used.

After the DAM files are defined by the declarative macros, the imperative macros are used to operate on the files. The imperative macros are divided into three groups: those for initialization, processing, and completion.

Applies to				
Input	Output			
X	X	M	BLKSIZE=nnnn	Length of one I/O area, in bytes
X	X	O	DEVICE=nnnn	This keyword will be ignored and the actual DASD device will be determined at OPEN time.
X	X	M	ERRBYTE=xxxxxxxx	Name of 2-byte field for error/status codes supplied by IOCS.
X	X	M	IOAREA1=xxxxxxxx	Name of I/O area.
X	X	M	SEEKADR=xxxxxxxx	Name of track-reference field.
X	X	M	TYPEFLE=xxxxxx	(INPUT or OUTPUT).
	X	O	AFTER=YES	WRITE filename, AFTER or WRITE filename, RZERO macro is used for this file.
X	X	O	CONTROL=YES	CNTRL macro is used for this file.
X	X	O	DEVADDR=SYSnnn	Symbolic unit required only when no extent statement is provided.
X	X	O	DSKXTNT=n	Indicates the number (n) of extent for a relative ID.
X	X	O	ERREXT=YES	Nondata transfer errors are to be indicated in ERRBYTE.
X		O	FEOVD=YES	Support for sequential disk end of volume records is desired.
X		O	HOLD=YES	Employ the track hold function.
X	X	O	IDLOC=xxxxxxxx	Name of field in which IOCS stores the ID of a record.
X	X	O	KEYARG=xxxxxxxx	Name of key field if READ filename,KEY; or WRITE filename,KEY; or WRITE filename,AFTER is used for this file.
X	X	O	KEYLEN=nnn	Number of bytes in record key if keys are to be processed. If omitted, IOCS assumes zero (no key).
X	X	O	LABADDR=xxxxxxxx	Name of your routine to check/write user labels.
X	X	O	RDONLY=YES	Generates a read-only module. Requires a module save area for each task using the module.
X		O	READID=YES	READ filename, ID macro is used for this file.
X		O	READKEY=YES	READ filename, KEY macro is used for this file.
X	X	O	RECFORM=xxxxxx	(FIXUNB, SPNUNB, VARUNB, or UNDEF). If omitted, FIXUNB is assumed.
X	X	O	RECSIZE=(nn)	Register number if RECFORM=UNDEF.
X	X	O	RELTYPE=xxx	(DEC or HEX). Indicates decimal or hexadecimal relative addressing.
X	X	O	SEPASMB=YES	DTFDA is to be assembled separately.
X	X	O	SRCHM=YES	Search multiple tracks, if record reference is by key.
	X	O	TRLBL=YES	Process trailer labels, LABADDR must be specified.
	X	O	VERIFY=YES	Check disk records after they are written.
X	X	O	WRITEID=YES	WRITE filename, ID macro is used for this file.
X	X	O	WRITEKY=YES	WRITE filename, KEY macro is used for this file.
X	X	O	XTNTXIT=xxxxxxxx	Name of your routine to process extent information.

M = Mandatory
O = Optional

Note: With Release 2 of VSE/Advanced Functions, the following operands of the DTFDA are no longer required:

DEVICE={2311|2314|3330|3340|3350}. The actual device type is determined by OPEN.
MODNAME=name. Open will always load an IBM-supplied logic module and link it to the DTF.

No compilation problem will occur when submitting the obsolete operands if properly specified; they will simply be ignored by OPEN.

Figure 4-6. DTFDA macro operands. For details of these operands, see *VSE/Advanced Functions Macro Reference.*

Initialization

The initialization macro OPEN must be used to activate a DAM file for processing. The OPEN macro associates the logical file declared in your program with a specific physical file on a DASD. The association by OPEN of your program's logical file with a specific physical file remains in effect throughout your processing of the file until you issue a CLOSE macro.

Included here under the category of initialization macros is the LBRET macro, which is concerned only

with label and extent processing. LBRET is used to return to IOCS from a subroutine of your program that writes or checks labels and extents.

If OPEN attempts to activate a LIOCS file (DTF) whose device is unassigned, the job is terminated. If the device is assigned IGN, an OPEN does not activate the file but turns on DTF byte 16, bit 2, to indicate the file is not activated. If DTF byte 16, bit 2 is on after issuing an OPEN, input/output operations should not be performed for the file, as unpredictable results may occur.

Whenever an input/output DASD file is opened and you plan to process user standard header labels (UHL only), you must provide the information for checking or building the labels. If this information is obtained from another input file, that file must be opened, if necessary, ahead of the DASD file. To do this, specify the input file ahead of the DASD file in the same OPEN or issue a separate OPEN preceding the OPEN for the file.

If the XTNTXIT operand is specified, OPEN stores the address of a 14-byte extent information area in register 1. Then, OPEN gives control to your extent routine. You can save this information for use in specifying record addresses. The next volume is opened (on an input file, only after the requested user labels are written). When all the volumes are open, the file is ready for processing. If the DASD device is file protected, all extents specified in extent cards are available for use.

If an *output* file is created using DAM, all volumes used must be mounted at the same time, and all the volumes must be opened before the processing is begun.

For each volume, OPEN checks the standard VOL1 label and checks the extents specified in the extent cards for the following:

1. The extents must not overlap.
2. Only type-1 extents can be used.
3. If user standard header labels are created, the first extent must be at least two tracks long for a CKD file.

OPEN checks all the labels in the VTOC to ensure that the created file does not write over an existing unexpired file. OPEN then creates the standard label(s) for the file and writes the label(s) in the VTOC.

If you wish to create your own user (UHL) labels for the file, include the DTF LABADDR operand. OPEN reserves the first track of the first extent for these header labels and gives control to your label routine.

Direct access *input* processing requires that all volumes containing the file be on-line and ready at the same time. All volumes used are opened before any processing can be done.

For each volume, OPEN checks the standard VOL1 label and then checks the file label(s) in the VTOC. OPEN checks some of the information specified in the extent cards for that volume. If LABADDR is specified, OPEN makes the user standard header labels available one at a time for checking.

Self-relocating programs using LIOCS must use OPENR to activate all files. In addition to activating files for processing, OPENR relocates all address constants within the DTF tables (zero constants are relocated only when they constitute the module address).

The LBRET macro is issued in your subroutines when you have completed processing labels or extents and wish to return control to IOCS. LBRET applies to subroutines that write or check DASD user standard labels or handle extent information. The operand used depends on the function to be performed. See also "Appendix C: Label Processing".

Processing

Once DAM files have been readied for processing with the initialization macros, the READ, WRITE, WAITF, and CNTRL macros may be used.

Loading and Processing a Direct Access File

The only difference between loading a direct access file (creating) and processing a direct access file (updating or retrieving records) is the file's initial status. In both cases, the same conversion algorithm is used for locating data blocks, and the entire file must be online.

Note: Multivolume direct access files on a 3340 cannot extend over different types of data modules.

Before creating a file, however, you must make sure that the disk storage area is cleared of any data that may have been stored previously. IBM provides two system utility programs to clear disk storage areas:

Device Support Facility
: This system utility program operates on complete volumes. It writes a preformatted VTOC and clears the entire volume. Afterwards each track contains a home address and a record zero describing the entire track as free space. The preformatted VTOC contains empty file labels. Although the Device Support Facility program cannot clear a portion of a volume, you can do so by writing a complete file con-

sisting of erased tracks preceded by record zero with the desired contents.

Clear Disk
This system utility program operates on logical files. It is used to preformat a disk storage area with dummy blocks of fixed-length format. It can be used either on a new pack after it has been initialized or on a used pack to clear data areas for a new file. Preformatting by means of the Clear Disk program is necessary for files of fixed-length records.

There are different methods that can be used for randomizing. The choice depends on the record structure (with or without key) and the record format (fixed or variable length).

Processing Records With a Key Area: If records are written with a key, certain functions which you must otherwise control yourself can be performed by the device. In the following text, a distinction is made between fixed-length and variable-length data blocks.

Fixed-length Blocks With a Key Area: The file should be preformatted by means of the Clear Disk utility program. The file will then contain dummy records of fixed length. If you place the same contents into dummy records and deleted records, you can use the same procedure for both creating and updating the file.

The key of a block allows you to distinguish a current data block from a dummy block. The keys of all dummy blocks have the same contents; keys of current data blocks are unique, each key identifying a particular data record. A dummy key identifies an empty location. You have two options for writing a record:

1. Randomizing to a CYLINDER address.
   You should specify the search-multiple-tracks option in the DTFDA macro (SRCHM=YES). This allows you to search for the first dummy record on a cylinder. You specify the cylinder address obtained from the randomizing algorithm. The search will start at the beginning of that cylinder and continue until either a dummy record is found or the end of the cylinder is reached.

   When a dummy record is found, the system returns a record address, and returns control to the problem program. You can write the new record at the address that was supplied. If no dummy record is found, the system indicates 'no record found'. The overflow routine must then become active. The technique for locating a record in the overflow area is the same as that for the prime data area.

   When randomizing to a cylinder address, it is preferable to use one or more separate cylinders as an independent overflow area. Cylinder overflow areas are not very useful here, unless the last tracks of each cylinder are excluded by the randomizing algorithm.

2. Randomizing to a CYLINDER and TRACK address.
   You may or may not use the search-multiple-tracks option. If you specify the option, the procedure is the same as above, except that the search begins at a specified track instead of at the beginning of a cylinder. If you do not specify the search-multiple-tracks option, the search for a dummy record will not extend beyond the specified track. The search continues until either a dummy record is found, or the end of the track is reached. When the system returns a record address, this address refers to the block following the one that was identified as a dummy block. The system will also return control to the problem program.

   If the system indicates 'no record found' on that track, you may issue a search for a subsequent track or activate your overflow routine.

   The method of searching specific tracks is probably more time consuming, but it gives you more direct control. You can choose between cylinder overflow areas and independent overflow areas. It is difficult to predict, though, which type will be most efficient. If the prime data area and the independent overflow area reside on the same volume, a switch to and from the overflow cylinders requires a movement of the read/write mechanism, which can be avoided if cylinder overflow areas are used.

Variable-length Blocks With a Key Area: The file should not be preformatted with the Clear Disk utility program. The Device Support Facility program can be used to clear a complete volume. To clear a particular area on a volume, you must write erased tracks with the appropriate contents in each record zero.

On each track of a file that contains variable-length blocks, record zero contains a count field (capacity record) that states the amount of free space at the end of that track. Space that is 'free' because a record has been deleted is not taken into account. Unlike fixed-length blocks, deleted variable-length blocks cannot be re-used for other data records.

You should always establish a randomizing algorithm that delivers a cylinder and a track address. The system checks the contents of the capacity record to determine whether or not the track can accommodate the new block. If it can, the new block is written after the last block on that track. If there is not enough space left in the prime data area, you are notified. You can perform the inquiry in the overflow area in exactly the same way, track by track, until a track is found that can accommodate the new record.

It is useful to provide cylinder overflow areas as well as a separate independent overflow area. When a prime data track overflows, try first to store the record in the cylinder overflow area. If this is not possible, store the record in the independent overflow area.

A record stored in the cylinder overflow area can later be retrieved automatically if the search-multiple-tracks option is specified by the retrieving program. For records that are stored in the independent overflow area, you must make a search when you later want to retrieve them.

Since records cannot be stored in the space occupied by deleted records, the cylinder overflow areas themselves may also overflow. In order to maintain processing efficiency, reorganization of the entire file will then soon be necessary. It can be done by reading the file track after track, clearing each track separately and then restoring each current data block as if it were new. Since deleted records are not restored, free space will be concentrated again at the end of the tracks. After the prime data tracks have been reorganized, the overflow area may then be processed, and an attempt will be made to write overflow records to the prime data area. Overflow records that cannot be moved to the prime data area are moved back into the overflow tracks. Deleted records are omitted.

Retrieving Records With a Key Area: Records may be retrieved by a search on key. If the option for a search on multiple tracks is specified, a record can be found on a cylinder, as long as you specify the start of the search at, or before, the record address. The same conversion algorithm that is applied for writing a record can be used for retrieving it.

A summary of the randomizing techniques discussed above is presented in Figure 4-9 later in this chapter.

Processing Records Without a Key Area: If records are written without a key, the location of a data block can be determined only by the randomizing algorithm. The device has no means of identifying a data block other than by the record address you have specified.

For data without a key, the most practical method of randomizing is to establish a conversion algorithm that calculates a cylinder, track, and record address. This implies that fixed-length records must be used, and that the file is preformatted by means of the Clear Disk utility program before being loaded. Variable-length blocks without a key cannot be processed directly on the basis of a unique record address, since writing to such an address requires a predefined block at that address. However, the size of that block cannot be predicted before the size of the actual data block to be inserted is known.

All of this makes the conversion algorithm for data without a key more critical, since each synonymous record becomes an overflow record.

In the prime data area, each block has a pointer field. As long as no synonyms are present for a certain prime data record, this pointer will be empty. If synonyms are present, this pointer will point to the synonym that comes first in the chain of overflow records. All synonyms for a particular prime data record are linked by overflow chain pointers (see Figure 4-7). The chain of overflow pointers is used to trace a specific overflow record; it says nothing, however, about the physical sequence of the records on the track. The physical sequence is invisible and of no concern to the user.

The procedure for adding a new record may be quite complex. Generally speaking you must do the following:

1. Compute a DASD record address by means of the randomizing algorithm.
2. Check whether or not the block at the computed address contains current data. This requires an input operation.
3. a. If the block contains no current data, write the new record at the computed address. Clear the overflow pointer to make sure it indicates that no synonyms are present.

   b. If the block does contain current data and the overflow pointer is empty: establish the address of a free record location for the synonym. (How this can be done will be described below.) Put this address into the overflow pointer. Restore the prime data block to its original location on disk and write the new record as the first overflow record.

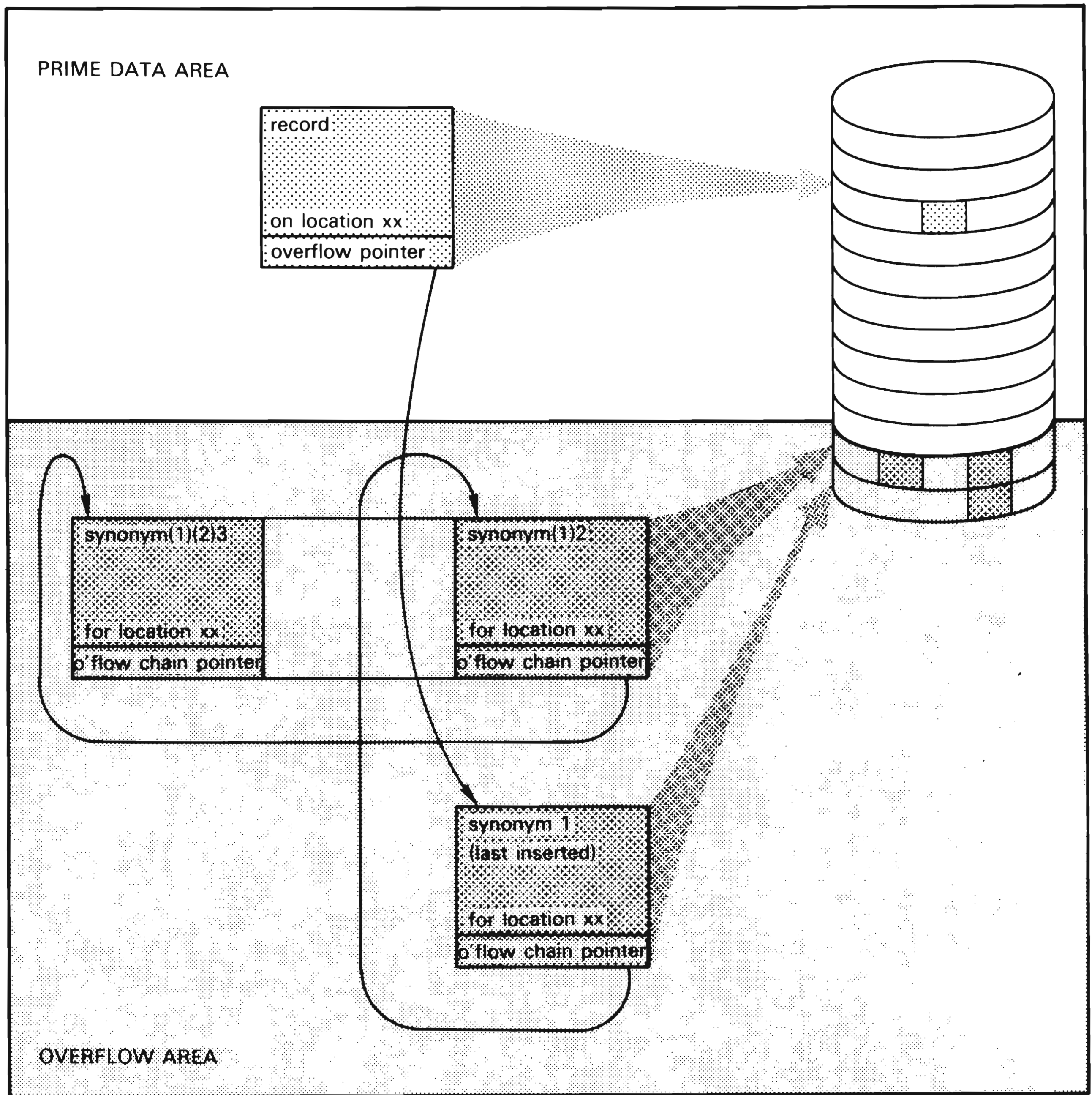


Figure 4-7. Prime data record and related overflow records.

c. If the block does contain current data and the overflow pointer indicates the presence of synonyms: save the address in the overflow pointer (this is the address of the first synonym in the overflow chain). Establish the address of a free record location for the new synonym. (How this can be done will be described below.) Write this address into the overflow pointer. Restore the prime data block to its original location on disk. Put the former contents of the overflow pointer into the overflow chain pointer of the new synonym. Finally, write the new synonym at its destined address in the overflow area.

   The effect of this procedure is that the newly inserted record becomes the first synonym in the overflow chain, and the former first one becomes the second. The rest of the sequence remains unchanged.

The randomizing algorithm calculates only prime data addresses. You must therefore establish an address in the overflow area by another method. You want to be able to find the address of a 'free' record location without having to scan the entire

overflow area; you would lose time in searching the overflow area block by block. A good method is to reserve the first record of the overflow area as an 'overflow area descriptor record'. This record is made to contain, at all times, the address of the first free block in the overflow area. This block has a pointer to the next free block. If a new record must be added to the overflow area, the 'overflow area descriptor record' gives the direct address of the block where this new record can be stored. The pointer to the next free record is then moved to the 'overflow area descriptor record'. At the same time, the new overflow record is added to a chain, as was explained before.

When a record is deleted from the overflow area, the address of that block is moved to the 'overflow area descriptor record', and becomes the address of the new first free record. The address that was in the 'overflow area descriptor record' is moved to the block that just became free, becoming the pointer to the next free block. The examples in Figure 4-8 illustrate this process.

Block 1 in the overflow area is the 'overflow area descriptor record'. The data portion of this block may contain any information you need, in addition to a pointer that points to the first 'free' block. In the top diagram it points, as an example, to block 3. Block 3, in turn, points to block 5 as the next 'free' block, etc. Thus, starting in block 1, you can easily locate all blocks that are free.

The same diagram illustrates the overflow chain: block 4 in the prime data area points to block 2 in the overflow area as being its first synonym. This synonym location, in turn, points to a next synonym, if any, and so on. Thus, overflow blocks 2, 4, and 6 form the overflow chain for prime data block 4.

If, for example, a new record must be placed on prime data location 4 (according to some conversion algorithm), this new record must be placed in the overflow area since prime data block 4 already contains current data. In this situation, the new record can be written into overflow block 3, which is the first 'free' block, and added to the overflow chain that already exists. The center diagram in Figure 4-8 shows the situation after the new record has been added. Note that the new record becomes the first overflow block in the overflow chain, and that block 1 in the overflow area now points to block 5 as the first 'free' overflow block.

When a block must be deleted from the overflow area, you must locate it properly following the overflow chain. The deleted record becomes the first 'free' overflow block. The bottom diagram in Figure 4-8 shows the situation after deleting block 6 from the overflow area.

A result of this method may be that a chain of records must be searched before the desired record is found. A requirement of this method is that the pointers must be adjusted when a record is deleted.

There is an alternative method that can be used for fixed-length records without a key. In this method, the randomizing algorithm calculates a cylinder and track address only. You must then check to see whether the track can accommodate the new record. This means that a record-by-record scan must be performed until a record is found that contains no current data. Most likely, more than one block will have to be read before the right one is retrieved and the overflow area must be used if the track is full. Since overflow records are now chained by track, the overflow chains may be much longer than when randomizing down to a record address. As a result, this procedure will be rather time consuming, and is therefore not very attractive. For variable-length blocks this method is not practical; it may impose serious retrieval problems.

A summary of the randomizing techniques discussed is presented in Figure 4-9.

Reading Blocks of Data

The READ macro transfers a record from DASD to an input area in virtual storage. The input area must be specified in the DTFDA IOAREA1 operand, and the WAITF macro must be used.

The READ macro returns control to the problem program after requesting PIOCS to execute a CCW chain. You can perform processing unrelated to that block of data and then issue a WAITF macro to check for the completion of the read operation.

The READ macro is written in either of two forms depending on the type of reference used to search for the record. Both forms may be used for records in any one DTFDA-specified file if the file has keys.

This macro always requires two parameters. The first parameter specifies the name of the file from which the record is to be retrieved. This name is the same as that specified in the DTFDA header entry for the file and can be specified either as a symbol or in register notation. The second parameter specifies the type of reference used for searching the records in the file.

If records are undefined (RECFORM=UNDEF), DAM supplies the data length of each record in the designated register in the DTF RECSIZE operand.

Initial status of DASD file

Prime Data Area

Block 1 contains data	Block 2 contains data	Block 3 contains data	Block 4 contains data	Block 5 free	Block 6 contains data	Bl co da
Pointer field	Pointer field	Pointer field	points to Block 2	Pointer field	Pointer field	Point

Overflow Area

Block 1 Area descriptor record	Block 2 contains data	Block 3 free	Block 4 contains data	Block 5 free	Block 6 contains data	Bl co da
points to Block 3	points to Block 4	points to Block 5	points to Block 6	points to Block 8	Pointer field	Pointe

Status after inserting a new synonym for prime data block 4

Prime Data Area

			Block 4 contains data			
			points to Block 3			

Overflow Area

Block 1 Area descriptor record	Block 2 contains data	Block 3 contains data	Block 4 contains data	Block 5 free	Block 6 contains data	
points to Block 5	points to Block 4	points to Block 2	points to Block 6	points to Block 8	Pointer field	

Status after deleting overflow block 6

Prime Data Area

			Block 4 contains data			
			points to Block 3			

Overflow Area

Block 1 Area descriptor record	Block 2 contains data	Block 3 contains data	Block 4 contains data	Block 5 free	Block 6 free	
points to Block 6	points to Block 4	points to Block 2	Pointer field	points to Block 8	points to Block 5	

Figure 4-8. Sample overflow organization. Blocks that remain unchanged in the center and bottom diagrams are left blank. The 'free-record pointers' in the overflow area must be written by the user after the file has been preformatted by the Clear Disk program and before the file is loaded for the first time.

LOADING AND PROCESSING DIRECT ACCESS FILES	
RECORDS WITH A KEY	
Fixed-length blocks	
Loading:	The file is preformatted by the Clear Disk program. Randomize to a track or a cylinder address, whichever method is used for the file. A record will become an overflow record if the search for a dummy record is unsuccessful.
Processing:	Randomize to the track or the cylinder address, whichever method is used for the file.
Variable-length blocks	
Loading:	Randomize to a track address. The file is not preformatted; record zero (capacity record) indicates how much space is left on the track. A record will become an overflow record if the space left is not large enough.
Processing:	Randomize to the track address.
RECORDS WITHOUT A KEY	
Fixed-length blocks	
Loading:	The file is preformatted by the Clear Disk program. Randomize to a record address. Each synonym will become an over-flow record to be inserted logically in an overflow chain.
Processing:	Randomize to the record address. Read the record and check whether it is the one desired. If it is not, search the overflow chain.

Figure 4-9. Summary of randomizing methods.

Record Reference by Key: If the record reference is by key (control information in the key area of the DASD record), the second parameter in the READ macro must be the word KEY, and the READKEY operand must be specified in the DTFDA:

```
READ filename,KEY
```

Whenever this method of reference is used, your program must supply the desired record key to IOCS before the READ macro is issued. For this, the key must be stored in the key field (specified in the DTFDA KEYARG operand). When the READ macro is executed, IOCS searches the previously specified track (stored in the 8-byte track-reference field) for the desired key. When a DASD record containing the specified key is found, the data area of the record is transferred to the data portion of the input area.

Only the specified track is searched unless you request that multiple tracks be searched on each READ (by including the SRCHM operand in the DTFDA). With this entry, the specified track and all following tracks are searched until the desired record is found or the end of the cylinder is reached. The search of multiple tracks continues through the cylinder even though part of the cylinder may be assigned to a different file.

Record Reference by ID: If the record reference is by ID (identifier in the count area of records), the second paramter in the READ macro must be the letters ID, and the READID operand must be included in the DTFDA:

```
READ filename,ID
```

Whenever this method of reference is used, your program must supply both the track information and the record number in the track-reference field. When the READ macro is executed, IOCS searches the specified track for the particular record. When a record containing the specified ID is found, both the key area (if present and specified in the DTFDA KEYLEN operand) and the data area of the record are transferred to key and data portions of the input area.

LIOCS can be requested to return the ID of records after reading. You must specify the name of the field that is to contain returned IDs in the IDLOC=*name* parameter of the DTFDA macro. The ID returned is the ID of the record following the one read, but if the search-multiple-tracks option is specified (referencing by KEY), the ID of the record read.

Writing Blocks of Data

Data blocks can be written as new records or as updates for existing records. When a new record is written over a dummy record, this is also treated as an update.

The system can assign a new record to a record location that was not used before, by means of the capacity record. For overwriting an existing record, whether updating an old record or making use of a dummy record for actual data, a reference must be made by either ID or KEY.

To perform the various write operations the WRITE macro is issued, but in different formats. In all cases, the WRITE macro returns control to the problem program after requesting services from PIOCS. You can perform processing unrelated to the block of data to be written. You must issue a WAITF macro to check for the completion of the write operation.

Adding New Records: This is done with the WRITE macro in the format:

```
WRITE filename,AFTER[,EOF]
```

The program must supply the track address. The system examines the capacity record in record zero to determine the location and the amount of space available for the record.

If the remaining space is large enough, the count area, the key area (if any), and the data area are

written to the location immediately following the last record on that track. IOCS updates the capacity record.

If the space remaining on the track is not large enough, the problem program is notified.

This format of the WRITE macro cannot return an ID.

EOF is optional and applies only to the WRITE filename, AFTER form of the macro. This form writes an end-of-file record (a record with a length of zero) on a specified track after the last record on a track.

Overwriting Existing Records: If reference is by ID, the macro format is:

```
WRITE filename,ID
```

The program must supply the track address and the record number of the record to be written. The system searches for this ID and starts writing the key (if any) and the data. If an ID was requested, the ID returned will be the ID of the next record in the file.

If reference is by key, the macro format is:

```
WRITE filename,KEY
```

The program must supply the key of the record to be located, and the address of the track on which the record resides. The system then searches that track or, if the search-multiple-tracks option is specified in the DTFDA declarative macro, searches through the cylinder starting with the track specified. When the key is found, the data is written without the key.

If the DTFDA macro specifies that an ID must be returned, this ID will be the ID of the record following the one written; if the search multiple-tracks option is specified, the ID of the record written will be returned.

Write Verification: If you specify in the DTFDA macro that write operations must be verified (VERIFY=YES), a read command is issued after a write operation with any of the options ID, KEY, or AFTER. This is done without actually transferring data. The system checks whether the data, as it was recorded, is valid.

Clearing a Track: You can cause the contents of a track to be erased by issuing a WRITE macro in the format:

```
WRITE filename,RZERO
```

The program must supply the cylinder address and the track address. The system searches for this track, resets the capacity record (in record zero) to indicate the maximum capacity for the track, and erases the remainder of the track.

This format of the WRITE macro can be used to initialize a limited number of tracks or cylinders for a file.

Seeks: The READ and WRITE macros do not have to be preceded by a CNTRL macro with the SEEK operand. They automatically seek to the correct cylinder by means of the supplied track address. It may, however, improve processing speed to issue a seek in order to position the access mechanism to the correct cylinder before the actual value for ID or KEY is available. Keep in mind that such a preliminary seek operation may be canceled if more than one problem program is operating on the same volume (not necessarily the same file) at the same time. A seek issued by one program may be made useless by another program that issues an I/O request on the same volume. The CNTRL macro with the SEEK operand returns control to the problem program as soon as the operation is initiated.

If records in the file are undefined (that is, RECFORM=UNDEF), you must determine the length of each record and load it into a register for IOCS use before you issue the WRITE macro for that record. The register for this purpose must be specified in the DTFDA RECSIZE operand.

If you are creating variable length or spanned unblocked records with WRITE filename,AFTER you must put the data length of the record to be written plus 4 into the 5th and 6th bytes of the control fields preceding the data. In the case that you are updating records previously read by a READ macro from the same physical file, you should not change the control fields. Otherwise, the wrong length record bit will be set in the error information returned to your program.

Record Reference by Key: If the DASD location for writing records is determined by the record key (control information in the key area of the DASD record), the word KEY must be entered as the second parameter of the WRITE macro. Also the WRITEKY operand must be included in the DTFDA.

Whenever this method of reference is used, your program must supply the key of the desired record to IOCS before the WRITE is issued. The key must be stored in the key field (specified by the DTFDA KEYARG operand). When the WRITE is executed, IOCS searches the previously specified track (stored in the track-reference field) for the desired key. When a DASD record containing the specified key is found, the data in the output area is transferred to the data area of the DASD record. This replaces the information previously recorded in the data area. The DASD count field of the original record controls

the writing of the new record. If a record is shorter than the original record, it is padded with zeros. A record longer than the original record is written only to the extent of the area indicated in the count field on the track, and any excess bytes are lost. IOCS turns on the wrong-length-record bit in the error-status field if any short or long records occur.

Only the specified track is searched unless you request that multiple tracks be searched on each WRITE macro. Searching multiple tracks is specified by including the SRCHM operand in the DTFDA. In this case, the specified track and all following tracks are searched until the desired record is found or the end of the cylinder is reached. The search of multiple tracks continues through the cylinder even though part of the cylinder may be assigned to a different file.

Record Reference by ID: If the DASD location for writing records is determined by the record ID (identifier in the count area of records), ID must be entered as the second parameter of the WRITE macro and the WRITEID operand must be included in the DTFDA.

Whenever this method of reference is used, your program must supply both the track information and the record number in the track-reference field. When the WRITE is executed, IOCS searches the specified track for the particular record. When the DASD record containing the specified ID is found, the information in the output area is transferred to the key area (if present and specified in DTFDA KEYLEN) and to the data area of the DASD record.

If FIXUNB or UNDEF is specified in the RECFORM operand, the key must precede your data in the IOAREA1 area, otherwise you must load the key into the key field (specified by the KEYARG operand) before you issue the WRITE macro. This replaces the key and data previously recorded.

IOCS uses the count field of the original record to control the writing of the new record. A record longer than the original record is written only to the extent of the area indicated in the count field on the track, and any excess bytes are lost. IOCS turns on the wrong-length-record bit in the error/status field if any long records occur. If an updated record is shorter than the original record, it is padded with binary zeros to the length of the original record. The wrong-length-record bit is not set on.

Record Reference by AFTER: If a record is written following the last record previously written on a track (regardless of its key or ID), the second parameter of the WRITE macro must be AFTER and the AFTER=YES operand must be included in the DTFDA.

Whenever this method of reference is used for writing records, your program must supply the track information in the track-reference field. When WRITE is executed, IOCS examines the capacity record (record 0) on the specified track to determine the location and amount of space available for the record. If the remaining space is large enough, the information in the output area is transferred to the track in the location immediately following the last record. The count area, the key area (if present and specified by DTFDA KEYLEN), and the data area are written. IOCS then updates the capacity record. If the space remaining on the track is not large enough for the record, or the track is not followed by enough empty tracks in the case of spanned records, IOCS does not write the record and, instead, sets an indication in your error/status byte specified by the DTFDA ERRBYTE operand.

Whenever a new file is built in an area of the disk pack containing outdated records, the capacity records must first be set up to reflect empty tracks by issuing the WRITE RZERO macro.

For the 2311 and 2314, the capacity record will take into account a track tolerance of about 5%, to ensure that minor hardware imprecisions on the disk tracks do not interfere with program execution. If a record is close to the maximum record size for a track, the capacity record could thus show a negative value.

Record Reference by RZERO: Executing a WRITE filename,RZERO resets the capacity record to reflect an empty track. Your program must supply, in SEEKADR, the cylinder and track number of the track to be reinitialized. Any record number is valid but will be ignored. IOCS writes a new R0 with the maximum capacity of the track in a two-byte field and erases the full track after R0. The maximum track capacities are:

for 2311	3,625
for 2314 or 2319	7,294
for 3330 or 3333	13,165
for 3340	8,535
for 3350	19,254

This form of the WRITE macro should be issued every time your program reuses a certain portion of a pack or data module. It may be used as a utility function to initialize a limited number of tracks or cylinders.

Completion of Read or Write Operations

You must issue a WAITF macro to check if a read or write operation has been completed. This macro tests for errors and exceptional conditions. Any exceptional condition discovered is passed to a special two-byte field, the name of which is specified in the DTFDA macro. This field must be defined in the problem program.

The WAITF macro makes sure that the transfer of a record is complete. It requires only one parameter: the name of the file containing the record. The parameter can be specified either as a symbol or in register notation.

This macro must be issued before your program attempts to process an input record which has been read or to build another output record for the file concerned. The program does not regain control until the data transfer is complete. Thus, the WAITF macro must be issued after any READ or WRITE macro for a file, and before the succeeding READ or WRITE macro for the same file. The WAITF macro makes error/status information, if any, available to your program in the field specified by the DTFDA ERRBYTE operand.

Non-Data Device Command

By issuing a CNTRL macro with a filename,SEEK specified, you can cause access movement to begin for the next read or write operation. While the arm is moving for a SEEK, you can process data and/or request I/O operations on other devices.

IOCS seeks the track that contains the next block for that file without your having to supply a track address. If the CNTRL macro is not used, IOCS performs the seek or restore operations when a READ, WRITE, GET, or PUT macro is issued.

Issuing a CNTRL macro to seek a track address might not result in an improvement of throughput if the volume containing your file is being shared with files that are accessed by another program or task active at the same time. A condition such as this is even more likely to arise if your file is stored on a physical volume that represents two or more logical volumes of another device (a 3344, for example, which represents four logical 3348 70M data modules per spindle and access mechanism).

Error Handling

When you specify ERRBYTE=name in the DTFDA macro and ERREXT=YES in DTFDA, DAM will return to you I/O error condition codes in the two-byte field whose name is specified with ERRBYTE.

The ERRBYTE codes are available for testing by your program after the attempted transfer of a record is complete. You must issue the WAITF macro before you interrogate the error status information. After testing the ERRBYTE status code, your program can return to IOCS by issuing another macro. One or more of the error status indication bits may be set to 1 by IOCS. An explanation of these bits is given in Figure 4-10.

The ERREXT operand enables unrecoverable I/O errors (occurring before a data transfer takes place) to be indicated to your program.

Byte	Bit	Error/Status Code Indication	Explanation
0	0	Not applicable	Not applicable
0	1	Wrong-length record	The wrong-length record indication is applicable to fixed-length, undefined length, variable-length, and spanned records. **Fixed-length Records:** This bit is set on under the following conditions: • A READ KEY or WRITE KEY is issued, and the keylength differs from the length as specified by KEYLEN=n. No data is transferred. • A READ KEY is issued, and the data length differs from the specified length (BLKSIZE minus KEYLEN, or BLKSIZE minus KEYLEN plus 8 if AFTER=YES was specified). • A READ ID is issued, and the length of the record (including key if KEYLEN was specified) differs from the specified length (BLKSIZE, or BLKSIZE minus 8 if AFTER=YES was specified). • A WRITE KEY is issued, and the data length of the record is greater than specified in the count field in the DASD record on disk. The original record positions are filled, and the remainder of the updated record is truncated and lost. • A WRITE ID is issued, and the record length is greater than specified in the count field in the DASD record on disk. The original record positions are filled, and the remainder of the updated record is truncated and lost. **Note:** If an updated record is shorter than the original record, it is padded with binary zeros to the length of the original record. The wrong-length record bit is not set on. **Undefined-length Records:** This bit is set on under the following conditions: • A READ KEY or WRITE KEY is issued, and the keylength differs from the length as specified by KEYLEN=n. No data is transferred. • A READ KEY is issued, and the data length is greater than the maximum data size (BLKSIZE minus KEYLEN, or BLKSIZE minus KEYLEN plus 8 if AFTER=YES was specified). IOCS supplies the actual data length of the record read in the RECSIZE register. • A READ ID is issued, and the length of the record (including key if KEYLEN was specified) is greater than the maximum record length (BLKSIZE, or BLKSIZE minus 8 if AFTER=YES was specified). IOCS supplies the actual data length of the record read in the RECSIZE register. • A WRITE (KEY, ID, or AFTER) is issued, and the data length (loaded into the RECSIZE register) of the record is greater than the maximum data size (BLKSIZE minus KEYLEN, or BLKSIZE minus KEYLEN plus 8 if AFTER=YES was specified). The length of the record written is equal to the maximum data size. • A WRITE KEY is issued and the data length (loaded into the RECSIZE register) is greater than specified in the count field of the DASD record on disk. The original record positions are filled, and the remainder of the updated record is truncated and lost. • A WRITE ID is issued, and the record length is greater than specified in the count field of the DASD record on disk. The original record is truncated and lost. **Note:** If an updated record is shorter than the original record, it is padded with binary zeros to the length of the original record. The wrong length record bit is not set on. **Variable-length Records:** This bit is set on under the following conditions: • When a READ is issued and the LL count[1] is greater than the maximum value specified by the BLKSIZE operand. • When a nonformatting WRITE is issued and the record is larger than the physical record on the device, the record is written with the low-order bytes truncated. The indicator also is set on if the record is shorter than the physical record, but the low-order bytes of the physical record are padded with binary zeros. • When a formatting WRITE is issued and the LL count[1] is greater than the maximum specified block size, the record is written with the low-order bytes truncated. **Spanned Records:** This bit is set on under the following conditions: • When a READ is issued and the logical record size is larger than the value specified by BLKSIZE minus 8. Only the number of bytes specified is read. • When a nonformatting WRITE is issued and the record length is not the same as that of the record being processed. If the length specified is longer than the record being processed, the low-order bytes are ignored. If the length specified is less than the record being processed, it is padded with binary zeros.

[1] The LL count is contained in the first two bytes of the block descriptor and counts the length of the physical block including all control information. For more details see *VSE System Data Management Concepts.*

Figure 4-10. ERRBYTE error status indication bits (Part 1 of 2).

Byte	Bit	Error/Status Code Indication	Explanation
0	1	Wrong-length record (continued)	• If a formatting WRITE is issued and the logical record size is larger than the size specified with BLKSIZE minus 8, the record is truncated to the size specified. • If the first physical record encountered is not an only or first segment. The no-record-found indicator is also set on. • If another first segment is encountered after the first segment is read out before a middle or last segment.
0	2	Non-data-transfer error	The block in error was neither read nor written. If ERREXT is specified and this bit is off, transfer took place and your program should check for other errors in the ERRBYTE field.
0	3	Not applicable	Not applicable
0	4	No room found	This indication is applicable only when the WRITE AFTER form of the macro is used for a file. The bit is set on if IOCS determines that there is not enough room left on the track to write the record. The record is not written. With spanned records the no-room-found condition is set if not at least one data byte will fit on the specified track in addition to the key (if any) and the 8 byte control field, or if any successive tracks required to transfer the record are not completely empty.
0	5	Not applicable	Not applicable
0	6	Not applicable	Not applicable
0	7	Reference outside extents	The relative address given is outside the extent area of the file. No I/O activity has been started and the remaining bits should be off. If IDLOC is specified, its value is set to 9s for a zoned decimal ID or to Fs for a hexadecimal ID.
1	0	Data check in count area	This is an unrecoverable error.
1	1	Track overrun	The number of bytes on the track exceeds the theoretical capacity.
1	2	End of cylinder	This indication bit is set on when SRCHM is specified for READ or WRITE KEY and the end-of-cylinder is reached before the record is found. If IDLOC is also specified, certain conditions also turn this bit on (see "IDLOC operand").
1	3	Data check when reading key or data	This is an unrecoverable error.
1	4	No record found	This indication is given when a search ID or key is issued and a record is not found. This applies to both READ commands and WRITE commands and may be caused by these conditions: a. The record searched for does not exist in the file. b. The record cannot be found because of a machine error (that is, incorrect seek). For spanned record processing, if the first physical record encountered is not the first or only segment, this indicator is set on.
1	5	End of file	This indication is applicable only when the record to be read has a data length of zero. The ID returned in IDLOC, if specified, is hexadecimal FFFF or, in the case of RELTYPE=DEC, zoned decimal 9's. The bit is set only after all the data records have been processed. For example, in a file having n data records (record n+1 is the end-of-file record), the end-of-file indicator is set on when you read the n+1 record. This bit is also posted when an end-of-volume marker is detected. It is your responsibility to determine if this bit means true EOF or end of volume on a SAM file. This bit is also posted upon successful execution of a WRITE filename,AFTER,EOF macro.
1	6	End of volume	This indication is given in conjunction with the end-of-cylinder indication. This bit is set on if the next record ID (n+1,0,1) that is returned on the end of the cylinder is higher than the volume address limit. The volume address limit is: for 2311 cylinder 199, head 9 for 2314 or 2319 cylinder 199, head 19 for 3330-1, 3330-2 or 3333 cylinder 403, head 18 for 3330-11 cylinder 807, head 18 for 3340 with 3348 model 35 cylinder 347, head 11 for 3340 with 3348 model 70 cylinder 695, head 11 for 3350 cylinder 554, head 29 These limits allow for the reserved alternate track area. If both the end of cylinder and EOV indicators are set on, the ID returned in IDLOC is FFFF or, in the case of RELTYPE=DEC, zoned decimal 9's.
1	7	Not applicable	Not applicable

Figure 4-10. ERRBYTE error status indication bits (Part 2 of 2).

Completion

The CLOSE completion macro must be used after the processing of a file is completed. Theses macros end the association of the logical file declared in your program with a specific physical file on a DASD.

The CLOSE macro deactivates any file that was previously opened. If trailer labels are specified, they are written on output, and checked on input. A file may be closed at any time by issuing this macro. No further commands can be issued for the file unless it is reopened.

CLOSER *must* be used if the file was activated by means of the OPENR macro.

Processing with ISAM

Before any processing can be done on an indexed sequential file, it must first be defined by the declarative macros DTFIS and ISMOD. Figure 4-11 shows the operands for the DTFIS macro and Figure 4-12 for ISMOD.

After the ISAM files are defined by the declarative macros, the imperative macros are divided into three groups: those for initialization, processing, and completion.

DTFIS Operands for I/O Area Specification

Certains of the DTFIS operands define the size and format of the I/O area. These operands are discussed below and their results are illustrated in Figure 4-13.

IOAREAL=name

This operand must be included when a file is created (loaded) or when records are added to a file. It specifies the name of the output area used for loading or adding records to the file. The specified name must be the same as the name used in the DS instruction that reserves the area of storage. The ISAM routines construct the contents of this area and transfer the records to DASD.

This output area must be large enough to contain the count, key, and data areas of records. Furthermore, the data-area portion must provide enough space for the sequence-link field of overflow records whenever records are added to a file (see Figure 4-14).

If IOAREAL is increased to permit the reading and writing of more than one physical record on DASD at a time, the IOSIZE operand must be included when records are added to the file. In this case, the IOAREAL must be at least as large as the number of bytes specified in the IOSIZE operand.

When simultaneously building two ISAM files using two DTFs, do not use a common IOAREAL. Also, do not use a common area for IOAREAL, IOAREAR, and IOAREAS in multiple DTFs.

IOAREAR=name

This operand must be included whenever records are processed in random order. It specifies the name of the input/output area for random retrieval (and updating). The specified name must be the same as that used in the DS instruction that reserves this area of storage.

The I/O area must be large enough to contain the data area for records. Furthermore, the data-area portion must provide enough space for the sequence-link field of overflow records (see Figure 4-15).

IOAREAS=name

This operand must be included whenever records are processed in sequential order by key. It specifies the name of the input/output area used for sequential retrieval (and updating). The specified name must be the same as that used in the DS instruction that reserves this area of storage.

This I/O area must be large enough to contain the key and data areas of unblocked records and the data area for blocked records. Furthermore, the data-area portion must provide enough space for the sequence-link field overflow records (see Figure 4-15).

IOAREA2=name

This operand permits overlapping of I/O with indexed sequential processing for either the load (creation) or sequential retrieval functions. Specify the name of an I/O area to be used when loading or sequentially retrieving records. The I/O area must be at least the length of the area specified by either the IOAREAL operand for the load function or the IOAREAS operand for the sequential retrieval function. If the operand is omitted, one I/O area is assumed. If TYPFLE=RANSEQ, this operand must not be specified.

IOREG=(r)

This operand must be included whenever records are retrieved and processed directly in the I/O area. It specifies the register that ISAM uses to indicate which individual record is available for processing. ISAM puts the address of the current record in the designated register (any of 2 through 12) each time a READ, WRITE, GET, or PUT is executed.

Applies to						
Ran. Rtvl.	Seq. Rtvl.	Load	Add			
X	X	X	X	M	DSKXTNT=n	Maximum number of extents specified for this file.
X	X	X	X	M	IOROUT=xxxxxx	(LOAD, ADD, RETRVE, or ADDRTR).
X	X	X	X	M	KEYLEN=nnn	Number of bytes in record key (maximum is 255).
X	X	X	X	M	NRECDS=nnn	Number of records in a block. Specify for blocked records only; if unblocked, 1 is assumed.
X	X	X	X	M	RECFORM=xxxxxx	(FIXUNB or FIXBLK).
X	X	X	X	M	RECSIZE=nnnn	Number of characters in logical record.
X	X	X	X	O	CYLOFL=nn	Number of tracks for each cylinder overflow area. Maximum = 8 for 2311, 18 for 2314, 17 for 3330 and 3333, 10 for 3340.
X	X	X	X	O	DEVICE=nnnn	(2311, 2314, 3330, 3340). If omitted, 2311 is assumed.
X	X	X	X	O	ERREXT=YES	Non data-transfer error returns and ERET desired.
X	X	X	X	O	HINDEX=nnnn	(2311, 2314, 3330, 3340). Unit containing highest level index. If omitted, 2311 is assumed.
X	X		X	O	HOLD=YES	Track hold function is desired.
X			X	O	INDAREA=xxxxxxxx	Symbolic name of cylinder index area.
X			X	O	INDSKIP=YES	Index skip feature is to be used.
X			X	O	INDSIZE=nnnnn	Number of bytes required for the cylinder index area.
		X	X	O	IOAREAL=xxxxxxxx	Name of I/O area.
X				O	IOAREAR=xxxxxxxx	Name of I/O area.
	X			O	IOAREAS=xxxxxxxx	Name of I/O area.
	X	X		O	IOAREA2=xxxxxxxx	Name of second I/O area.
X	X			O	IOREG=(nn)	Register number. Omit if WORKA or WORKS is specified.
			X	O	IOSIZE=nnnn	Bytes allotted to IOAREAL.
X	X			O	KEYARG=xxxxxxxx	Name of key field in storage, for random retrieval or sequential retrieval starting by key.
X	X	X	X	O	KEYLOC=nnnn	Number of high-order position of key field within record, if RECFORM=FIXBLK.
X	X	X	X	O	MODNAME=xxxxxxxx	Name of ISMOD logic module for this DTF. If omitted, IOCS generates standard name.
X	X	X	X	O	MSTIND=YES	Master index used.
X	X	X	X	O	RDONLY=YES	Generates a read-only module. Requires a module save area for each task using the module.
X	X	X	X	O	SEPASMB=YES	DTFIS is to be assembled separately.
X	X			O	TYPEFLE=xxxxxx	(RANDOM, SEQNTL, or RANSEQ).
X	X	X	X	O	VERIFY=YES	Check disk records after they are written.
		X	X	O	WORKL=xxxxxxxx	Name of work area for loading or adding to the file.
X				O	WORKR=xxxxxxxx	Name of work area for random retrieval. Omit IOREG.
	X			O	WORKS=YES	GET or PUT specifies work area.

M = Mandatory
O = Optional

Figure 4-11. DTFIS macro. For details of these operands, as well as for those of ISMOD, see *VSE/Advanced Functions Macro Reference*.

Operand	Remarks
ERREXT=YES	Required if non-data-transfer error conditions or ERET are desired.
CORDATA=YES	Required to add records using the DTF IOSIZE operand.
CORINDX=YES	Required to add or retrieve records with the cylinder index entries in virtual storage.
HOLD=YES	Specifies the track hold option.
IOAREA2=YES	Required if two I/O areas are to be used.
IOROUT= {LOAD| ADD| RETRVE| ADDRTR}	Specifies function to be performed.
RDONLY=YES	Required if a read-only module is to be generated.
RECFORM= {FIXUNB| FIXBLK| BOTH}	Describes file. Required if IOROUT specifies ADD or ADDRTR. If IOROUT specifies LOAD or RETRVE, BOTH is assumed.
RPS=SVA	To assemble RPS logic modules.
SEPASMB=YES	If the module is assembled separately.
TYPEFLE= {RANDOM| SEQNTL| RANSEQ}	Required if IOROUT specifies RETRVE or ADDRTR.

Figure 4-12. Operands of the ISMOD macro

IOROUT= {LOAD|ADD|RETRVE|ADDRTR}

This entry must be included to specify the type of function to be performed. The parameters have the following meanings:

LOAD
: To build a logical file on a DASD or to extend a file beyond the highest record presently in a file.

ADD
: To insert new records into a file.

RETRVE
: To retrieve records from a file for either random or sequential processing and/or updating.

ADDRTR
: To both insert new records into a file (ADD) and retrieve records for processing and/or updating (RTR).

IOSIZE=n

This operand specifies the (decimal) number of bytes in the virtual-storage area assigned for the add function using IOAREAL. The number n can be computed using the following formula:

n=m(keylength+blocksize+40)+24

where m is the maximum number of physical records that can be read into virtual storage at one time; 40 is the sum of 8 for the count field and 32 for an ISAM CCW; 24 is another ISAM CCW. The number n must also be at least equal to

(keylength+blocksize+74)

This formula accounts for a needed sequence link field for unblocked records or short blocks (see Figure 4-15 and Figure 4-16).

If the operand is omitted, or if the minimum requirement is not met, no increase in throughput is realized.

The number n should not exceed the track capacity because throughput cannot be increased by specifying a number larger than the capacity of a track.

Initialization

The OPEN macro must be used to activate an ISAM file for processing. These macros associate the logical file declared in your program with a specific physical file on a DASD. The association by OPEN of your program's logical file with a specific physical file remains in effect throughout your processing of the file until you issue a CLOSE macro.

Self-relocating programs using LIOCS must use OPENR to activate all files. In addition to activating files for processing, OPENR relocates all address constants (except zero constants) within the DTF tables.

If OPEN attempts to activate a LIOCS file (DTF) whose device is unassigned, the job is terminated. If the device is assigned IGN, the OPEN does not activate the file but turns on DTF byte 16, bit 2, to indicate that the file is not activated. If DTF byte 16, bit 2, is on after issuing an OPEN, input/output operations should not be performed for the file, or unpredictable results may occur.

Whenever a DASD file is opened, you must provide the information for checking or building the labels. (See "Appendix C".)

When a file is created or extended, those volumes of the file to be written on are opened as output files. If the file consists of more than one volume, all the volumes must be on line and ready when the file is first opened.

For each volume, OPEN checks the standard VOL1 label and performs extensive checks on the extents specified in the EXTENT job control statements for that volume. The extents must meet the following conditions:

1. All prime data extents must be contiguous.

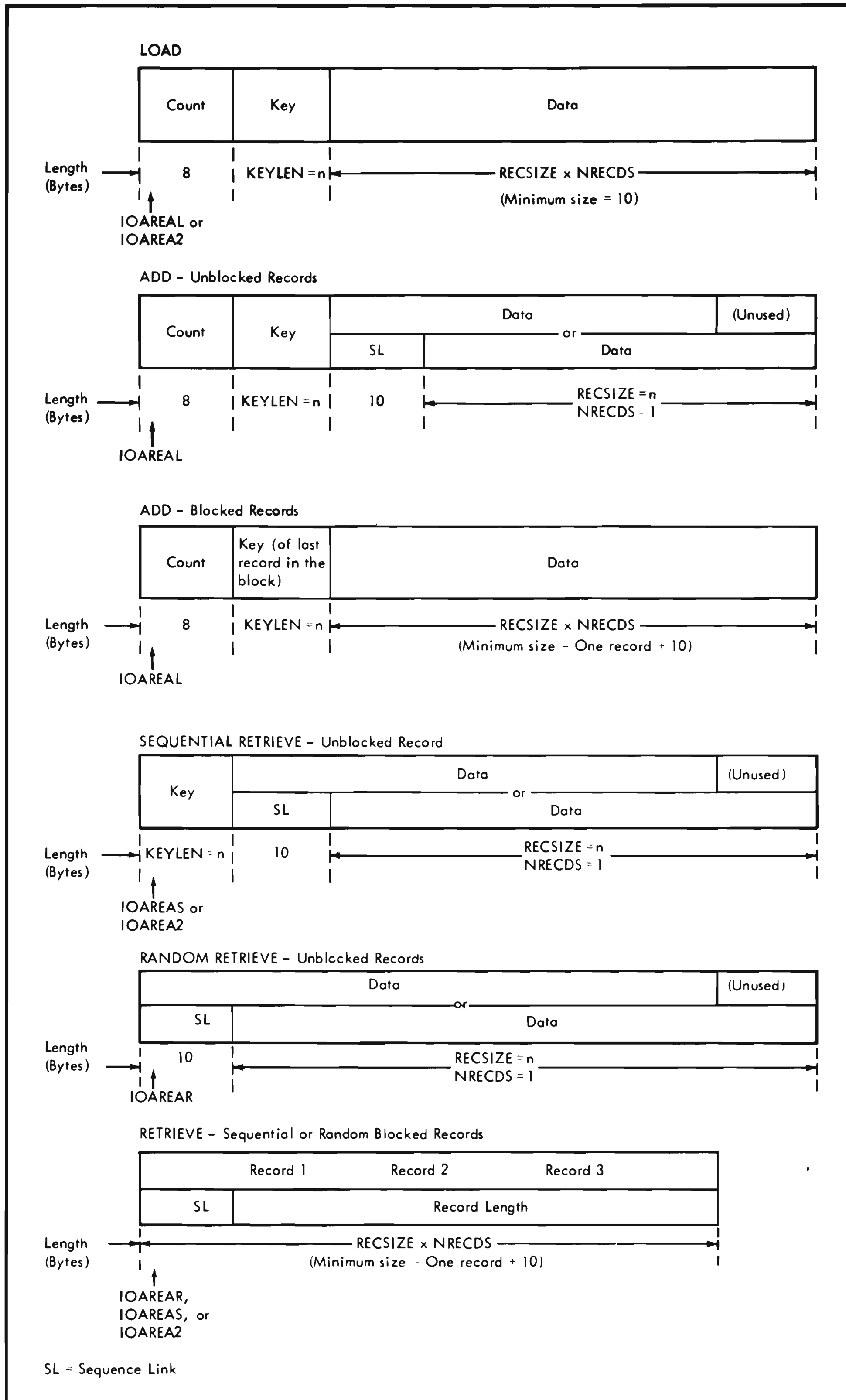


Figure 4-13. I/O areas resulting from different DTFIS operands.

Function	Output Area Requirements (in Bytes)			
	Count	Key	Sequence Link	Data
Load Unblocked Records	8	Key Length	—	Record Length
Load Blocked Records	8	Key Length	—	Record Length x Blocking Factor
Add Unblocked Records	8	Key Length	10	Record Length
Add Blocked Records	8	Key Length	—	Record Length x Blocking Factor
			OR*	
	8	Key Length	10	Record Length
* Whichever Is Larger				

Figure 4-14. Output area requirements for loading or adding records to a file by ISAM.

Function	I/O Area Requirements (in Bytes)			
	Count	Key	Sequence Link	Data
Retrieve Unblocked Records	—	Key Length for sequential unblocked records	10	Record Length
Retrieve Blocked Records	—	—	—	Record Length (including keys) x Blocking Factor
			OR*	
	—	—	10	Record Length
* Whichever is Larger				

Figure 4-15. I/O area requirements for random or sequential retrieval by ISAM.

2. The master and cylinder index extents must be contiguous and on the same unit.
3. No extents must overlap.
4. Only type 1, 2, or 4 extents are valid.
5. The extent sequence numbers must be in the following order:

   0 for master index, when present. 1 for cylinder index. 2, 3, 4,... for the prime data and independent overflow tracks.
6. For a single volume, only one prime data extent should be specified.

The EXTENT job control statements for the independent overflow tracks can be placed either before or after all the EXTENT job control statements for the prime data extents.

OPEN checks all the labels in the VTOC to ensure that the file to be created does not write over an existing file. Any expired labels are deleted from the VTOC. After having checked the VTOC, OPEN creates the standard labels for the file and writes the labels in the VTOC. If the DASD device is file protected, all extents specified in the EXTENT job control statements are available for writing. All volumes containing an ISAM file must be on-line and ready when the file is first opened.

For each volume, OPEN checks the extents specified in the EXTENT job control statements for that volume (for example, checks that the data extents are contiguous). OPEN also checks the standard VOL1 label and then goes to the VTOC to check the file label(s) before opening the next volume. After all the volumes are opened, the file is ready for processing. If the DASD device is file protected, all extents specified in EXTENT job control statements are file protected for the user.

Processing

Once ISAM files have been readied for processing with the OPEN macro, the processing macros described in this section may be used.

In this section, first the frequently used ERET macro is described, and then the groups of macros used for:

- Loading or extending a file
- Adding records to a file
- Retrieving records randomly
- Retrieving records sequentially.

Error Handling

At the completion of each imperative macro, your error routine should check filenameC. See the DTFIS ERREXT operand for details and for the format of filenameC. The ERET (error return) macro enables a program specifying the ERREXT operand in the DTF

to return to IOCS and specify an action to be taken for each error condition.

The ERREXT=YES operand is required for ISAM to supply your program with detailed information about unrecoverable I/O errors occurring before a data transfer takes place, and for your program to be able to use the ERET imperative macro to return to IOCS specifying an action to be taken for an error condition.

Some error information is available for testing by your program after each imperative macro is executed, even if ERREXT=YES is not specified, by referencing field filenameC. Filename is the same name as that specified in the DTF header entry for the file. One or more of the bits in the filenameC byte may be set to 1 by IOCS. The meaning of the bits varies depending on which parameter was specified in the IOROUT operand; Figure 4-16 shows the meaning if IOROUT=ADD, RETRVE, or ADDRTR was specified; Figure 4-17 shows the meaning if IOROUT=LOAD was specified.

If ERREXT=YES is not specified, IOCS returns the address of the DTF table in register 1, as well as any data-transfer error information in filenameC, after each imperative macro is executed; non-data-transfer error information is not given. After testing filenameC, return to IOCS by issuing any imperative macro except ERET; no special action is taken by IOCS to correct or check an error.

If ERREXT=YES is specified, IOCS returns the address of an ERREXT parameter list in register 1 after each imperative macro is executed, and information about both data-transfer and non-data-transfer errors in filenameC. The format of the ERREXT parameter list is shown in Figure 4-18. After testing filenameC and finding an error, return to IOCS by using the ERET imperative macro; IOCS takes the action indicated by the ERET operand. If HOLD=YES (and ERREXT=YES), ERET must be used to return to IOCS to free any held track.

In your program, you should check byte 16, bit 7 of the DTF for a blocksize compatibility error when adding to, or extending a file. If the blocksize of your program is not equal to the blocksize of the previously built file, this bit will be set to 1.

Note: The ERREXT routine does not handle nonrecoverable errors that are posted in filenameC. Examples of nonrecoverable errors are: no record found (may also be caused by hardware errors), prime data area full, master index full, etc. The supervisor may recover from a no record found condition if byte 3, bit 5 of the DTF is set. However, a recovery would then be initiated also for a nonrecoverable no record found condition.

Your error routine should determine whether or not data was transferred. This can be done by checking the data transfer bit (byte 2, bit 2) in the DTF. If the data transfer bit is on, the data was not read or written. If it is off, data transfer did take place.

If any IOCS macro other than ERET is issued in the error routine, the contents of registers 14 and 13 (with RDONLY) should be saved before use and restored after use.

Note: If the error occurred on an index record, you should not IGNORE this record unless it is first checked for accuracy. If the record was read inaccurately, you should RETRY to read the record.

Loading or Extending a File

The function of originally loading a file of presorted records onto a DASD, and of extending the file by adding new presorted records beyond the previous high record, are the same. Both are considered a load operation (specified by the DTFIS operand IOROUT=LOAD), and use the same macros. However, the codes field in a DLBL job control statement must specify ISC for load creation and ISE for load extension.

The areas of the volumes used for the file are specified by EXTENT job control statements. The areas are:

> The **prime data** area where the data records are written.
>
> A **cylinder index** area where you want ISAM to build the cylinder index.
>
> A **master index** area if a master index is to be built (specified by the DTFIS MSTIND operand).

During a load operation, ISAM builds the track, cylinder, and master indexes.

A combination of three different macros is required in your program to load original or extension records onto a DASD. These macros are SETFL, WRITE, and ENDFL.

SETFL sets the ISAM processing mode for loading or extending a file. WRITE performs the actual loading of new records into the file. ENDFL turns the load mode off. These three macros are described in detail below.

The SETFL (set file load mode) macro causes ISAM to set up the file so that the load or extension function can be performed. This macro must be issued whenever the file is loaded or extended.

When loading a file, SETFL preformats the last track of each track index. When extending a file, SETFL preformats only the last track of the last track index plus each new track index for the extension of the file. This allows prime data on a shared track to

Bit	Cause	Explanation
0	DASD error	Any uncorrectable DASD error has occurred (except wrong length record).
1	Wrong length record	A wrong length record has been detected during an I/O operation.
2	End of file	The EOF condition has been encountered during execution of the sequential retrieval function.
3	No record found	The record to be retrieved has not been found in the file. This applies to Random (RANSEQ) and to SETL in SEQNTL (RANSEQ) when KEY is specified, or after GKEY.
4	Illegal ID specified	The ID specified to the SETL in SEQNTL (RANSEQ) is outside the prime file limits.
5	Duplicate record	The record to be added to the file has a duplicate record key of another record in the file.
6	Overflow area full	An overflow area in a cylinder is full, and no independent overflow area has been specified; or an independent overflow area is full, and the addition cannot be made. You should assign an independent overflow area or extend the limit.
7	Overflow	The record being processed in one of the retrieval functions (RANDOM/SEQNTL) is an overflow record.

Figure 4-16. FilenameC - status or condition code byte if IOROUT=ADD, RETRVE, or ADDRTR.

Bit	Cause	Explanation
0	DASD error	An uncorrectable DASD error has occurred (except wrong length record).
1	Wrong length record	A wrong length record has been detected during an I/O operation.
2	Prime data area full	The next to the last track of the prime data area has been filled during the load or extension of the file. You should issue the ENDFL macro, then do a load extend on the file with new extents given.
3	Cylinder Index area full	The Cylinder Index area is not large enough to contain all entries needed to index each cylinder specified for the prime data area. This condition can occur during the execution of the SETFL. You must extend the upper limit of the cylinder index by using a new extent card.
4	Master Index full	The Master Index area is not large enough to contain all the entries needed to index each track of the Cylinder Index. This condition can occur during SETFL. You must extend the upper limit, if you are creating the file, by using an extent card. Or, you must reorganize the file and assign a larger area.
5	Duplicate record	The record being loaded is a duplicate of the previous record.
6	Sequence check	The record being loaded is not in the sequential order required for loading.
7	Prime data area overflow	There is not enough space in the prime data area to write an EOF record. This condition can occur during the execution of the ENDFL macro.

Figure 4-17. FilenameC - status or condition code byte if IOROUT=LOAD.

be referenced even though no track indexes exist on the shared track.

The name of the file loaded is the only parameter required for this macro and is the same as that specified in the DTFIS header entry for the file. It can be specified as a symbol or in register notation. Register notation is necessary if your program is to be self-relocating.

When a WRITE macro with the parameter NEWKEY is issued in your program between a SETFL macro and an ENDFL macro, ISAM loads a record onto the DASD.

The WRITE macro for loading and extending requires two parameters. The first parameter is the name of the file specified in the DTFIS header entry. The filename can be specified as a symbol or in register notation. The second parameter must be NEWKEY.

Before issuing the WRITE macro, your program must store the key and data portions of the record in a work area (specified by DTFIS WORKL). ISAM builds the output record in the I/O area (see Figure 4-13) by moving the data record to the data area, moving the key to the key area, and building the count area. When the I/O area is filled, ISAM transfers the record to DASD storage and then constructs the count area for the next record. The WAITF macro should not be used when loading or extending an ISAM file.

Before records are transferred, ISAM performs both a sequence check and a duplicate-record check. This ensures that the records are in order by key.

After each WRITE is issued, ISAM makes the ID of the record or block available to your program. The ID is located in an 8-byte field labeled filenameH, which must not exceed seven characters. For example, if the filename in the DTFIS header entry is

Bytes	Bits	Contents
0-3	-	DTF address.
4-7	-	Virtual storage address of the record in error.
8-15	-	DASD address of the error (mbbcchhr) where m is the extent sequence number and r is a record number which can be inaccurate if a read error occurred during a read of the highest level index.
16	.	Record identification:
	1	Data record
	2	Track index record
	3	Cylinder index record Master index record
	.	Type of operation:
	4	Not used
	5	Not used
	6	Read
	7	Write
17	-	Command code of failing CCW.

Figure 4-18. ERREXT parameter list.

PAYRD, the ID field is addressed by PAYRDH. The ID of any selected record can be punched or printed for later use by referencing this field. Using filenameH is required if you plan to retrieve records in sequential order starting with the ID of a particular record.

As records are loaded or extended on DASD, ISAM uses the I/O areas to write:

- The new track address each time a track is filled.
- Two track index records (one prime data, one overflow) each time a track is filled.
- A cylinder index record each time a cylinder is filled.
- A master index record (if DTFIS MSTIND is specified) each time a cylinder index is filled.

The ENDFL (end file load mode) macro ends the mode initiated by the SETFL macro. The name of the file to be loaded is the only parameter required, and is the same as the name specified in the DTFIS header entry for the file. The filename can be specified either as a symbol or in register notation. Register notation is necessary if your program is to be self-relocating. The ENDFL macro must be issued only after a SETFL and before a CLOSE.

The ENDFL macro performs an operation similiar to CLOSE for a blocked file. It writes the last block of data records, if necessary, and then writes an end-of-file record after the last data record. Also, it writes any index entries that are needed followed by dummy index entries for the unused portion of the prime data extent.

Adding Records to a File

If after having created the file, you want to add records with keys that are within the key range already present, you must use a different routine to place these records in the proper sequence and to write them. Each record is inserted in the proper place sequentially by key. To provide this function, specify ADD or ADDRTR in the DTFIS IOROUT operand.

The routine searches the indexes to determine the appropriate location for the new record. If the proper place for the new record is on a prime data track, the block on that location is read and the new record is inserted. Succeeding records on that track are shifted. The last record on the track is forced off the track and moved to the overflow area. Shifting on the last track of a file does not necessarily produce an overflow record, since there may be free space available. Figure 3-13 illustrates this process.

If it turns out that the proper place for an additional record is in the overflow chain, the record will be physically written into the next free location in the overflow area. Logically, however, it will be properly inserted in the overflow chain according to the value of its key.

The beginning of the overflow chain associated with a given prime data track is found in the overflow entry (in the track index) which also indicates the highest key of that chain. The chain is followed until a record is found with a key higher than that of the record to be inserted. The information in the link fields is updated to maintain the logically sequential order by key.

When you want to add a whole string of new records it is advisable to presort them in descending sequence. This will save time in executing the additions of these records.

The file may contain either blocked or unblocked records, as specified by the DTFIS RECFORM operand. When the file contains blocked records, you must provide ISAM with the location of the key field by means of the DTFIS KEYLOC operand. The records to be inserted are written one record at a time. The records must contain a key field in the same location as the records already in the file. Whenever the addition of records follows sequential retrieval (ADDRTR), the macro ESETL must be issued before a record is added. Two macros - WRITE and WAITF - are used in a program to actually add records to a file.

Each WRITE macro must be followed by a WAITF macro to ensure that the data transfer is complete before a new record is prepared. ISAM determines the appropriate location to insert the record; it checks for duplicate record keys and writes the record. If the new record is inserted on a prime data track, ISAM shifts all succeeding records on that track and sets up the last one as an overflow record. If the new record is inserted in an overflow chain, ISAM updates the appropriate linkages to maintain logically sequential order by key. The indexes are updated as well.

Whenever the addition of records follows sequential retrieval, the macro ESETL must be issued before a record can be added.

New records are added to the file by means of the WRITE macro with the NEWKEY parameter specified. Before the WRITE macro is issued for unblocked records, the program must store the record (key and data) to be added into a work area specified in the DTFIS WORKL operand. For blocked records, the program must store only the data since the key is assumed to be a part of the data. Before any records are transferred, ISAM checks for duplicate record keys. If none are found, ISAM inserts the record into the file.

To insert a record into a file, ISAM performs an index search at the highest level. This search determines whether the key of the record to be inserted is lower or higher than the key of the last record in the file. If it is lower, the record can be inserted, and searching the master index (if available), the cylinder index, and the track index determines the appropriate location to insert the record.

To add an entry to an unblocked file, an equal/high search is performed in the prime data area of the track. When such a condition occurs, the record is read from the track and placed in the I/O area specified in the DTFIS IOAREAL operand. The two records are then compared to check for duplicate records. If a duplication is found, this information is posted in the DTF table at filenameC. If none is found, the appropriate record (in your work area) is written directly to the track. The record (just displaced from the track) in the I/O area is moved by ISAM to your work area, and the next record on the track is read into the I/O area. Then, the record in the work area is written on the track. Succeeding records are shifted until the last record on the track is set up as an overflow record.

If the add I/O area (IOAREAL) is increased to permit the reading or writing of more than one record at a time, an equal/high search is performed in the prime data area of the track. When such a condition occurs, as many records as fit are read from the track and placed in the I/O area (specified in the DTFIS operand IOAREAL). The added record is compared with existing records in the I/O area. If a duplicate key is found, the condition is posted for you in the DTF table at filenameC. If no duplicate is found, the records are shifted in virtual storage, leaving the record with the highest key remaining in the work area. The other records are rewritten directly onto the track. Any remaining record(s) on the track are then read into the I/O area. This process continues until the last record on the track is set up as an overflow record. It is then written into the appropriate overflow area, and the track index entries are updated. This area becomes the cylinder overflow area, if CYLOFL is specified and the area is not filled.

Note that if a file was created with CYLOFL=n, all DTFs that add records to the file must specify the same number (n) of cylinder overflow tracks.

If the cylinder overflow area is filled, or if only an independent area is specified by an EXTENT job control statement, the end record is transferred to the independent overflow area. If an independent overflow area was not specified (or is filled) and the cylinder area is also filled, no room is made available to store the overflow record. ISAM posts this condition in the DTF table at filenameC. In all cases, ISAM determines whether room is available before any records are written.

If records are to be added to a blocked file, a work area must be specified by the DTFIS WORKL operand. Each added record must contain a key field in the same location as the records already in the file. You must specify the high-order position of the key field (relative to the leftmost position of the logical record). Use the DTFIS KEYLOC operand for this purpose.

When a WRITE macro is issued, ISAM first locates the correct track by referring to the necessary master (if available), cylinder, and track indexes. Then, a search on the key areas of the DASD records on the track is made to locate the desired block of records. The block of records is read into the I/O area. If IOREAL is included for reading and writing more than one record on DASD at a time, several blocks may be read into the I/O area.

ISAM then examines the key field within each logical record to find the exact position in which to insert the new record and then checks for any duplicate records. If a duplicate key exists, the condition is posted in filenameC. If the key of the record inserted (contained in WORKL) is low, the record is exchanged with the record presently in the block.

This procedure continues with each succeeding record in the block until the last record is moved into the work area. ISAM then updates the key area of the DASD record to reflect the highest key in the block. If IOAREAL was included, succeeding blocks in the I/O are also updated. The block (or blocks) is then written back onto DASD. The remaining blocks on the track are similarly processed until the last logical record on the track is moved into the work area. This record (set up as an overflow record with the proper sequence-links) is then moved to the overflow area. The indexes are updated and ISAM returns control to the program for the next record to be added. If the overflow area is filled, the information is posted in filenameC.

If the proper track for a record is an overflow track (determined by the track index), ISAM searches the overflow chain and checks for any duplication. If no duplication is found, ISAM writes the record (preceded by a sequence-link field in the data area of the DASD record) and adjusts the appropriate linkages to maintain sequential order by key. The new record is written in either the cylinder overflow area or an independent overflow area. If these areas are filled, this condition is posted in filenameC.

If the new record is higher than all records presently in the file (end-of-file), ISAM checks to determine whether the last track containing data records is filled. If it is not, the new record is added, replacing the end-of-file record. The end-of-file record is written in the next record location on the track, or on the next available prime data track. Another track must be available within the file limits. If the end-of-file record is the first record on any track, the new record is written in the appropriate overflow area. After each new record is inserted in its proper location, ISAM adjusts all indexes affected by the addition.

Random Retrieval of Records

Records in an ISAM file can be retrieved in random order for processing and/or updating. Retrieval must be specified in the DTFIS with the operand IOROUT=RETRVE or IOROUT=ADDRTR. Random processing must be specified in the DTFIS with the operand TYPEFLE=RANDOM or TYPEFLE=RANSEQ.

Because random reference to the file is by record key, your progam must supply the key of the desired record. To do this, the key must be stored in the key field specified by the DTFIS KEYARG operand. The specified key designates both the record to be retrieved and the record to be written back into the file in an updating operation. Adding and updating should not be interspersed. Records that are added to a file (between the READ and WRITE macros for a particular record to be updated) can result in a lost record and duplicate key.

The DTFIS RECSIZE operand should specify the same value as entered at load time. If these values differ, no error will result; however, the RECSIZE from the load DTFIS is used. The necessary information for a retrieval operation comes from the format-2 label and not the RETRVE operand in the DTFIS.

The READ macro causes ISAM to retrieve the specified record from the file. This macro requires two parameters. The first parameter specifies the name of the file from which the record is to be transferrred to virtual storage. This name is the same as the name specified in the DTFIS header entry for the file and can be specified as a symbol or in register notation. The second parameter must be the word KEY.

To locate a record, ISAM first searches the indexes to determine the track on which the record is stored and then searches the track for the specific record. When the record is found, ISAM transfers it to the I/O area specified by the DTFIS IOAREAR operand. The ISAM routines also move the record from the I/O area to the specified work area if the WORKR operand is included in the DTFIS.

When records are blocked, ISAM transfers the block that contains the specified record to the I/O area. It makes the individual record available for processing either in the I/O area or in the work area (if specified). For processing in the I/O area, ISAM supplies the address of the record in the register specified by the DTFIS IOREG operand. The ID of the record can be referenced using filenameG. A WAITF macro must follow a READ macro.

The WRITE macro with the parameter KEY is used for random updating. It causes ISAM to transfer the specified record from virtual storage to DASD. This macro requires two parameters. The first parameter specifies the name of the file to which the record is transferred. The specified name is the same as that used in the DTFIS header entry and in the preceding READ macro. The name can be specified as a symbol or in register notation. The second parameter must be the word KEY.

ISAM rewrites the record following a READ macro for the same file. The record is updated from the work area (if one is specified) or from the I/O area. The key need not be specified again ahead of the WRITE macro. A WAITF macro must follow a WRITE macro.

The WAITF macro is issued to ensure that record transfer is completed. Filename is the same name as that used in the DTFIS header entry, and can be specified as a symbol or in register notation.

This macro must be issued before your program attempts to process the input record which has been read or to build another output record for the designated file. The program does not regain control until the previous transfer of data is complete, unless ERREXT=YES is specified in the DTFIS and an error occurs. In this case, the ERET macro should be issued to handle the error and complete the transfer of data.

The WAITF macro posts any exceptional conditions in the DTFIS table at filenameC. The WAITF macro applies to the functions described in "Adding Records to a File" and "Random Retrieval of Records," above.

Sequential Retrieval of Records

Records of an ISAM file can be retrieved in sequential order by key for processing and/or updating. The DTFIS IOROUT=RETRVE operand must be specified. Sequential processing must be specified in the DTFIS TYPEFLE= operand by specifying SEQNTL or RANSEQ.

Although records are retrieved in order by key, sequential retrieval can start at a record in the file identified either by key or by the ID (identifier in the count area) of a record in the prime data area. Sequential retrieval can also start at the beginning of the logical file. You must specify, in SETL, the type of reference you use in your program.

Whenever the starting reference is by key and the file contains blocked records (RECFORM=FIXBLK), you must also provide ISAM with the position of the key records. This is specified in the DTFIS KEYLOC operand. To search for a record, ISAM first locates the correct block by the key in the key area of the DASD record. The key area contains the key of the highest record in the block. ISAM then examines the key field within each record in the block to find the specified record. As with random retrieval, the RECSIZE operand should specify the same number as indicated when the file was loaded.

The SETL (set limits) macro initiates the mode for sequential retrieval and initializes ISAM to begin retrieval at the specified starting address. The first operand (filename) specifies the same name as that used in the DTFIS header entry, as a symbol or in register notation. Register notation is necessary if your program is to be self-relocating.

The second operation specifies where processing is to begin.

If you are processing by the record ID, the operand id-name or (r) specifies the symbolic name of the 8-byte field in which you supply the starting (or lowest) reference for ISAM use. This field contains the information shown in Figure 4-19.

If processing begins with a key you supply, the second operand is KEY. The key is supplied in the field specified by the DTFIS KEYARG operand. If the specified key is not present in the file, an indication is given at filenameC.

BOF specifies that retrieval is to start at the beginning of the logical file.

Selected groups of records within a file containing identical characters or data in the first locations of each key can be selected by specifying GKEY (generic key) as the second operand. GKEY allows processing to begin at the first record (or key) within the desired group. You must supply a key that identifies the significant (high order) bytes of the required group of keys. The remainder (or insignificant) bytes of the key must be padded with blanks, binary zeros, or bytes lower in collating sequence than any of the insignificant bytes in the first key of the group to be processed. For example, a GKEY specification of D6420000 would permit processing to begin at the first record (or key) containing D642xxxx, regardless of the characters represented by the x's. Your program must determine when the generic group is completed. Otherwise, ISAM continues through the remainder of the file.

Note: If the search key is greater than the highest key on the file, the filenameC status byte is set to X'10' (no record found).

The ESETL (end set limit) macro should be issued before issuing a READ or WRITE if ADDRTR and/or RANDSEQ are specified in the same DTF. Another SETL can be issued to restart sequential retrieval. Sequential processing must always be terminated by issuing an ESETL macro.

The GET macro causes ISAM to retrieve the next record in sequence from the file. If records are to be processed in the I/O area (specified by the DTFIS IOAREAS operand), the only required parameter is the name of the file from which the record is to be retrieved. This is the same name as that specified in the DTFIS header entry and can be specified as a symbol or in register notation. ISAM transfers the record from the file to the I/O area after which the record is available for the execution of the next instruction in your program. The key is located at the beginning of IOAREAS and the register (IOREG) points to the data. If the records are blocked, ISAM makes each record available by supplying its address in the register specified by the DTFIS IOREG operand. The key is contained in the record.

If records are to be processed in a work area (specified by the DTFIS WORKS operand), it requires

two parameters, both of which can be specified as symbols or in register notation. The first parameter is the name of the file, and the second is the name of the work area. When using register notation, workname should *not* be preloaded into register 1.

If the records are blocked, each GET that transfers a block of records to virtual storage will also write the preceding block back into the file in its previous location if a PUT macro is issued for at least one of the records in that block. If no PUT macro was issued, updating is not required for the block and a GET does not rewrite the block. Whenever an unblocked record is retrieved from the prime data area, ISAM supplies the ID of that record in the field addressed by filenameH. If blocked records are specified, ISAM supplies the ID of the block.

The PUT macro is used for sequential updating of a file, and causes ISAM to transfer records to the file in sequential order. PUT returns a record to a file. A GET macro must precede each PUT macro.

If records are to be processed in the I/O area (specified by the DTFIS IOAREAS operand), PUT requires only the name of the file to which the records are to be transferred. The name is the same as that used in the DTFIS header entry and can be specified in register notation or as a symbol.

If records are to be processed in a work area, PUT requires two parameters, both of which can be specified either as a symbol or in register notation. The first parameter is the name of the file, and the second is the name of the work area. When using register notation, workname should not be loaded into register 1. The work area name may be the same as that specified in the preceding GET for the file, but this is not required. ISAM moves the record from the work area specified in the PUT macro to the I/O area specified for the file in the DTFIS IOAREAS operand.

When the records are unblocked, each PUT writes a record back onto the file in the same location from which it was retrieved by the preceding GET for the file. Thus, each PUT updates the last record that was retrieved from the file. If some records do not require updating, a series of GETs can be issued without intervening PUTs. Therefore, it is not necessary to rewrite unchanged records.

When the records are blocked, PUTs do not transfer records to the file. Instead, each PUT indicates that the block is to be written after all the records in the block are processed. When processing for the block is complete and a GET is issued to read the next block into virtual storage, the GET also writes the completed block back into the file in its previous location. If a PUT is not issued for any record in the block, GET does not write the completed block. The ESETL macro writes the last block processed, if necessary, before the end-of-file.

The ESETL (end set limit) macro ends the sequential mode initiated by the SETL macro. For filename specify the same name as was specified in the DTFIS header entry. The name can be specified as a symbol or in register notation. If the records are blocked, ESETL writes the last block back if a PUT was issued.

Note: If ADDRTR and/or RANSEQ are specified in the same DTF, ESETL should be issued before issuing a READ or WRITE; another SETL can be issued to restart sequential retrieval. Sequential processing must always be terminated by issuing an ESETL macro.

Completion

The CLOSE completion macro must be used after the processing of a file is completed. The CLOSE macro ends the association of the logical file declared in your program with a specific physical file on a DASD.

Byte	Identifier	Contents in Hexadecimal	Information
0	m	02-F5	Number of the extent in which the starting record is located.
1-2	bb	0000 (disk)	Always zero for disk
3-4	cc	 0000-00C7 (2311, 2314, 2319) 0000-0193 (3330, 3333) 0000-015B (3348 model 35) 0000-02B7 (3348 model 70)	Cylinder number for disk: for 2311, 2314, 2319; 0-199 for 3330, 3333; 0-403 for 3340 with 3348 model 35; 0-347 for 3340 with 3348 model 70; 0-695
5-6	hh	0000-0009 (2311) 0000-0013 (2314, 2319) 0000-0012 (3330, 3333) 0000-000B (3340)	Head position for disk.
7	r	01-FF	Record location.

Figure 4-19. Field supplied for SETL processing by record ID.

The CLOSE macro deactivates any file that was previously opened. A file may be closed at any time by issuing this macro. Once a file is closed, no further commands can be issued for the file unless it is reopened.

If a load or load extension file is not closed, the format-2 label associated with the file is not updated with the information that is in the DTF. Further processing of such a file may give unpredictable results.

CLOSER must be used if the file was activated by means of the OPENR macro.

See Appendix C for information on label processing done by the CLOSE macro.

Programming Considerations

Recordsize=keylength+(blocking factor x record length).

The maximum record size possible for ISAM on the various direct access devices is shown as follows:

Device	Maximum Record Size (in Number of bytes)
2311	3,605
2314	7,249
2319	7,249
3330	12,974
3333	12,974
3340	8,293

Formulas to facilitate estimating the disk storage requirements for an ISAM file on the various direct-access devices are given later in this section.

When writing ISAM programs, do not forget to include the DLBL job control statement. On any one given volume, only one prime data EXTENT is allowed. Information about the DLBL job control statement will be found in *VSE/Advanced Functions System Control Statements.* Examples of complete sets of job control statements for ISAM will also be found there.

VSE does not support a null ISAM file. If an attempt is made to access a null file, IOCS places the X'10' error indication in field filenameC.

Severe degradation can occur if too many records are added to an ISAM file without reorganizing it. To assist you in determining when reorganization of a file is required, ISAM maintains a helpful set of statistics. These statistics are maintained in the format-2 DASD label recorded with the file; when the file is processed, the statistics occupy fields within the DTFIS table. You can test these fields as you process the file. The fields, and the names by which you reference them are:

- prime record count (filenameP).
  A 4-byte count of the number of records in the prime data area. The field is used for DTFIS ADD, while a 4-byte count field at filenameP+4 is used for DTFIS LOAD.
- overflow record count (filenameO).
  A two-byte count of the number of records in the overflow area(s).
- available independent overflow tracks (filenameI).
  A two-byte count of the number of tracks remaining in the independent overflow area, if used.
- cylinder overflow areas full (filenameA).
  A two-byte count of the number of cylinder areas that are full, necessitating use of the independent overflow area.
- non-first-overflow reference (filenameR).
  A four-byte count of the number of times a random reference (READ) is made to records that are the second or higher links in an overflow chain.

In addition to these fields maintained automatically by ISAM, there is another field (filenameT) which you can use to keep a count of the records tagged for deletion. This field is kept in the format-2 DASD label recorded with the file and is available in the DTFIS table when the file is processed. You may tag the records for deletion by any method you desire, so long as the keys of the records are not changed in such a way that the sequence in the file would be altered. For instance, you could overwrite the data portion of a record with zeros; or a special field within a record could indicate that the record is deleted. You can keep a count of such records in filenameT. When reorganizing the file, tagged or deleted records can be eliminated. Additional information on reorganizing an ISAM file appears in Chapter 3.

ISAM Disk Storage Space Formulas

IBM 2311

Three formulas compute IBM 2311 disk storage requirements for an ISAM file. The known quantities for the computations given are:

D = Logical Record Length
K = Key Length
B = D x blocking factor
X = Number of prime data tracks, shared with track index and non-shared, per cylinder.

1. To calculate the number of prime data records per cylinder (Npr):

$Npr = A + C(X-1)$

Where:
A = Number of prime data records on a shared track
C = Number of prime data records on a non-shared track

Notes: These values must be whole numbers. A shared track is one in which prime data records occupy space not needed by the track index. The last track of the prime data area cannot be used during a load or an extension of a file. The programmer should issue the ENDFL macro and perform a load extend on the file.

To compute A:

a. Determine the size of the track index (T_1) in bytes:

$T_1 = (2X + 1)(81 + 1.049 (K + 10))$

If $T_1 > 3{,}625$, part of track 1 may be required for the track index in addition to track 0. The additional requirement on track 1 (T_1') is calculated as follows:

- Determine the number of track index entries (N) on track 0:

$$N = \frac{3{,}625}{81 + 1.049(K+10)} \quad \text{N rounded to next smaller integer}$$

- $T_1' = (2X-1)(81 + 1.049 (K + 10))-N(81 + 1.049 (K + 10))$

If $T_1' \leq 0$, only track 0 is needed for the track index (no index entries being required for track 0); you can adjust X and proceed to compute C.

If T_1' is still larger than 3,625, repeat the procedure with X reduced by 1.

If $T_1' > 0$, use T_1' instead of T_1 in step b.

b. Determine the number of bytes remaining on the track for prime data records (T_2):

$T_2 = 3{,}625-T_1$

c. Determine the size of the last prime record (T_3) on a track:

$T_3 = 20 + K + B$

If $(T_2-T_3) < 0$, set A = 0.

If $(T_2-T_3) = 0$, set A = 1.

$$\text{If } (T_2-T_3) > 0, \text{ set } A = 1 + \frac{T_2-T_3}{81 + 1.049(K+B)}$$

To compute C:

$$C = 1 + \frac{3{,}605-(K + B)}{81 + 1.049(K+B)}$$

2. To determine the number of overflow records per track (Nor):

$$Nor = 1 + \frac{3{,}605-(K + D + 10)}{81 + 1.049(K+D+10)}$$

3. To determine the number of cylinder or master index records per track (Nir):

$$Nir = 1 + \frac{3{,}605-(K + 10)}{81 + 1.049(K + 10)}$$

Note: Allow for a dummy record.

IBM 2314/2319

Three formulas compute IBM 2314/2319 disk storage requirements for an ISAM file. The known quantities for the computations given are:

D = Logical Record Length
K = Key Length
B = D x blocking factor
X = Number of prime data tracks, shared with track index and non-shared, per cylinder.

1. To calculate the number of prime data records per cylinder (Npr):

   $Npr = A + C(X-1)$

   Where:
   A = Number of prime data records on a shared track
   C = Number of prime data records on a non-shared track

   Notes: These values must be whole numbers. A shared track is one in which prime data records occupy space not needed by the track index. The last track of the prime data area cannot be used during a load or an extension of a file. The programmer should issue the ENDFL macro and perform a load extend on the file.

   To compute A:

   a. Determine the size of the track index (T_I) in bytes:

      $T_I = (2X + 1)(146 + 1.043\,(K + 10))$

      If $T_I > 7{,}294$, part of track 1 may be required for the track index in addition to track 0. The additional requirement on track 1 (T_I') is calculated as follows:

      - Determine the number of track index entries (N) on track 0:

        $N = \frac{7{,}249}{146 + 1.043(K + 10)}$ N rounded to next smaller integer

      - $T_I' = (2X-1)\,(146 + 1.043\,(K + 10)) - N(146 + 1.043\,(K + 10))$

      If $T_I' \leq 0$, only track 0 is needed for the track index (no index entries being required for track 0); you can adjust X and proceed to compute C.

      If T_I' is still larger than 7,294, repeat the procedure with X reduced by 1.

      If $T_I' > 0$, use T_I' instead of T_I in step b.

   b. Determine the number of bytes remaining on the track for prime data records (T_2):

      $T_2 = 7{,}294 - T_I$

   c. Determine the size of the last prime record (T_3) on a track:

      $T_3 = 45 + K + B$

      If $(T_2 - T_3) < 0$, set A = 0.

      If $(T_2 - T_3) = 0$, set A = 1.

      If $(T_2 - T_3) > 0$, set $A = 1 + \frac{T_2 - T_3}{146 + 1.043(K + B)}$

   To compute C:

   $C = 1 + \frac{7{,}249-(K+B)}{146 + 1.043(K + B)}$

2. To determine the number of overflow records per track (Nor):

   $Nor = 1 + \frac{7{,}249-(K+D+10)}{146 + 1.043(K + D + 10)}$

3. To determine the number of cylinder or master index records per track (Nir):

   $Nir = 1 + \frac{7{,}249-(K + 10)}{146 + 1.043(K + 10)}$

Note: Allow for a dummy record.

IBM 3330/3333

Three formulas compute IBM 3330/3333 disk storage requirements for an ISAM file. The known quantities for the computations given are:

D = Logical Record Length
K = Key Length
B = D x blocking factor
X = Number of prime data tracks, shared with track index and non-shared, per cylinder.

1. To calculate the number of prime data records per cylinder (Npr):

Npr=A + C(X-1) if there are shared data tracks
Npr=CX if there are no shared data tracks

Where:
A = Number of prime data records on a shared track
C = Number of prime data records on a non-shared track

Notes: These values must be whole numbers. A shared track is one in which prime data records occupy space not needed by the track index. The last track of the prime data area cannot be used during a load or an extension of a file. The programmer should issue the ENDFL macro and perform a load extend on the file.

To compute A:

a. Determine the size of the track index (T_1) in bytes:

$T_1 = (2X+1)(191+K+10)$

If $T_1 > 13{,}165$, part of track 1 may be required for the track index in addition to track 0. The additional requirement on track 1 (T_1') is calculated as follows:

- Determine the number of track index entries (N) on track 0:

$$N = \frac{13{,}165}{191 + K + 10)}$$ N rounded to next smaller integer

- $T_1' = (2X-1)\,(191+K+10))-N(191+K+10)$

If $T_1' \leq 0$, only track 0 is needed for the track index (no index entries being required for track 0); you can adjust X and proceed to compute C.

If T_1' is still larger than 13,165, repeat the procedure with X reduced by 1.

If $T_1' > 0$, use T_1' instead of T_1 in step b.

b. Determine the number of bytes remaining on the track for prime data records (T_2):

$T_2 = 13{,}165\text{-}T_1$

$$A = \frac{T_2}{191 + K + B}$$

To compute C:

$$C=1+\frac{13{,}165\text{-}(K + B)}{191 + K + B)}$$

2. To determine the number of overflow records per track (Nor):

$$Nor = 1 + \frac{13{,}165}{191 + K + D + 10}$$

3. To determine the number of cylinder or master index records per track (Nir):

$$Nir = 1 + \frac{13{,}165}{191 + K + 10}$$

IBM 3340

Three formulas compute IBM 3340 disk storage requirements for an ISAM file. The known quantities for the computations given are:

D = Logical Record Length
K = Key Length
B = D x blocking factor
X = Number of prime data tracks, shared with track index and non-shared, per cylinder.

1. To calculate the number of prime data records per cylinder (Npr):

   Npr=A + C(X-1) if there are shared data tracks
   Npr=CX if there are no shared data tracks

   Where:
   A = Number of prime data records on a shared track
   C = Number of prime data records on a non-shared track

   Notes: These values must be whole numbers. A shared track is one in which prime data records occupy space not needed by the track index. The last track of the prime data area cannot be used during a load or an extension of a file. The programmer should issue the ENDFL macro and perform a load extend on the file.

   To compute A:

   a. Determine the size of the track index (T_I) in bytes:

      $T_I = (2X+1)(242+K+10)$

      If $T_I > 8{,}535$, part of track 1 may be required for the track index in addition to track 0. The additional requirement on track 1 (T_I') is calculated as follows:

      - Determine the number of track index entries (N) on track 0:

        $N = \frac{8{,}535}{242 + K + 10}$ N rounded to next smaller integer

      - $T_I' = (2X-1)(242+K+10)-N(242+K+10)$

      If $T_I' \leq 0$, only track 0 is needed for the track index (no index entries being required for track 0); you can adjust X and proceed to compute C.

      If T_I' is still larger than 8,535, repeat the procedure with X reduced by 1.

      If $T_I' > 0$, use T_I' instead of T_I in step b.

   b. Determine the number of bytes remaining on the track for prime data records (T_2):

      $T_2 = 8{,}535-T_I$

   $A = \frac{T_2}{242 + K + B}$

   To compute C:

   $C = 1 + \frac{8{,}535}{242 + K + B}$

2. To determine the number of overflow records per track (Nor):

   $Nor = 1 + \frac{8{,}535}{242 + K + D + 10}$

3. To determine the number of cylinder or master index records per track (Nir):

   $Nir = 1 + \frac{8{,}535}{242 + K + 10}$

Chapter 5: Processing Diskette Files

Before any processing can be done on a diskette file, it must be defined by a DTF macro. The DTFDU macro defines sequential processing for a diskette file; its operands are listed in Figure 5-1. Alternatively, you may use DTFDI to provide device independence for system logical units. See Chapter 8 for more information on device-independent files. Finally, you must use DTFPH if you plan to process a diskette file with the PIOCS (Physical IOCS) macros. (See Chapter 9.)

There are two logic module generation macros associated with DTFDU, DUMODFI for input files, and DUMODFO for output files. For details of these and other macros, see *VSE/Advanced Functions Macro Reference.*

Note that special records (deleted or sequential relocated records) on an input file are skipped, and not passed to the user. The DTFDU macro cannot be used when a card image diskette file is to be processed under VSE/POWER.

Opening a File

Output

When a multi-volume diskette file is created, feeding from diskette to diskette is automatically performed by IOCS. If the file was defined by a DTFDI macro, the last diskette is ejected automatically by IOCS. If the DTFDU macro was used, the ejection of the last diskette is controlled by the FEED operand of this macro.

When a file is opened, OPEN checks the VTOC on the diskette and:

- ensures that the file to be created does not have the same name as an existing unexpired file.
- ensures there is at least one track available to be allocated.
- allocates space for the file, starting at the track following the last unexpired or write-protected file on the diskette.

Applies to				
Input	Output			
X		M	EOFADDR=xxxxxxxx	Name of your end-of-file routine. (Required for input only).
X	X	M	IOAREA1=xxxxxxxx	Name of first I/O area.
X	X	M	RECSIZE=nnn	Length of one record in bytes.
X	X	O	CMDCHN=nn	Number of read/write CCWs (records) to be command-chained.
X	X	O	DEVADDR=SYSxxx	Symbolic unit, required only when not provided on an EXTENT statement.
X	X	O	DEVICE=3540	Must be 3540. If omitted, 3540 is assumed.
X	X	O	ERREXT=YES	Indicates additional errors and ERET desired. Specify ERROPT.
X	X	O	ERROPT=xxxxxxxx	IGNORE, or SKIP, or name of error routine.
X	X	O	FEED=xxx	YES means feed at end-of-file. NO means no feed. YES assumed if omitted.
	X	O	FILESEC=YES	YES means create file secure.
X	X	O	IOAREA2=xxxxxxxx	Name of second I/O area, if two areas are used.
X	X	O	IOREG=(nn)	General register 2 to 12 in parentheses. Omit WORKA.
X	X	O	MODNAME=xxxxxxxx	Name of DUMODFx logic module for this DTF. If omitted, IOCS generates standard name.
X	X	O	RDONLY=YES	Generates a read-only module. Requires a module save area for each task using the module.
X	X	O	SEPASMB=YES	DTFDU is to be assembled separately.
X	X	O	TYPEFLE=xxxxxx	INPUT or OUTPUT. If omitted, INPUT is assumed.
X		O	VERIFY=YES	3741/3742 input is verified.
X		O	VOLSEQ=YES	YES means OPEN is to check sequencing of multivolume files.
X	X	O	WORKA=YES	GET or PUT specifies work area. Omit IOREG.
	X	O	WRTPROT=YES	File will be created with Write-Protect on (cannot be overwritten).

M = Mandatory
O = Optional

Figure 5-1. DTFDU macro operands.

After this check, OPEN creates the format-1 label for the file and writes the label in the VTOC. Each time you determine that all processing for an extent is complete, you must feed to make the next diskette available and then issue another OPEN for the file, to make the next extent available. CLOSE will automatically cause the last volume to be fed out. If the last extent of the file is completely processed before a CLOSE is issued, OPEN assumes an error condition and the job is canceled.

Input

When processing input files on diskettes with physical IOCS, OPEN is used to check standard labels.

When a multi-volume diskette file is read, feeding from diskette to diskette is automatically performed by IOCS. If the file was defined by a DTFDI macro, the last diskette is not ejected. If the DTFDU macro was used, the ejection is controlled by the FEED operand of this macro.

When the first volume is opened, OPEN checks the VTOC on the diskette and determines the extent limits of the file from the file label.

After the label is checked, OPEN makes the first extent available for processing. Each time you determine that all processing for a diskette is complete, you must feed to make the next diskette available, and then issue another OPEN for the file, to make the next extent available. If another extent is not available, OPEN stores the character F (for EOF) in byte 31 of the DTF table. You can determine the end of file by checking the byte at filename +30.

The extents are made available in the order of the sequence number on the extent statements (if the statements are not numbered, job control numbers them consecutively). The same extent statements used to build the file can be used when the file is used as input.

Processing Records with Command Chaining

Elsewhere in this manual the concept of a block of data has been discussed. Except for FBA direct access storage devices, a block is treated as a physical entity which is read or written as a *unit* on the external device. To apply this concept correctly to diskette files, you must be aware of the following -- you can achieve faster performance by allowing LIOCS to read and write multiple records each time the diskette is accessed. Physically, each logical data record that the user reads or writes is a separate record on the diskette. By allowing LIOCS to *chain* I/O operations to the device on input, the user-provided I/O area will be filled each time the device is accessed. On output, LIOCS waits until the I/O area provided is full before writing individual logical records on the diskette.

When records on diskettes are specified as command chained in the DTFDU CMDCHN operand, each individual record must be located for processing. Therefore, command chained records are handled as follows:

1. The first GET macro transfers a chain of records (2, 13, or 26 records depending on the CMDCHN operand) from diskette to the input area. It also initializes the specified register to the absolute address of the first data record, or it transfers the first record to the specified work area.
2. Subsequent GET macros either add an indexing factor to the register or move the proper record to the specified work area, until all records in the block are processed.
3. Then, the next GET makes a new chain of records available in virtual storage, and either initializes the register or moves the first record.

So, for diskette files you can logically "block" records in the I/O areas by chaining I/O operations; however, each record on the diskette remains a physically separate entity.

If, for example, you want to process 80-byte records, you could establish two I/O areas, each 160 bytes in length, and you could indicate chaining of two records. Then, when a record is requested for input, data transfer would occur as illustrated in Figure 5-2.

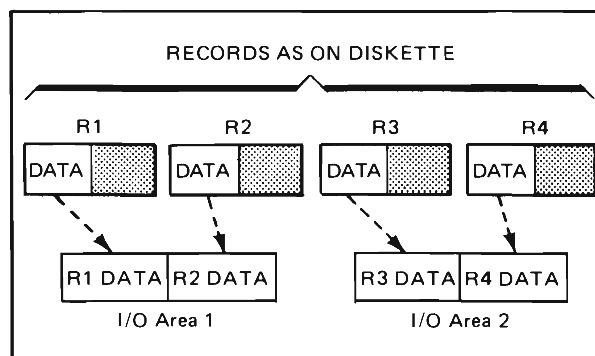


Figure 5-2. Diskette data transfer on input.

Physically, the diskette records are always 128 bytes in length (See Figure 5-4.) Because only 80 bytes are desired, only the first 80 bytes of each physical record are placed in the I/O areas.

For output, when an I/O area is full, records are written on the diskette as shown in Figure 5-3.

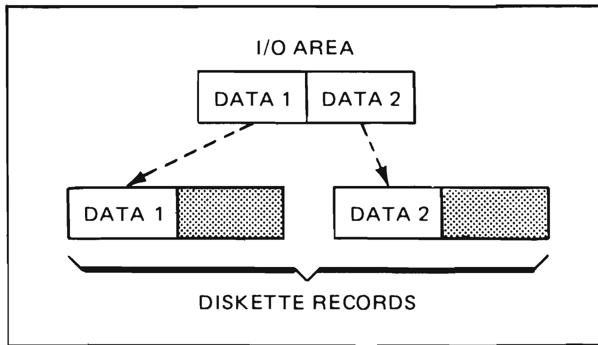


Figure 5-3. Diskette data transfer on output.

When you want to take advantage of this command chaining, you can chain either 2, or 13, or 26 diskette records and have them processed — read or written — together as a group: specify the desired number in the CMDCHN operand of the DTFDU declarative macro. In conjunction with this operand, specify either the IOREG operand if you want the records to be processed in an I/O area (one or two I/O areas), or the WORKA operand if you want the records to be processed in a workarea.

TRACKS PER VOLUME		77
Track 0	System Use	
Tracks 1 - 73	Data Records	
Track 74	Reserved	
Tracks 75 - 76	Alternates for Defective Tracks	
RECORDS PER TRACK		26
BYTES	PER RECORD	128
	PER TRACK	3,328
	PER VOLUME	242,944

Figure 5-4. Diskette layout and storage capacity.

The first GET macro transfers a chain of records from diskette to the input area. It also initializes the specified register to the absolute address of the first data record, or it transfers the first record to the specified workarea. Subsequent GET macros either add an indexing factor to the register or move the proper record to the specified work area, until all records in the block are processed. Then, the next GET makes a new chain of records available in virtual storage, and either initializes the register or moves the first record.

The PUT macro accomplishes this data transfer for output in the same way. That is, when command chained records are written on diskettes, the individually built records must be formed into a chain in the output area before they can be transferred to the output file.

Command chained records can be built directly in an output area or in a work area. Each PUT macro for these records either adds an indexing factor to the register (IOREG), or moves the completed record to the proper location in the output area. When an output chain of records is complete, a PUT macro causes the chain of records to be transferred to the output file and initializes the register, if one is used.

Closing a File

For diskette files, CLOSE sets the multivolume indicator in the HDR1 label to indicate the last volume of the file. Then, it sets up the end-of-data address in the HDR1 label and feeds the last diskette, determined by the FEED operand in the DTF macro.

Error Handling

By specifying the ERREXT and ERROPT operands in the DTFDU and logic module generation macros, LIOCS assists you in processing permanent I/O errors.

Specifying ERREXT=YES in DTFDU and DUMODFx enables your ERROPT routine to return to DUMODFx with the ERET macro. It also enables permanent errors to be indicated to your program. For ERREXT facilities, the ERROPT operand must be specified. However, to take full advantage of this option use the ERROPT=name operand.

Specify the DTF ERROPT operand if you do not want a job to be terminated when a permanent error cannot be corrected in the diskette error routine. If attempts to reread a chain of records are unsuccessful, the job is terminated unless the ERROPT entry is included. Either IGNORE, SKIP, or the name of an error routine can be specified. The functions of these parameters are described below.

IGNORE
: The error condition is ignored. The records are made available for processing. On output, the error condition is ignored and the records are considered written correctly.

SKIP
: No records in the error chain are made available for processing. The next chain of records is read from the diskette, and processing continues with the first record of that chain. On output, the SKIP option is the same as the IGNORE option.

name
: IOCS branches to your error routine named by this parameter regardless of whether or not ERREXT=YES is specified. In this routine you

can process or make note of the error condition as desired.

If ERREXT is not specified, register 1 contains the address of the first record in the error chain. When processing in the ERROPT routine, you reference records in the error chain by referring to the address supplied in register 1. The contents of the IOREG register or work area are variable and should not be used to process error records. Also, GET macros must not be issued for records in the error chain. If any other IOCS macros (excluding ERET if ERREXT=YES) are used in this routine, the contents of register 12 (with RDONLY) and 14 must be saved and restored after their use. At the end of the routine, return control to IOCS by branching to the address in register 14. For a read error, IOCS skips that error chain of records, and makes the first record of the next chain available for processing in the main program.

If ERREXT is specified, register 1 contains the address of a two part parameter list containing the 4-byte DTFDU address and the 4-byte address of the first record in the error chain. Register 14 contains the return address. Processing is similar to that described above except for addressing the records in error.

At the end of its processing, the error processing routine returns to LIOCS by issuing the ERET macro.

For an input file, the program:

- skips the error chain and reads the next chain with an ERET SKIP, or
- ignores the error with an ERET IGNORE, or
- it makes another attempt to read the error chain with an ERET RETRY.

For an output file the only acceptable parameters are IGNORE or name, and the program:

- ignores the error condition with ERET IGNORE or ERET SKIP, or
- attempts to write the error chain with an ERET=RETRY. Bad spot control records (1, 2, 13, or 26 records depending on the CMDCHN specification) are written at the current diskette address, and the write chain is retried in the next 1, 2, 13, or 26 (depending on the CMDCHN specification) sectors on the diskette.

The DTFDU error options are shown in Figure 5-5.

To Terminate the job,	specify nothing;
Skip the error record,	specify ERROPT=SKIP;
Ignore the error record,	specify ERROPT=IGNORE;
Process the error record,	specify ERROPT=name;
After processing the record, to leave the error-processing routine and	
Skip the (input) record,	execute ERET SKIP;
Ignore the record,	execute ERET IGNORE;
Retry reading or writing the record,	execute ERET RETRY.

Figure 5-5. DTFDU error options.

Chapter 6: Processing Magnetic Tape Files

Before any processing can be done on a magnetic tape file, it must be defined by the DTFMT macro and the logic module generation macro, MTMOD. The operands for DTFMT are listed in Figure 6-1, and the operands for MTMOD are listed in Figure 6-2. For details about these and other macros, see *VSE/Advanced Functions Macro Reference.*

Label Processing

You can issue an LBRET macro in your program when you have completed processing labels and wish to return control to IOCS. LBRET applies to subroutines that write or check magnetic tape user-standard or nonstandard labels. The operand used — 1, 2, or 3 — depends on the function to be performed. The functions and operands are explained below. See also the section "Label Processing" in Appendix C.

Checking User Standard Tape Labels: IOCS reads and passes the labels to you one at a time until a tapemark is read, or until you indicate that you do not want any more labels. Use LBRET 2 if you want to process the next label. If IOCS reads a tapemark, label processing is automatically terminated. Use LBRET 1 if you want to bypass any remaining labels.

Writing User Standard Tape Labels: Build the labels one at a time and return to IOCS, which writes the labels. When LBRET 2 is used, IOCS returns control to you (at the address specified in LABADDR) after writing the label. Use LBRET 1 to terminate the label set.

Writing or Checking Nonstandard Tape Labels: You must process all your nonstandard labels at once. Use LBRET 2 after all label processing is completed and you want to return control to IOCS. Appendix C shows an example of this.

Block Size

The BLKSIZE operand of DTFMT specifies the number of bytes transferred to or from the I/O area and tape. If a READ or WRITE macro specifies a length greater than the BLKSIZE value for work files, the record to be read or written will be truncated to fit in the I/O area. The maximum block size is 32,767 bytes. The minimum size of a physical tape record (gap to gap) is 12 bytes. A record of eleven bytes or less is treated as noise.

For output processing of variable records, the minimum physical record length is 18 bytes. If less than 18 bytes are specified for variable blocked or variable unblocked records, BLKSIZE=18 is assumed.

For output processing of spanned records, the minimum physical record length is 18 bytes. If SPNBLK or SPNUNB and TYPEFLE=OUTPUT are specified in the DTFMT and the BLKSIZE is invalid or less than 18 bytes, an MNOTE is generated and BLKSIZE=18 is assumed.

For ASCII tapes, the BLKSIZE includes the length of any block prefix or padding characters present. If ASCII=YES and BLKSIZE is less than 18 bytes (for fixed-length records only) or greater than 2048 bytes, an MNOTE is generated because this length violates the limits specified by American National Standards Institute, Inc.

Reading Magnetic Tape Backwards

If records on magnetic tape are read backwards (the DTF operand READ=BACK is specified), blocks of records are transferred from tape to virtual storage in reverse order. The last block is read first, the next-to-last block is read second, etc. For blocked records, each GET macro also makes the individual records available in reverse order. The last record in the input area is the first record available for processing (either by indexing or in a work area).

Any 9-track tape can be read backwards. 7-track tape can be read backwards only if the data conversion special feature was not used when the tape was written.

Forcing End-Of-Volume

The FEOV (force end-of-volume) macro is used for either input or output files on magnetic tape drives, to which programmer logical units were assigned, to force an end-of-volume condition before sensing a tapemark or reflective marker. This indicates that processing of records on the current volume is finished, but that more records for the same logical file are to be read from, or written on, a following volume. If a spanned record is begun on an output file and there is not enough space to contain it, MTMOD issues an FEOV at the end of the last completed spanned record. The last spanned record (for which there was no room) is rewritten on a new volume.

The name of the file, specified in the header entry, is the only parameter required. The name can

Applies to					
Input	Output	Work			
X	X	X	M	BLKSIZE=nnnnn	Length of one I/O area in bytes (maximum = 32,767).
X	X	X	M	DEVADDR=SYSxxx	Symbolic unit for tape drive used for this file.
X		X	M	EOFADDR=xxxxxxxx	Name of your end-of-file routine.
X	X	X	M	FILABL=xxxx	(NO, STD, or NSTD). If NSTD specified, include LABADDR. If omitted, NO is assumed.
X	X		M	IOAREA1=xxxxxxxx	Name of first I/O area.
X	X		O	ASCII=YES	ASCII file processing is required.
X	X		O	BUFOFF=nn	Length of block prefix if ASCII=YES.
X			O	CKPTREC=YES	Checkpoint records are interspersed with input data records. IOCS bypasses checkpoint records.
X	X	X	O	ERREXT=YES	Additional errors and ERET are desired.
X	X	X	O	ERROPT=xxxxxxxx	(IGNORE, SKIP, or name of error routine). Prevents job termination on error records.
X	X	X	O	HDRINFO=YES	Print header label information if FILABL=STD.
X	X		O	IOAREA2=xxxxxxxx	If two I/O areas are used, the name of the second area.
X	X		O	IOREG=(nn)	General registers 2-12, written in parentheses. Use only if GET or PUT does not specify work area or if two I/O areas are used. Omit WORKA.
X	X		O	LABADDR=xxxxxxxx	Name of your label routine if FILABL=NSTD, or if FILABL=STD and user-standard labels are processed.
X			O	LENCHK=YES	Length check of physical records if ASCII=YES and BUFOFF=4.
X	X	X	O	MODNAME=xxxxxxxx	Name of MTMOD logic module for this DTF. If omitted, IOCS generates standard name.
		X	O	NOTEPNT=xxxxxx	(YES or POINTS). YES if NOTE, POINTW, POINTR, or POINTS macro used. POINTS if only POINTS macro used.
X	X	X	O	RDONLY=YES	Generate read-only module. Requires a module save area for each task using the module.
X		X	O	READ=xxxxxx	(FORWARD or BACK). If omitted, FORWARD assumed.
X	X	X	O	RECFORM=xxxxxx	(FIXUNB, FIXBLK, VARUNB, VARBLK, SPNUNB, SPNBLK, or UNDEF). For work files use FIXUNB or UNDEF. If omitted, FIXUNB is assumed.
X	X		O	RECSIZE=nnnn	If RECFORM=FIXBLK, number of characters in record. If RECFORM=UNDEF, general registers 2-12, written in parentheses. Not required for other records.
X	X	X	O	REWIND=xxxxxx	(UNLOAD or NORWD). Unload on CLOSE or end-of-volume, or prevent rewinding. If omitted, rewind only.
X	X	X	O	SEPASMB=YES	DTFMT is to be assembled separately.
	X		O	TPMARK= {YES|NO}	Causes IOCS to write or to omit a tapemark ahead of data records if FILABL=NSTD or NO is specified.
X	X	X	O	TYPEFLE=xxxxxx	(INPUT, OUTPUT, or WORK). If omitted, INPUT is assumed.
	X		O	VARBLD=(nn)	General registers 2-12 written in parentheses, if RECFORM=VARBLK and records are built in the output area.
X			O	WLRERR=xxxxxxxx	Name of wrong-length-record routine.
X	X		O	WORKA=YES	GET or PUT specifies work area. Omit IOREG.

M = Mandatory
O = Optional

Figure 6-1. DTFMT macro operands.

Applies to					
Input	Output	Work			
X	X		O	ASCII=YES	ASCII file processing is required.
X			O	CKPTREC=YES	Checkpoint records are interspersed with input data records. IOCS bypasses checkpoint records.
X	X	X	O	ERREXT=YES	Indication of additional errors and ERET are desired.
X	X	X	O	ERROPT=xxxxxxxx	(IGNORE, SKIP, or name of error routine). Prevents job termination on error records.
		X	O	NOTEPNT=xxxxxx	(YES or POINTS). YES if NOTE, POINTW, POINTR, or POINTS macro used. POINTS if only POINTS macro used.
X	X	X	O	RDONLY=YES	Generate read-only module. Requires a module save area for each task using the module.
X		X	O	READ=xxxxxx	(FORWARD or BACK). If omitted, FORWARD assumed.
X	X	X	O	RECFORM=xxxxxx	(FIXUNB, FIXBLK, VARUNB, VARBLK, SPNUNB, SPNBLK, or UNDEF). For work files use FIXUNB or UNDEF. If omitted, FIXUNB is assumed.
X	X	X	O	TYPEFLE=xxxxxx	(INPUT, OUTPUT, or WORK). If omitted, INPUT is assumed.
X	X	X	O	SEPASMB=YES	MTMOD is to be assembled separately.
X	X		O	WORKA=YES	GET or PUT specifies work area. Omit IOREG.

M = Mandatory
O = Optional

Figure 6-2. MTMOD macro operands.

Magnetic Tape Device	Maximum Data Rates		Control Unit
	Kilo-bytes per second	Bytes per inch	
IBM 2401,M. 1	30	800	2803 or 2804
2	60	800	
3	90	800	
4	60	1,600	
5	120	1,600	
6	180	1,600	
8	60	800	
IBM 2415,M. 1-3	15	800	*
4-6	30	1,600	
IBM 2420,M. 5	160	1,600	2803
7	320	1,600	
IBM 3410/3411,M. 1	20	1,600	**
2	40	1,600	
3	80	1,600	
IBM 3420,M. 3	120	1,600	3803
4	470	6,250	
5	200	1,600	
6	780	6,250	
7	320	1,600	
8	1,250	6,250	
IBM 8809	20/160	1,600	***

* Control unit is included in the device.
** A first tape drive in a string of 3410's must be a 3411 which is a 3410 with a control unit.
*** Native attachment via file/tape adapter.

Figure 6-3. Characteristics of magnetic tape devices.

be specified either as a symbol or in register notation.

When LIOCS macros are used for a file, FEOV initiates the same functions that occur at a normal end-of-volume condition, except for checking of trailer labels.

For an input tape, FEOV immediately rewinds the tape (as specified by REWIND) and provides for a volume change (as specified by the ASSGN cards). Trailer labels are not checked. FEOV then checks the standard header label on the new volume and allows you to check any user-standard header labels if LABADDR is specified. If nonstandard labels are specified (FILABL=NSTD), FEOV allows you to check these labels as well.

For an output tape, FEOV writes:

- A tapemark (two tapemarks for ASCII files.)
- A standard trailer label and user-standard labels (if any).
- A tapemark.

If the volume is changed, FEOV then writes the header label(s) on the new volume (as specified in the DTFMT REWIND, FILABL, LABADDR operands, and the ASSGN cards). If nonstandard labels are specified, FEOV allows you to write trailer labels on the completed volume and header labels on the new volume, if desired.

Error Handling

Certain DTFMT and MTMOD macro operands are provided to assist you in processing I/O and record-length errors. Before discussing the use of these macro operands in detail, it is important that you understand the basic alternatives open to you regarding the handling of magnetic tape file errors.

For example, the first decision you must make is whether or not you want to code your own error processing routine for the file and have LIOCS exit to it when an error condition occurs.

The alternative to doing your own error processing is to rely on LIOCS to satisfy the handling of error conditions in a more general and limited way. However, even if you choose this alternative, you still have some options open to you and these will be discussed more fully when the use of a specific macro operand is described.

The DTFMT and MTMOD macro operands which you may use to achieve the desired error processing are: ERREXT, WLRERR (only in the DTF macro), and ERROPT.

By including ERREXT=YES in your DTFMT and MTMOD macros, you indicate to LIOCS that control can be returned from your error processing routine by means of the ERET imperative macro. If you omit the ERREXT operand, a return to LIOCS from your error processing routine must be accomplished with a branch to the address contained in register 14.

Including ERREXT=YES also causes nonrecoverable I/O errors occurring before data transfer takes place to be indicated to your error processing routine.

To take full advantage of the capabilities of the ERREXT operand, you must also include the ERROPT=name operand.

If you specify ERREXT=YES, when an error condition occurs, register 1 will contain the address of a 2-part parameter list. The first four bytes of the list is the address of the DTF table and the second four bytes is the address of the physical record in error. You can make use of both addresses in your error routine, the first address to interrogate specific indicators in the CCB (the first 16 bytes of the DTF table — see Figures 9-3 and 9-4), and the second address to access the record for error processing.

If you omit the ERREXT operand, when an error condition occurs, register 1 will contain the address of the physical record in error. In this case, your routine must use register 1 to access the record for error processing.

Note: If ERREXT is not specified for an output file, no code is generated and an MNOTE is issued. If an error condition occurs, the job is canceled.

The WLRERR=name operand (only in the DTF macro) is used only in conjunction with magnetic tape input files. With it you identify your routine for processing wrong-length records. If you omit this operand, one of the following actions will occur if a wrong-length record is detected:

- If the ERROPT operand is also omitted, the wrong-length record condition is ignored, or
- If the ERROPT operand is included for this file, the wrong-length record is treated as an error block and it is handled according to your specification for an error (see the discussion of the ERROPT operand which follows).

The ERROPT operand is used to indicate your choice of action if an error block is encountered.

If either FILABL=STD, or CKPTREC=YES, or both, is specified, the error block is included in the block count. After this, the job is automatically terminated unless the ERROPT operand is included to specify other procedures to be followed in case of an error condition. Either, IGNORE, SKIP, or the symbolic name of an error routine can be specified; however, for output files, only IGNORE and name are valid parameters. The functions of these specifications are:

IGNORE

For input files, the error condition is completely ignored and records are made available for processing by your main program. When reading spanned records, the entire spanned record or block of spanned records is returned to you rather than just the one physical record in which the error occurred.

When writing spanned records, the error is ignored and the physical record containing the error is treated as a valid record. The remainder, if any, of the spanned record segments are written, if possible.

SKIP

For input files, no records in the error block are made available for processing by your main program. The next block is read from the tape and processing continues with the first record of that block. The error block is included in the block count. When reading spanned records, the entire spanned record or block of spanned records is skipped rather than just the one physical record in which the error occurred.

When writing spanned records, the error is ignored and the physical record containing the error is treated as a valid record. The remainder, if any, of the spanned record segments are written.

name
: IOCS branches to your error processing routine named by this parameter (regardless of whether or not you have included ERREXT=YES). In your routine, you process or make note of the error condition as you wish.

Programming Your Error Processing Routines

You may perform any kind of error processing you want in your error routine; however, you must abide by certain rules and restrictions.

- In your error processing routine, you must not issue a GET to the file.
- If your routine issues any other IOCS macros (excluding ERET when you have specified ERREXT=YES), the contents of register 13 (with RDONLY) and register 14 must be saved before and restored after the macros are used.
- If your routine issues I/O macros which use the same read-only module that caused control to pass to the error routine, you must provide another save area. One save area is used for the normal I/O operations and the second for I/O operations in the error routine itself. Before returning to the module that entered the error routine, register 13 must be set to the save area address originally specified for the task.
- If you have specified ERREXT=YES, register 1 contains the address of the 2-part parameter list; otherwise, register 1 contains the address of the physical record in error. You must access the error block, or the records in the error block, by the address in the parameter list or in register 1. (The content of the IOREG register or work area — if either is specified — is unpredictable and, therefore, should not be used for error processing.) When spanned records are processed, the address is that of the whole spanned record, blocked or unblocked.

Note: For ASCII tapes, the pointer to the block in error indicates the first logical record following the block prefix.

- If ERREXT=YES, the data transfer bit (byte 2, bit 2) of the DTF table (CCB) should be tested to determine whether a nonrecoverable I/O error has occurred. If the bit is on, the physical record in error has not been read or written. If the bit is off, data was transferred and your routine must address the physical record in error to determine the action to be taken.
- At the conclusion of error processing, your routine must return control to LIOCS either by branching to the address in register 14 or, if you have specified ERREXT=YES, via the ERET IGNORE or SKIP option.

Note: The RETRY option of the ERET macro is not valid for magnetic tape files and results in job termination if used.

Wrong-length Error Processing Considerations

If the block read is shorter than the length specified in the BLKSIZE operand, the first two bytes of the DTF table (CCB) contain the number of bytes left to be read (residual count). Therefore, the size of the actual block is equal to the specified block size minus the residual count. If the block read is longer than the length specified in the BLKSIZE operand, the residual count is zero, and there is no way to compute the actual size of the block. In this latter case, the number of bytes transferred is equal to the length specified in the BLKSIZE operand, and the remainder of the block is lost (truncated).

When fixed-length unblocked records are specified (RECFORM=FIXUNB), a wrong-length record error condition is given when the length of the record read is not the same as specified in the BLKSIZE operand. For EBCDIC fixed-length blocked records, record length is considered incorrect if the physical tape record (gap-to-gap) that is read is not a multiple of the logical record length, as specified in the RECSIZE operand. This permits reading of short blocks of logical records without a wrong-length record indication.

For EBCDIC variable-length records (blocked and unblocked), the record length is considered incorrect if the length of the tape record is not the same as the block length specified in the 4-byte block length field. The residual count can be obtained by addressing the halfword in the DTF table at *filename*+98.

For ASCII variable-length records (blocked and unblocked), a check on the physical record length is made if LENCHK=YES is specified. The physical record length is considered incorrect if the tape record is not the same as the block length specified in the 4-byte block prefix. In this case, the WLR bit (byte 5, bit 1) in the DTF table is set off.

For undefined records, a wrong-length record is indicated if the record read is longer than the size specified in the BLKSIZE operand.

Other Error Processing Considerations

- If a parity error is detected when a block of records is read, the tape is backspaced and reread a specified number of times (device ERP) before the block is considered an error block.

  Output parity errors are considered to be an error block if they exist after IOCS attempts to forward erase and write the tape output a specified number of times (device ERP). Under this condition, your error processing routine must treat the device as inoperative and must not attempt further processing on it. Any subsequent attempt to return to MTMOD results in job termination.

- A sequence error may occur if LIOCS is searching for a first segment of a logical spanned record and fails to find it. If you have specified either WLRERR=name or ERROPT=name, the error recovery procedure is the same as for wrong-length record errors. If you have specified neither WLRERR=name nor ERROPT=name, LIOCS ignores the sequence error and searches for the next first segment.

Figure 6-4 summarizes the DTFMT error options for various combinations of error specifications and errors.

To Terminate the job,	specify nothing;
Skip the error record,	specify ERROPT=SKIP;
Ignore the error record,	specify ERROPT=IGNORE;
Process the error record, and/or (for wrong-length record error)	specify ERROPT=name, specify WLRERR=name.
After processing the record, to leave the error-processing routine and	
Skip the record (input only),	execute BR 14, or execute ERET SKIP;
Ignore the record,	execute ERET IGNORE;
Retry the record,	execute ERET RETRY.

Figure 6-4. DTFMT error options.

Non-Data Device Operations

By using the CNTRL macro, your program can control a number of magnetic tape handling operations that are not concerned with reading or writing data. The format of the CNTRL macro is as follows:

```
[name]    CNTRL         {filename|(1)},code
```

Where *code* is one of the 3-character mnemonic codes in the following list of function categories:

Rewinding tape to the load point:
- REW - Rewind
- RUN - Rewind and unload

Moving tape to a specified position:
- BSR - Backspace to interrecord gap
- BSF - Backspace to tapemark
- FSR - Forward space to interrecord gap
- FSF - Forward space to tapemark

Forward or backward logical record spacing:
- FSL - Forward space logical record
- BSL - Backward space logical record

Writing a tapemark:
- WTM - Write tapemark

Erasing a portion of the tape:
- ERG - Erase gap (writes blank tape)

The tape rewind (REW and RUN) and tape movement (BSR, BSF, FSR, and FSF) functions can be used before a tape file is opened. This allows the tape to be positioned at the desired location for opening a file, so that:

- The tape can be positioned to a file located in the middle of a multifile-reel.
- Rewinding of the tape can be performed even if NORWD was specified in the DTF REWIND operand.

Note: If you are using a self-relocating program, you must open the file before issuing any commands for it.

The tape movement functions (BSR, BSF, FSR, and FSF) apply only to input files, and the following should be considered:

1. The FSF (or BSR) function permits you to skip over a physical tape record (from one interrecord gap to the next). The record passes without being read into storage. The FSF (or BSF) function permits you to skip to the end of the file (identified by a tapemark).
2. The functions of FSR, FSF, BSR, and BSF always start at an interrecord gap.
3. If blocked input records are processed and if you do not want to process the remaining records in the block, nor one or more succeeding blocks, issue a RELSE macro before the CNTRL macro. The next GET then makes the first record of the new block available for processing. If the CNTRL macro (with FSR, for example) is issued without a preceding RELSE, the tape is advanced. The next GET makes the next record in the old block available for processing.
4. For any I/O area combination except one I/O area and no work area, IOCS always reads one tape block ahead of the one that is being processed. Thus, the next block after the current one is in storage ready for processing. Therefore, if a CNTRL FSR is given, the second block beyond the present one is passed without being read into storage.

5. If FSR or BSR is used, LIOCS does not update the block count. Furthermore, IOCS cannot sense tapemarks on an FSR or BSR command. Therefore, IOCS does not perform the usual EOV or EOF functions in these cases.

The tape spacing functions (FSL or BSL) apply to spanned record input files only. These codes are used when logical record spacing is desired. Consider these factors when FSL or BSL is specified:

1. Logical record spacing is ignored if it immediately follows a RELSE macro.
2. Forward and backward spacing refer to the absolute direction of the spacing. For example, if BSL is specified on an input file with READ=BACK, only one logical record is skipped.
3. If an end-of-file, end-of-volume, or an error condition occurs while a FSL or BSL spacing function is being executed, the condition is handled as if it occurred during a normal GET operation.

Chapter 7: Processing Unit Record Files

Unit record files are, in general, characterized by utilizing a wide variety of storage media. These range from punched card and paper tape through printer and console to the latest in magnetic ink (MICR) and machine readable printed (OCR) media. For some of these, each record is complete on one *unit* of information storage such as a punched card or MICR-inscribed check. For other files, such as printer or console file, a unit is the line of print or display characters, rather than a physical entity like a piece of paper. For yet others, as with paper tape or OCR journal tape, the nature of a unit is not so well defined.

The result of this variety is that unit record programming is highly device-dependent, to the degree that different DTFs are needed to define files for different types of unit record I/O devices. This chapter discusses the following unit record files, based on device types:

- Punched Card Files
- Printer Files
- Console Files
- Magnetic Ink Character Reader Files
- Optical Reader Files
- Paper Tape Files

Processing Punched Card Files

Before punched card files can be processed, they must be defined by the DTFCD macro and the CDMOD logic module generation macro. DTFCD and CDMOD are also required to define 3881 Optical Mark Reader files. See the section "Processing Optical Reader Files" for more information on processing 3881 files. The DTFCD operands are listed in Figure 7-1. For details of the DTFCD and CDMOD macros, see *VSE/Advanced Functions Macro Reference.*

The range of punched card equipment provided by IBM allows the user to select devices that best support his applications. Some of these devices perform only one function, for example reading or punching. Other types are able to perform different functions in separate card paths, while yet others can perform different functions in a single card path.

The first part of this section ("Associated Files") provides hints to bear in mind when using the IBM 2560, 3525, or 5424/5425 to perform multiple functions on a file in one pass.

The IBM 3504 and 3505 offer support for a different kind of application: these card readers can be equipped with the Optical Mark Reader special feature, which allows reading of up to 40 columns of marked data. Hints for dealing with this OMR data are given under "OMR Data".

When only one function is to be performed on a card file the DTFCD declarative macro is used for the files to be read or punched, and the DTFPR macro for files to be printed. The FUNC operand specifies the function to be performed on the file, namely:

FUNC=R	for reading,
FUNC=P	for punching,
FUNC=W	for printing,
FUNC=I	for punching/interpreting.

When more than one function is performed on a file, one speaks of *associated files.* All combinations of the three functions read, punch, and print are possible.

Associated Files

In VSE assembler language, a file definition must be given for each function to be performed on the cards. (Punch-interpret is an exception.) These definitions are established by DTFCD macros for read and punch files, and by the DTFPR macro for print files. The files are associated by means of their respective FUNC and ASOCFLE operands. The FUNC operand specifies the function combination to be performed on the card file. Valid parameters are:

FUNC=RP	for read-punch,
FUNC=RW	for read-print,
FUNC=RPW	for read-punch-print,
FUNC=PW	for punch-print.

The ASOCFLE operand in each DTF identifies one associated read, punch, or print file by its filename (ASOCFLE=*filename).* Figure 7-2 shows how, for the different function combinations, the ASOCFLE operands must be specified in the two or three DTF macros. For example, if FUNC=PW is specified, specify the filename of the punch DTFCD in the ASOCFLE operand of the print DTFPR.

Associated files can have only one I/O area each. The I/O area with or without workarea may be different for each associated file, or it may be the same. When, for example, you specify the same I/O area (or workarea) for an RW file, the same information will be read and printed. Or, if you use the same I/O area (or workarea) for the print and punch files of an RPW card file, the information that is printed will be the same as the information punched.

Applies to					
Input	Output	Combined			
X	X	X	M	DEVADDR=SYSxxx	Symbolic unit for reader-punch used for this file.
X	X	X	M	IOAREA1=xxxxxxxx	Name of first I/O area, or separate input area if TYPEFLE=CMBND and IOAREA2 are specified.
X	X		O	ASOCFLE=xxxxxxx	Name for FUNC=RP, RW, RPW, PW.
X	X	X	O	BLKSIZE=nnn	Length of one I/O area, in bytes. If omitted, 160 is assumed for a column binary on the 2560, 3504, 3505, or 3525; 96 is assumed for the 2596 or 5424/5425, otherwise 80 is assumed.
X	X	X	O	CONTROL=YES	CNTRL macro used for this file. Omit CTLCHR for this file. Does not apply to 2501.
	X		O	CRDERR=RETRY	RETRY if punching error is detected. Applies to 2520 and 2540 only.
	X		O	CTLCHR=xxx	(YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit if TYPEFLE=CMBND. Omit CONTROL for this file.
X	X	X	O	DEVICE=nnnn	(1442, 2501, 2520, 2540, 2560P, 2560S, 2596, 3504, 3505, or 3525. Specify 5425P or 5425S for 5424/5425 (P or S). If omitted, 2540 is assumed.
X		X	O	EOFADDR=xxxxxxxx	Name of your end-of-file routine.
X	X		O	ERROPT=xxxxxx	IGNORE, SKIP, or name. Applies to 2560, 3504, 3505, 3525 and 5424/5425 only.
X	X		O	FUNC=xxx	R, P, I, RP, RW, RPW, PW. Applies to 2560, 3525, and 5424/5425 only.
X	X	X	O	IOAREA2=xxxxxxx	Name of second I/O area, or separate output area if TYPEFLE=CMBND. Not allowed if FUNC=RP, RW, RPW, or PW. Not allowed for output file if ERROPT=IGNORE.
X	X		O	IOREG=(nn)	Register number, if two I/O areas used and GET or PUT does not specify a work area. Omit WORKA.
X	X		O	MODE=xx	(E or C) for 2560. (E, C, O, R, EO, ER, CO, CR) for 3504 and 3505. (E, C, R, ER, CR) for 3525. If omitted E is assumed.
X	X	X	O	MODNAME=xxxxxxxx	Name of CDMOD logic module for this DTF. If omitted, IOCS generates standard name.
		X	O	OUBLKSZ=nn	Length of IOAREA2 if TYPEFLE=CMBND. If OUBLKSZ omitted, length specified by BLKSIZE is assumed for IOAREA2.
X	X	X	O	RDONLY=YES	Generates a read-only module. Requires a module save area for each task using the module.
X	X	X	O	RECFORM=xxxxxx	(FIXUNB, UNDEF, or VARUNB). If omitted, FIXUNB is assumed. Input or combined files always FIXUNB.
	X		O	RECSIZE=(nn)	Register number if RECFORM=UNDEF. General registers 2-12, written in parentheses.
X	X	X	O	SEPASMB=YES	DTFCD is to be assembled separately.
X	X		O	SSELECT=n	(1 or 2) for 1442, 2520, 2596, 3504, or 3525. (1, 2, or 3) for 2540. (1, 2, 3, 4, or 5) for 2560. (1, 2, 3, or 4) for 5424/5425. Stacker-select character.
X	X	X	O	TYPEFLE=xxxxxx	(INPUT, OUTPUT, or CMBND). If omitted INPUT assumed. CMBND may be specified for 1442N1, 2520B1, or 2540 punch-feed-read only.
X	X	X	O	WORKA=YES	GET or PUT specifies work area. Omit IOREG. Not allowed for output file if ERROPT=IGNORE.

M = Mandatory
O = Optional

Figure 7-1. DTFCD macro operands.

Code in FUNC=operand	filename specification in ASOCFLE=operand of		
	read DTFCD	punch DTFCD	print DTFPR
FUNC=PW		filename of print DTFPR	filename of punch DTFCD
FUNC=RP	filename of punch DTFCD	filename of read DTFCD	
FUNC=RPW	filename of punch DTFCD	filename of print DTFPR	filename of read DTFCD
FUNC=RW	filename of print DTFPR		filename of read DTFCD

Examples:

1. If FUNC=PW is specified:
   a. specify the filename of the print DTFPR in the ASOCFLE operand of the punch DTFCD and
   b. specify the filename of the punch DTFCD in the ASOCFLE operand of the print DTFPR.
2. If FUNC=RPW is specified:
   a. specify the filename of the punch DTFCD in the ASOCFLE operand of the read DTFCD, and
   b. specify the filename of the print DTFPR in the ASOCFLE operand of the punch DTFCD, and
   c. specify the filename of the read DTFCD in the ASOCFLE operand of the print DTFPR.

Figure 7-2. ASOCFLE operand usage with print associated files

Processing Considerations for Associated Files

VSE does not provide any special macros to control the overlapping of reading with processing. For the IBM 2560 and the 5424/5425, however, the assembler language programmer who uses LIOCS can achieve improved performance through the use of dummy PUTs as described in the following text. The assembler language programmer who uses PIOCS can design his own overlapped processing.

Note: When using associated files and ASSGN ...,IGN. all logical units of the associated files must be ASSGN-ed IGN.

Read-Punch Associated Files: For RP associated files, the GET for the readfile and the PUT for the punch file must both be issued for each card. If punching is not desired, the output area or the workarea must be filled with blanks. LIOCS tests for blanks in the output area or workarea and, if it finds them, suppresses the punching. When the operand CTLCHR=YES or ASA is specified in the DTFCD macro for the punch file, the control character must always be present in the first byte of the output area or workarea; only the data portion following the control character may be filled with blanks. If the CNTRL macro is used, it must be issued before the PUT. As a result of the PUT, LIOCS will initiate the reading of the next card, and read it into a special buffer, which is part of the DTF table for the read file. The user need not, and cannot, set up this buffer, nor control its use. The next GET will obtain the data from this buffer and move it into the input area. Thus, by issuing the PUT as soon as possible after the GET, as much as possible of the next card will be read while the program is doing other processing.

Read-Print Associated Files: For RW associated files, the GET for the read file must be issued for each card. The PUT for the print file needs to be issued only when actual printing is desired. But since the PUT initiates the reading of the next card, it is advisable to issue a PUT even if no printing is desired and to fill the output area or work area with blanks (as described for the RP files above). If no PUT is issued, overlapped processing cannot take place.

Read-Punch-Print Associated Files: For RPW associated files, the GET for the read file and the PUT for the punch file must both be issued for each card. The PUT for the print file, however, needs to be issued only when actual printing is desired. Since it is this PUT that initiates the reading of the next card, it is advisable to issue the PUT for the print file even if no printing is desired and to fill the output area or workarea with blanks (as described for the RP files above). If no PUT for the print file is issued, overlapped processing cannot take place. When the operand CTLCHR=YES or ASA is specified in the DTFCD macro for the punch file, the control character must always be present in the first byte of the output area or workarea; only the data portion following the control character may be filled with blanks. If the CNTRL macro is used, it must be issued before the PUT for the punch file.

Associated files are discussed further in *VSE System Data Management Concepts.*

OMR Data

Format Descriptor Card: When you process an input file on the IBM 3504 or the 3505, you can specify MODE=O in the DTFCD macro to indicate optical mark read mode and MODE=R to indicate read column eliminate mode.

The specification of either O or R requires a format descriptor card defining the columns to be read or eliminated. This descriptor card must be the first card in the file. When it is found, an 80-byte record

is built which relates to the specified format on a column-per-column basis. If the format descriptor record is not found, a message is issued to the operator and the job is terminated. The format descriptor card is written as follows:

FORMAT (n1,n2)[,(n3,n4)...]

FORMAT must be punched in columns 2-7, followed by a blank in column 8. Operands begin in column 9 and may continue through column 71; they must be separated by a comma. Continuation cards can be specified by punching an X in column 72; coding on the next card must then begin in column 16. Both n1 and n2 must be greater than or equal to 1, and less than or equal to 80. The operand n2 must be greater than or equal to n1. If the format descriptor card is written

FORMAT (n1,n2),(n3,n4),...

n2 must be less than n3. For OMR, n3 minus n2 must be greater than or equal to 2.

For MODE=O, n1 indicates the first column, and n2 indicates the last column to be read in OMR mode. Only every other column between n1 and n2 can be read in OMR mode; therefore, n1 and n2 must both have even values or both have odd values.

For MODE=R, n1 indicates the first column *not* to be read, and n2 indicates the last column *not* to be read.

For example, if the operand MODE=O is specified and you want to read columns 1, 3, 5, 7, 9, 70, 72, 74, 76, 78, and 80 in OMR mode, you would use the following format descriptor card:

FORMAT (1,9),(70,80)

Or, if the operand MODE=R is specified and you want to read all card columns except 20 through 30 and 52 through 76, you would use the following format descriptor card:

FORMAT (20,30),(52,76)

Data Card: The following rules apply to the coding of an input card to be read in OMR mode:

- Mark characters (character to be read optically) must be separated by at least one column that contains neither marks nor punches. "M" in the example indicates mark characters and "ƀ" indicates the blanks:

  MƀMƀMƀƀM

- Mark characters must be separated from any columns containing punched holes (in the example indicated by "H") by at least one column that contains neither marks nor punches:

  MƀHƀHHH

- Mark characters in odd columns must be separated from mark characters in even columns by at least two columns that contain neither marks nor punches:

  MƀMƀƀMƀM

OMR Data Record: Although OMR data is physically located in alternating columns, the data in the I/O area is compressed into contiguous bytes. The relationship of the data on card columns to the location of the data in storage is as follows:

1. If column n does not contain OMR data, the data content of column n+1 represents the contiguous byte in virtual storage which follows the column n data byte.
2. If column n does contain OMR data, the data content of column n+2 represents the contiguous byte in virtual storage which follows the column n data byte. The data contents of column n+1 is not placed in virtual storage.
3. The data content of column 1 always represents the first data byte in virtual storage.

Figure 7-3 shows how these rules apply to the data card and its format descriptor card, and the record which results from reading the data card.

When a weak mark or poor erasure is detected in a column, the column's data is replaced with a hexadecimal 3F (X'3F') when reading in EBCDIC mode, or two hexadecimal 3Fs (X'3F3F') when reading in column binary mode. Checking for this condition is the user's responsibility.

If X'3F' is placed in the data, an X'3F' is also placed in byte 80 of the I/O area when reading in EBCDIC mode, or in byte 160 when reading in column binary mode, to indicate an OMR reading error. You can then determine whether or not an OMR reading error occurred on the card by checking this byte. If, however, the I/O area length is less than 80 for EBCDIC mode or less than 160 for column binary mode, the X'3F' is not placed in virtual storage. In this case, to determine if a reading error occurred, you must check each OMR byte for an X'3F'.

Updating Records

A card record may, with some devices, be read and then have additional information punched back into the same card. This is possible with the 2560, 3525, and 5424/5425, and with the 1442, 2520, and 2540 equipped with the special punch-feed-read feature.

Card column	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Card Data	P_1	P_2	ƀ	M_4	ƀ	M_6	ƀ	ƀ	M_9	ƀ	M_{11}	ƀ	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}	P_{19}	P_{20}
Format Data	ƀ	ƀ	ƀ	F_4	–	F_6	–	ƀ	F_9	–	F_{11}	–	ƀ	ƀ	ƀ	ƀ	ƀ	ƀ	ƀ	ƀ
	Switch from punch to mark					Switch from even to odd marks					Switch from mark to punch									
Format Descriptor Card		F	O	R	M	A	T		(	4	.	6	)	,	(	9	,	1	1	)
Channel Data	P_1	P_2	ƀ	M_4	M_6	ƀ	M_9	M_{11}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}	P_{19}	P_{20}				

ƀ = Must have neither hole nor mark data
ƀ = Hexadecimal 40
– = May be character or blank
P_x = Punched data in column x
M_x = Mark data in column x
F_x = Format data for column x

Figure 7-3. OMR coding example.

For the card devices, there are two ways of specifying in the DTFCD that such updating is desired; which way is used depends on device type.

- For the 1442, 2520, or 2540 equipped with the punch-feed-read feature, use a combined file by specifying TYPEFLE=CMBND in the DTFCD. An example of a combined card file is given below. For the 2540 with the punch-feed-read feature, the file to be updated must be in the punch feed.
- For the 2560, 3525, or 5424/5425, use associated files. Associated files are defined in the associated file declarations (DTFCD and DTFPR) by the ASOCFLE and FUNC operands.

When updating a file, one I/O area can be specified (using the IOAREA1 operand) for both the input and output of a card record. If a second I/O area is required, it can be specified with the IOAREA2 operand. For associated DTFCD files, however, two I/O areas are not allowed.

A PUT for a combined card file must always be followed by a GET before another PUT is issued: GETs, however, can be issued as many times in succession as desired. The corresponding rules for an associated card file are given in the section "GET/CNTRL/PUT Sequence for Associated Files," below.

For a file using the 2540 with the punch-feed-read special feature, a PUT macro must be issued for each card. For a 1442 or 2520 file, however, a PUT macro may be omitted if a particular card does not require punching. The operator must run out the 2540 punch following a punch-feed-read job.

In the combined card file example of Figure 3-2 data is punched into the same card which was read. Information from each card is read, processed, and then punched into the same card to produce an updated record.

End-of-File Handling

The EOFADDR operand must be included for input and combined files and specifies the symbolic name of your end-of-file routine. IOCS automatically branches to this routine on an end-of-file condition. In your routine you can perform any operations required for the end of the file (you generally issue a CLOSE instruction for the file).

IOCS detects end-of-file conditions in the card reader by recoginizing the characters /* punched in card columns 1 and 2 (column 3 must be blank). If the system logical units SYSIPT and SYSRDR are assigned to a 5424/5425, IOCS requires that the /* card, indicating end-of-file, be followed by a blank card. An error condition results if cards are allowed

to run out without a /* trailer card (and without a /& card if end-of-job).

Error Handling

The ERROPT operand specifies the error option used for an input or output file on a 2560, 3504, 3505, 3525, or 5424/5425. Either IGNORE, SKIP, or the symbolic name of an error routine can be specified for output files. This operand must be omitted when using 2560 or 5424/5425 associated output files. The functions of these parameters are described below.

IGNORE indicates that the error is to be ignored. The address of the record in error is put in register 1 and made available for processing. For output files, byte 3, bit 3 of the CCB is also set on (see Figure 9-3); you can check this bit and take the appropriate action to recover from the error. Only one I/O area and no work area is permitted for output files. When IGNORE is specified for an input file associated with a punch file (FUNC=RP or RPW) and an error occurs, a PUT for the card in error must nevertheless be given for the punch file.

SKIP indicates that the record in error is not to be made available for processing. The next card is read and processing continues.

If name is specified, IOCS branches to your routine when an error occurs, where you may perform whatever actions you desire. Register 1 contains the address of the record in error, and register 14 contains the return address. GET macros must not be issued in the error routine for cards in the same device (or in the same card path for the 2560 or 5424/5425). If the file is an associated file, PUT macros must not be issued in the error routine for cards in the same device (for the 2560 or 5424/5425 this applies to cards in either card path). If any other IOCS macros are issued in the routine, register 14 must be saved. If the operand RDONLY=YES is specified, register 13 must also be saved. At the end of your routine return to IOCS by branching to the address in register 14. If the input file is associated with an output file (FUNC=RP, RPW or RW), no punching or printing must be done for the card in error. IOCS continues processing by reading the next card.

Note: When ERROPT is specified for an input file and an error occurs, there is danger that the /* end-of-file card may be lost. This is because IOCS, after taking the action for the card in error specified by the ERROPT operand, returns to normal processing by reading the next card which is assumed to be a data card. If this card is in fact an end-of-file card, the end-of-file condition cannot be recognized.

If an ERROPT routine issues I/O macros using the same read-only module that causes control to pass to the error routine your program must provide another save area. One save area is used for the normal I/O operations, and the second for I/O operations in the ERROPT routine. Before returning to the module that entered the ERROPT routine, register 13 must contain the save area address originally specified for the task.

Programming Considerations

If OMR or RCE is specified for a 3505 card reader or if RCE is specified for a 3525 card punch, OPEN retrieves the data from the first data card and analyzes this data to verify the presence of a format descriptor card. If a format descriptor card is found, OPEN builds an 80-byte record corresponding to the format descriptor card. If a format descriptor card is not found, a message is issued and the job is canceled.

For a 2560, 3525, or 5424/5425 print only file, OPEN will feed the first card to ensure that a card is at the print station.

For 2560, 3525, or 5424/5425 associated files, all of the associated files must be opened before a GET or PUT is used for any of the files.

When a 2540 is used for a card input file, each GET macro normally reads the record from a card in the read feed. However, if the 2540 has the special punch-feed-read feature installed and if TYPEFLE=CMBND is specified in the DTFCD macro, each GET reads the record from a card in the punch feed, at the punch-feed-read station. This record can be updated with additional information that is then punched back into the same card when the card passes the punch station and a PUT macro is issued.

2560 Printing

The 2560 has a maximum of 6 print heads, one for each print line. For a description of how the print heads may be set, see the appropriate IBM 2560 Multi-Function Card Machine manuals. The output area can be as large as 384 bytes, the equivalence of 64 characters per line.

With one PUT macro, one logical line of up to 384 characters in length is printed. This logical line is split up into 6 physical lines. Thus a single PUT macro prints all the information for a card. The next PUT macro will cause printing for the next card.

3525 Printing

For a 3525 with a 2-line printer, output is automatically printed on lines 1 and 3.

When automatic line positioning is used for a print-only file on a 2-line printer, the one PUT macro

causes line 3 to be printed and the other PUT causes a new card to be fed and the printing of line 1 to be started.

With a multiline printer, card feeding is caused by the PUT macro which follows the PUT causing printing on line 25. This PUT macro also starts the printing on line 1 of the next card. If you want to control line positioning, either the CONTROL operand or the CTLCHR operand must be specified in the DTFPR macro. (CONTROL and CTLCHR are not valid for card feeding when they are specified for a printer file associated with a read or punch file.) You are then responsible for all spacing and skipping during printing. If CTLCHR=YES is specified, you are also responsible for card feeding. The following restrictions apply to user-controlled line positioning:

1. Any attempt to print on lines other than lines 1 or 3 on a 2-line printer results in a command reject. Otherwise, 2-line printer support is identical to multiline printer support.
2. A space after printing command for line 25 results in positioning on line 1 of the next card.
3. Any attempt to print and suppress spacing results in a command reject.
4. Any skip command to a channel number less than or equal to the present channel position results in line positioning at that channel positioning on the next card.
5. If CONTROL or CTLCHR is specified, FUNC is ignored for 2-line printer support.

5424/5425 Printing

The 5424/5425 has a maximum of 4 print heads, one for each print line. The output area can be as large as 128 bytes, the equivalent of 32 characters per line.

With one PUT macro, one logical line of up to 128 characters in length is printed. This logical line is split up into 4 physical lines. Thus a single PUT macro prints all the information for a card. The next PUT macro will cause printing for the next card.

Closing a File

For the 2560, 3525, or 5424/5425, when CLOSE is issued for a file, it must also be issued for any associated files without any intervening input/output operations. Reopening one associated file requires reopening the others.

For 2560 or 5424/5425 read associated files, the last card must not be punched or printed. When a read file (single or associated) is closed, the last card read will be selected into the output stacker when 2560 "unit exception" has occurred - that is, when there is no following card. Two extra feed cycles are executed to perform this. When a punch or print file (without an associated read file) is closed, LIOCS performs one feed cycle to select the last card into the output stacker. When an associated punch-print file is closed, LIOCS performs one feed cycle to select the last card into the output stacker; if a print PUT was not specified for the last card, LIOCS executes the punch PUT before performing one feed cycle to select the card into the output stacker.

When O or R has been included in the DTFCD MODE operand for a 3504, 3505, or 3525 running batched jobs, a non-data card must follow the card which causes your program to close the file.

For the 3525, Figure 7-4 shows the card movement caused by issuing CLOSE.

File Type	Feed Caused by CLOSE for:
Read	Read*
Punch	Punch
Print	Print
Read/Print	Print*
Read/Punch/Print	Print**
Read/Punch	Punch**
Punch/Print	Print
Punch/Interpret	Punch

* A card feed is executed only if R has been specified in the DTFCD MODE operand. Programs using read-column-eliminate mode must detect an end-of-file condition themselves.

** Delimiter cards cannot be punched or printed in these files. CLOSE always issues a feed command.

Figure 7-4. CLOSE card movement for the 3525.

Card Device and Printer Control

Output stacker selection for a card device, and line spacing or skipping for a printer, can be controlled either by specified control characters in the data records or by the CNTRL macro. Either method, but not both, may be used for a particular file.

The CNTRL macro cannot be used for an input file with two I/O areas (when the IOAREA2 operand is specified).

The CNTRL macro must not be used for printer or punch files if the data records contain control characters and the entry CTLCHR is included in the file definition.

Whenever CNTRL is issued in your program, the DTF CONTROL operand must be included (except for DTFMT and DTFDR) and CTLCHR must be omitted. If control characters are used when CONTROL is specified, the control characters are ignored and treated as data.

The CONTROL operand must not be specified for an input file used in association with a punch file

(when the operand FUNC=RP or RPW is specified) on the 2560, 3525, or 5424/5425; in this case, however, this operand can be specified in the DTFCD for the associated punch file.

CNTRL usually requires two or three parameters. The first parameter must be the name of the file specified in the DTF header entry. It can be specified as a symbol or in register notation.

The second parameter is the mnemonic code for the command to be performed. This must be one of a set of predetermined codes (see Figure 7-5).

When control characters in data records are used, the DTF CTLCHR operand must be specified, and every record must contain a control character in the virtual-storage output area. This control character must be the first character of each fixed-length or undefined record, or the first character following the record-length field in a variable-length record. The BLKSIZE specification for the output area must include the byte for the control character. If undefined records are specified, the RECSIZE specification must also include this byte.

When a PUT macro is executed, the control character in the data record determines the command code (byte) of the CCW that IOCS establishes. The control character is used as follows:

If CTLCHR=ASA, the control character is translated into the command code.

If CTLCHR=YES, the control character is used directly as the command code.

If a program using ASA control characters sends a space and/or skip command (without printing) to the printer, the output area must contain the first character forms control, and the remainder of the area must be blanks (X'40').

If a program using ASA control characters prints on the 3525, you must use a space 1 control character (a blank) to print on the first line of a card. The particular character to be included in the record depends on the function to be performed. For example, if double spacing is to occur after a particular line is printed, the code for the double spacing must be the control character in the output line to be printed. The first character after the control character in the output data becomes the first character punched or printed. Appendix A gives a complete list of control characters.

1442 and 2520 Card Read Punch Codes

Cards fed in the 1442 and 2520 are normally directed to the stacker specified in the DTF SSELECT operand. If SSELECT is omitted, they go to stacker 1. The CNTRL macro can be used to temporarily override the normally selected stacker.

Input File: CNTRL can be used only when one I/O area, with or without a work area, is specified for the file. To stack a particular card, the CNTRL macro should be issued after the GET for that card, and before the GET macro for the following card. When the next card is read, the previous card is stacked in the specified stacker.

Note: If CNTRL is not issued after each GET, the same card remains at the read station.

Output File: CNTRL can be used with any permissible combination of I/O areas and work areas. To stack a particular card, the CNTRL macro should be issued before the PUT for that card. After the card is

IBM Unit	Mnemonic Code	n_1	Command
1442, 2520 Card Read Punch	SS	1 2	Select stacker 1 or 2
	E		Eject to stacker 1 (1442 only)
2540 Card Read Punch 3504, 3505 Card Readers 3525 Card Punch	PS	1 2 3	Select stacker 1, 2, or 3 (For 3504, 3505, and 3525, 3 defaults to stacker 2)
2560 Multi-Function Card Machine	SS	1 2 3 4 5	Select stacker 1, 2, 3, 4, or 5.
2596 Card Read Punch	SS	1 2	Select stacker 1 for Read, or stacker 3 for Punch. Select stacker 2 for Read, or stacker 4 for Punch.
5424/5425 Multi-Function Card Unit	SS	1 2 3 4	Select stacker 1, 2, 3, or 4.

Figure 7-5. CNTRL Codes.

punched, it is stacked immediately into the specified pocket.

Combined File: CNTRL can be used with any permissible combination of I/O areas and work areas. If a particular card is to be selected, the CNTRL macro for the file should be issued after the GET and before the PUT for the card. When the next card is read, the previous card is stacked into the specified stacker.

2540 Card Read Punch Codes

Cards read or punched on the 2540 normally fall into the stacker specified in the DTF SSELECT operand (or the R1 or P1 stacker if SSLECT is omitted). The CNTRL macro with code PS is used to select a card into a different stacker, which is specified by the third operand, n1. The possible selections are shown below. (These selections are also those which may be specified in the DTF SSELECT operand.)

Feed	Stacker	Value of n1
Read	R1	1
Read	R2	2
Read	RP3	3
Punch	P1	1
Punch	P2	2
Punch	RP3	3

Input File: CNTRL can be used only when one I/O area, with or without a work area, is specified for the file. To stack a particular card, the CNTRL macro should be issued after the GET for that card. Before the next GET macro is executed, the card is stacked into the specified stacker.

Note: If your program indicates that operator intervention is required on a 2540 (for example, to correct a card out of sequence in a card deck), and your program has specified CONTROL=YES in CDMOD, and you do not use the CNTRL macro, then you should issue a CNTRL macro before the operator intervention is requested. Issuing CNTRL in this situation assures that subsequent commands issued to the 2540 after the operator intervention are not rejected as invalid.

Output File: CNTRL can be used with any permissible combination of I/O and work areas. When you want to select a particular card, CNTRL must be issued before the PUT for that card. However, CNTRL does not have to precede every PUT.

2560 and 5424/5425 Card Device Codes

Cards fed into the 2560 or 5424/5425 are normally directed to the output stacker specified in the DTF SSELECT operand. If SSELECT is omitted, cards which come from hopper 1 go to output stacker 1; and cards which come from hopper 2 go to output stacker 5 for the 2560, or to output stacker 4 for the 5424/5425. The CNTRL macro can be used to temporarily override the stacker selection specified in the SSELECT operand or by default.

Single File: CNTRL cannot be used for a print file. For a read file, to stack a particular card the CNTRL macro must be issued after the GET for that card. For a punch file, or a punch/interpret file (DTFCD FUNC=I), to stack a particular card, CNTRL must be issued before the PUT for that card.

Associated File: The sequence of CNTRL macro usage with associated files is described below and is summarized in the section "GET/CNTRL/PUT Sequence for Associated Files". CNTRL must be used with only one of the associated files:

- With the read file if the associated file is read/print. In this case, to stack a particular card, CNTRL must be issued after the GET and before any PUT for that card. If no PUT is issued for that card, then CNTRL must be issued after the GET for that card and before the GET for the next card.
- With the punch file if the associated file is anything other than read/print. In this case, to stack a card, CNTRL must be issued before the PUT which punches that card.

2596 Card Read Punch Codes

Cards fed into the 2596 are normally directed to the stacker specified in the DTF SSELECT operand. If SSELECT is omitted, cards go to stacker 1 for read and stacker 3 for punch. The CNTRL macro can be used to temporarily override the normally selected stacker. The possible selections are shown in Figure 7-5. (These selections are also those which may be specified in the DTF SSELECT operand.)

Input File: CNTRL can be used only when one I/O area, with or without a work area, is specified for the file. To stack a particular card, the CNTRL macro should be issued after the GET for that card, and before the GET for the next card. When the next card is read, the previous card is stacked in the specified stacker.

Output File: CNTRL can be used with any permissible combination of I/O areas and work area. To stack a particular card, the CNTRL macro should be issued before the PUT for that card. After the card is punched it is stacked immediately into the specified stacker.

3504 and 3505 Card Readers and 3525 Card Punch Codes

Cards read on the 3504 or 3505 or punched on the 3525 are normally directed to the stacker specified in the DTF SSELECT operand. If SSELECT is omitted and if no other CNTRL issuance in the program se-

lects stacker 2 or 3, stacker 1 is assumed. If a CNTRL macro is issued elsewhere in the program, selecting stacker 2 or 3, then stacker 1 must be explicitly selected. The CNTRL macro overrides the stacker selection specified in the SSELECT operand or by default. For input files, CNTRL can be used only when one I/O area is specified for the file.

3525 Card Printing Codes

The CNTRL macro can control spacing and skipping to a specific line on a card for the 3525 card print feature. The command code SP is used to direct the 3525 to space one, two, or three lines on a card; SK is used to skip to a channel (1 through 12) on a card.

The 3525 print channels correspond to specific rows on a printed card. The channels and their corresponding card rows are shown below:

```
Row Number    Corresponding Channel
   1 --------------------1
   2
   3 --------------------2
   4
   5 --------------------3
   6
   7 --------------------4
   8
   9 --------------------5
  10
  11 --------------------6
  12
  13 --------------------7
  14
  15 --------------------8
  16
  17 --------------------9 (overflow)
  18
  19 -------------------10
  20
  21 -------------------11
  22
  23 -------------------12 (overflow)
  24
  25
```

GET/CNTRL/PUT Sequence for Associated Files

For 2560, 3525, or 5424/5425 associated files, GET, CNTRL, and PUT macros must be used with the files in the sequence given in Figure 7-6. For example, to process a card of a read-punch associated file requires first issuing a GET macro for the file defined in the read DTFCD, and then issuing a CNTRL macro (if desired) for the file declared in the punch DTFCD, and then issuing a PUT macro for the file declared in the punch DTFCD.

GET/PUT sequences other than those given in Figure 7-6 will cause an abnormal termination with an illegal supervisor call 32 message. The use of CNTRL in sequences other than those shown in Figure 7-6 will cause unpredictable results.

Improved performance for the 2560 or 5424/5425 may be achieved by a type of overlapped processing through the use of dummy PUTs.

Processing Printer Files

Before it can be processed, a printer file must be defined with the DTFPR and PRMOD macros. Figure 7-7 lists the operands for these macros. (Note that not all operands are valid for PRMOD). For details, see *VSE/Advanced Functions Macro Reference.*

Associated Files

The ASOCFLE operand is used together with the FUNC operand to define associated files for the 2560, 3525, or 5424/5425. (For a description of associated files, see the section in "Processing Punched Card Files," preceding.)

ASOCFLE specifies the filename of an associated read and/or punch file, and enables macro sequence checking by the logic module of each associated file. One filename is required per DTF for associated files.

Figure 7-2 defines the filename specified by the ASOCFLE operand for each of the associated DTFs.

FUNC={W[T]|RW[T]|RPW[T]|PW[T]}

This operand specifies the type of file to be processed by the 2560, 3525, or 5424/5425. W indicates print, R indicated read, P indicates punch, and T (for the 3525 only) indicates an optional 2-line printer.

RW[T], RPW[T], and PW[T] are used, together with the ASOCFLE operand, to specify associated files; when one of these parameters, other than T, is specified for a printer file it must also be specified for the associated file(s). Note: Do not use T for associated files; it is valid only for printer files.

If a 2-line printer is not specified for the 3525, multi-line print is assumed. T is ignored if CONTROL or CTLCHR is specified.

Printer Overflow

The PRTOV (printer overflow) macro is used with a printer file to specify the operation to be performed when a carriage overflow condition occurs. To use this macro, the PRINTOV=YES operand must be included in the DTFPR.

The PRTOV macro causes a skip to channel 1, or branches to your routine, if an overflow condition (channel 9 or 12) is detected on the preceding space or print command. An overflow condition is not recognized during a carriage skip operation. After the execution of any command that causes carriage movement (PUT or immediate CNTRL), you should

To:	Issue:	For file declared in:	FUNC=
READ/PUNCH	GET	DTFCD (read file)	RP
	[CNTRL]*	DTFCD (punch file)	
	PUT	DTFCD (punch file)	
READ/PUNCH/PRINT	GET	DTFCD (read file)	RPW
	[CNTRL]*	DTFCD (punch file)	
	PUT	DTFCD (punch file)	
	[PUT]**	DTFPR	
READ/PRINT	GET	DTFCD	RW
	[CNTRL]*	DTFCD	
	[PUT]**	DTFPR	
PUNCH/PRINT	[CNTRL]*	DTFCD	PW
	PUT	DTFCD	
	[PUT]**	DTFPR	

* Optional. If used, however, the sequence is as shown.
** Optional provided you do not want to print on the card. If used, however, the sequence is as shown.

Figure 7-6. GET/CNTRL/PUT macro usage to process one card of an associated file.

issue a PRTOV macro before issuing the next CNTRL or PUT. This ensures that your overflow option is executed at the correct time.

On the 3525 card punch, a channel 9 test indicates print line 17. A channel 12 test indicates print line 23. An overflow condition from either of these channels causes:

- a transfer of control to the overflow routine specified in the PRTOV macro, or
- a skip to channel one to begin printing on the next card for print only files.

When the PRTOV macro is used on a 3525 2-line printer, the result of the test is always negative since lines 17 and 23 are not available. The test is logically a no-operation.

Note: PRTOV without the routine name option is invalid for 3525 associated files. A skip to channel one is valid only for 3525 print only files. PRTOV is not allowed for the 2560 or 5424/5425.

Printer Control

Line spacing or skipping for a printer can be controlled either by specified control characters in the data records or by the CNTRL macro. Either method, but not both, may be used for a particular file. For use of the latter method, see "Printer Codes," following.

When control characters in data records are used, the DTF CTLCHR operand must be specified, and every record must contain a control character in the virtual-storage output area. This control character must be the first character of each fixed-length or undefined record, or the first character following the record-length field in a variable-length record. The BLKSIZE specification for the output area must include the byte for the control character. If undefined records are specified, the RECSIZE specification must also include this byte. For maximum and default output area sizes for different printers, see Figure 7-8.

When a PUT macro is executed, the control character in the data record determines the command code (byte) of the CCW that IOCS establishes. The control character is used as follows:

If CTLCHR=ASA, the control character is translated into the command code.

If CTLCHR=YES, the control character is used directly as the command code.

If a program using ASA control characters sends a space and/or skip command (without printing) to the printer, the output area must contain the first-character forms control, and the remainder of the area must be blanks (X'40').

If a program using ASA control characters prints on the 3525, you must use a space 1 control character (a blank) to print on the first line of a card. The particular character to be included in the record depends on the function to be performed. For example, if double spacing is to occur after a particular line is printed, the code for double spacing must be the control character in the output line to be printed. The first character after the control character in the output data becomes the first character punched or printed. Appendix A gives a complete list of control characters.

PRMOD	DTFPR		
n/a	M	DEVADDR=SYSxxx	Symbolic unit for the printer used for this file.
n/a	M	IOAREA1=xxxxxxxx	Name for the first output area.
n/a	O	ASOCFLE=xxxxxxxx	Name of the associated file for FUNC=RW, RPW, PW.
n/a	O	BLKSIZE=nnn	Length of one output area, in bytes. If omitted, 121 is assumed for 1403, 1443, 3203 or PRT1; 136 is assumed for 3800 without TRC (or 137 with TRC); 64 is assumed for 2560 or 3525; 96 is assumed for 5203 or 5424/5425.[1]
	O	CONTROL=YES	CNTRL macro used for this file. Omit CTLCHR for this file. Not allowed for 2560 or 5424/5425.
O	O	CTLCHR=xxx	(YES or ASA). Data records have control character. YES for S/370 character set; ASA for American National Standards Institute character set. Omit CONTROL for this file. Not allowed for 2560 or 5424/5425.
O	O	DEVICE=xxxxx	1403, 1443, 2245, 2560P, 2560S, 3203, 3211, 3525, 3800, 5203. Specify 5425P or 5425S for 5424/5425 (P or S). Specify PRT1 for 3203-4, 3203-5, 3211, 3262, or 3289-4. If omitted, 1403 is assumed.[1]
O	O	ERROPT=xxxxxxxx	RETRY or the name of your error routine for PRT1. IGNORE for 3525. Not allowed for other devices.[1]
O	O	FUNC=xxxx	(W, RW, RPW, PW) for 2560 or 5424/5425. (W[T], RW[T], RPW[T], PW[T] for 3525.)
O	O	IOAREA2=xxxxxxxx	If two output areas are used, name of second area.
n/a	O	IOREG=(nn)	Register number, if two output areas used and PUT does not specify a work area. Omit WORKA.
n/a	O	MODNAME=xxxxxxxx	Name of PRMOD logic module for this DTF. If omitted, IOCS generates standard name. Not needed with 3800 advanced printer buffering.
O	O	PRINTOV=YES	PRTOV macro used for this file. Not allowed for 2560 or 5424/5425.
O	O	RDONLY=YES	Generates a read-only module. Requires a module save area for each task using the module.
O	O	RECFORM=xxxxx	(FIXUNB, VARUNB, or UNDEF). If omitted, FIXUNB is assumed.
O	O	RECSIZE=(nn)	Register number if RECFORM=UNDEF.
O	O	SEPASMB=YES	DTFPR is to be assembled separately.
O	O	STLIST=YES	1403 selective tape listing feature is to be used.
O	O	TRC=YES	For 3800, output data lines include table reference character.
n/a	O	UCS=xxx	(ON) process data checks. (OFF) ignores data checks. Only for printers with the UCS feature, PRT1, or 3800. If omitted, OFF is assumed.[1]
O	O	WORKA=YES	PUT specifies work area. Omit IOREG.

M = Mandatory
O = Optional
n/a = not allowed

[1] PRT1 refers to 3211-compatible printers (that is, with a device type code of PRT1).

Figure 7-7. DTFPR and PRMOD macro operands.

Printer Codes

The CNTRL macro can be used for forms control on any printer. CNTRL usually requires two or three parameters. The first parameter must be the name of the file specified in the DTF header entry. It can be specified as a symbol or in register notation.

The second parameter is the mnemonic code for the command to be performed. This must be one of a set of predetermined codes (see Figure 7-9).

The third parameter, n1, is required whenever a number is needed for stacker selection or immediate printer carriage control. The fourth parameter, n2, applies to delayed spacing or skipping. In the case of a printer file, the parameters n1 and n2 may be required.

The CNTRL macro must not be used for printer or punch files if the data records contain control characters and the CTLCHR operand is included in the file definition.

Whenever CNTRL is issued in your program, the DTF CONTROL operand must be included and CTLCHR must be omitted. If control characters are used when CONTROL is specified, the control characters are ignored and treated as data.

The codes for printer operation cause spacing (SP) over a specified number of lines, or skipping (SK) to a specified location on the form. The third parame-

Devices	Maximum length (in bytes) which can be specified[1]	Length assumed (in bytes)[2]
1403-1, -4	100	121
1403-6, -7	120	121
1403-2, -3, -5, -8, -9	132	121
1443	144	121
2560	384	64
3203	132	121
PRT1 (except 3211)	132	121
3211	150	121
3525	64	64
3800	204[3]	136[3]
5203	132	96
5424/5425	128	96

[1] RECFORM is FIXUNB or UNDEF and operand CTLCHR is not specified.

[2] The parameter BLKSIZE=n is omitted.

[3] Without TRC=YES. With TRC=YES, the maximum length is 205 and the assumed length is 137.

Notes:

- If CTRCHR=YES/ASA is specified, add 1 byte to the maximum length which can be specified.
- If RECFORM=VARUNB is specified add 4 bytes to the maximum value which can be specified.
- For the 2245, if RECFORM=VARUNB and CTLCHR=YES/ASA are specified, the maximum blocksize is 805 bytes.

Figure 7-8. Maximum and assumed lengths for the IOAREA1.

ter, n1, is required for immediate spacing and skipping (before printing). The fourth parameter, n2, is required for delayed spacing or skipping (after printing).

The SP and SK operations can be used in any sequence. However, two or more consecutive immediate skips (SK) to the same carriage channel on the same printer result in a single skip immediate. Likewise, two or more consecutive delayed spaces (SP) and/or skips (SK) to the same printer result in the last space or skip only. Any other combination of consecutive controls (SP and SK), such as immediate space followed by a delayed skip or immediate space followed by another immediate space, causes both specified operations to occur.

Printer With the UCS Feature: The CNTRL macro can be used before a PUT for a file to change the method of processing data checks. Data checks can be either processed with an indication given to the operator, or ignored with blanks printed in place of the unprintable characters.

A data check occurs when a character except null, (X'00'), or blank, (X'40') sent to the printer does not match any of the characters in the UCS buffer. On a 3800, a data check occurs when an attempt is made to merge a character with another character different from itself in the same print position, as well as when an unprintable character is transmitted.

Before opening a file, the BLOCK parameter of the UCS job control command determines for a 1403 whether data check processing takes place. For another UCS printer, the NOCHK option of the SYSBUFLD program (see *VSE/Advanced Functions System Control Statements)* has the same meaning. For a 3800, the DCHK parameter on the SETPRT job control statement (or SETPRT macro instruction) determines whether data checks are blocked or allowed.

If several DTFPRs are assigned to the same physical unit, the UCS parameter of the DTF last opened determines whether data check processing takes place. If a DTFDI is opened for a UCS printer, it has the effect of a NOCHK option. This change is operat-

IBM Unit	Mnemonic Code	n_1	n_2	Command
1403, 1443, 3203, PRT1, 3800, 5203 Printers 3525 Card Punch with Print feature[1]	SP	c	d	Carriage space 1, 2, or 3 lines
	SK	c	d	Skip to channel c and/or d (for 3525, a skip to channel 1 is valid only for print only files.)
1403, 5203 Printers with Universal Character Set feature or 3203, 3800, PRT1 Printers.[1]	UCS	ON OFF		Data checks are processed with an operator indication Data checks are ignored and blanks are printed.
PRT1 Printer[1]	FOLD			Print upper case characters for any byte with equivalent bits 2-7.
	UNFOLD			Print character equivalents of any EBCDIC byte.

c = An integer that indicates immediate printer control (before printing).
d = An integer that indicates a delayed printer control.

[1] PRT1 refers to 3211-compatible printers (that is, with a device type of PRT1).

Figure 7-9. CNTRL macro command codes.

ed on the physical device and is valid for all DTFs assigned to this device.

If the UCS form of the CNTRL macro is used for a printer (other than the 3800) without the UCS feature, the CNTRL macro is ignored.

FOLD and UNFOLD Codes: Except on a 1403, 3203, 3800, or 5203, the CNTRL macro can also be used before a PUT to control the printing of lower-case letters. Lower-case letters can either be printed or replaced by upper-case equivalents.

Prior to using a CNTRL macro, the printing of lower case letters is controlled by the UCB FOLD parameter of SYSBUFLD. If the FOLD parameter is specified, bits 0 and 1 are considered ones and the upper case equivalent of bits 2 to 7 is printed. If UNFOLD is specified, the character equivalent of the EBCDIC byte is printed.

When you issue a PUT for a printer file, this PUT causes the pertinent printer to space automatically by one line, provided the DTFPR macro for the file does not include the CTLCHR=code operand. Under these circumstances, there is no need to issue a CNTRL macro or to specify a control character in order or advance the paper on the printer by one line. If the DTFPR macro for the file does include CTLCHR=code, the appropriate control character must be moved to the first byte of the output area. For a list of control characters, see Appendix A.

STLSP=control field: This optional PUT operand specifies a control byte that allows for spacing while using the selective tape listing feature on the 1403 printer. To use this feature, the operand STLIST=YES must be specified in the DTFPR. Up to 8 paper tapes may be independently spaced. The control byte is set up like any other data byte in virtual storage. You can also use ordinary register notation to provide the address of the control byte. Registers 2 through 12 are available without restriction. You determine the spacing (which occurs after printing) by setting on the bits corresponding to the tapes you want to space. The correspondence between control byte bits and tapes is as follows:

Control byte bits	0	1	2	3	4	5	6	7
Tape position	8	7	6	5	4	3	2	1

The tape position 1 is the leftmost tape on the selective tape listing device.

Note: Double-width tapes must be controlled by both bits of the control field.

STLSK=control field: This optional PUT operand specifies a control byte that allows for skipping while using the selective tape listing feature on the 1403 printer. To use this feature, the operand STLIST=YES must be specified in the DTFPR. Up to 8 paper tapes may be independently skipped. The control byte is set up like any other data byte in virtual storage. You can also use ordinary register notation to provide the address of the control byte. Registers 2 through 12 are available without restriction. You determine the skipping (which occurs after printing) by setting on the bits corresponding to the tapes you want to skip. The correspondence between control byte bits and tapes is shown in the figure under "STLSP=control field", above.

Error Handling

The ERROPT operand specifies the action to be taken in the case of an equipment check error. The functions of the parameters are described below.

RETRY can be specified only for the a PRT1 printer. RETRY indicates that if an equipment check with command retry is encountered, the command is retried once. If the retry is unsuccessful a message is issued and the job is canceled.

IGNORE can be specified only for the 3525. IGNORE indicates that the error is to be ignored. The address of the record in error is put in register 1 and made available for processing. Byte 3, bit 3 of the CCB is also set on (see Figure 9-3); you can check this bit and take the appropriate action to recover from the error. IGNORE must not be specified for files with two I/O areas or a work area.

ERROPT=name can be specified only for a 3211-compatible printer. It indicates that if an equipment check with command retry is encountered, the command is retried once. If the retry is unsuccessful a message is issued and the job is canceled. With other types of errors (for these see the CCB, Figure 9-3) an error message is issued, error information is placed in the CCB, and control is given to your error routine, where you may perform whatever actions are desired. If any IOCS macros are issued in the routine, register 14 must be saved; if the operand RDONLY=YES is specified, register 13 must also be saved. To continue processing at the end of the routine, return to IOCS by branching to the address in register 14.

Processing Console Files

DTFCN defines an input or output file that is processed on a 3210 or 3215 console printer-keyboard, or a display operator console. DTFCN provides GET/PUT logic as well as PUTR logic for a file, and does not require a separate logic module macro to be coded.

Figure 7-10 lists the keyword operands contained in the operand field.

Programming Considerations

Communication with the operator console uses GET or PUT logic, combined with a TYPEFLE=INPUT defintion for GET, and OUTPUT specification for PUT. In addition, you may use the PUTR (PUT with reply) macro to issue a message to the operator that requires operator action and which will not be deleted from the display screen until the operator has issued a reply.

You may also use PUTR with the 3210 or 3215 console printer-keyboard, in which case PUTR functions the same as PUT followed by GET for these devices, but provides the message non-deletion code for the display operator console. Use of PUTR for the 3210 or 3215 is therefore recommended for compatibility if your program may at some time be run on the display operator console instead of the 3210 or 3215.

Use PUTR for fixed unblocked records (messages). Issue PUTR after a record has been built.

If PUTR is used in a program, TYPEFLE=CMBND must be specified. DEVADDR=SYSLOG must be specified if your DTFCN macro includes TYPEFLE=CMBND.

The IOAREA1 operand specifies the name of the I/O area used by the file. For PUTR macro usage, the first part of the I/O area is used for output, and the second part is used for input. The lengths of these parts are specified by the BLKSIZE and INPSIZE operands respectively. The I/O area is not cleared before or after a message is printed, or when a message is canceled and reentered on the console.

The BLKSIZE operand specifies the length of the I/O area; if the PUTR macro is used (that is, if TYPEFLE=CMBND is specified), this operand specifies the length of the output part of the I/O area. For the undefined record format, BLKSIZE must be as large as the largest record to be processed. The length must not exceed 256 characters.

Processing Magnetic Reader Files

Before a 1255, 1259, or 1419 magnetic reader input file can be processed, it must first be defined by the DTFMR and MRMOD macros. The operands of DTFMR are listed in Figure 7-11; for details, see *VSE/Advanced Functions Macro Reference.* The DTFMR and MRMOD macros also define files for the 1270 and 1275 optical reader/sorter.

Some general characteristics of magnetic reader processing are discussed below. For a discussion of optical reader processing, see the following section on the topic.

Characteristics of Magnetic Ink Character Reader (MICR)

Important general characteristics of Magnetic Ink Character Reader (MICR) processing are given in the *VSE System Data Management Concepts.*

In addition, examples of GET-PUT document processing and multiple 1419 operation (either all single or dual) will be found in *VSE/Advanced Functions System Generation.*

MICR Document Buffer

The MICR Document Buffer provides you with processing status indicators and detected error indicators. Before you can begin any MICR programming, you must be aware of the purpose and format of this buffer.

M	DEVADDR=SYSxxx	Symbolic unit for the console used for this file.
M	IOAREA1=xxxxxxxx	Name of I/O area.
O	BLKSIZE=nnn	Length in bytes of I/O area (for PUTR macro usage, length of output part of I/O area). If RECFORM=UNDEF, maximum is 256. If omitted, 80 is assumed.
O	INPSIZE=nnn	Length in bytes for input part of I/O area for PUTR macro usage.
O	MODNAME=xxxxxxx	Logic module name for this DTF. If omitted, IOCS generates a standard name. The logic module is generated as part of the DTF.
O	RECFORM=xxxxxx	(FIXUNB or UNDEF). If omitted, FIXUNB is assumed.
O	RECSIZE=(nn)	Register number if RECFORM=UNDEF. General registers 2-12, written in parentheses.
O	TYPEFLE=xxxxxx	(INPUT, OUTPUT, or CMBND). INPUT processes both input and output. CMBND must be specified for PUTR macro usage. If omitted, INPUT is assumed.
O	WORKA=YES	GET or PUT specifies work area.

M = Mandatory
O = Optional

Figure 7-10. DTFCN macro operands.

M	DEVADDR=SYSnnn	Symbolic unit assigned to the magnetic character reader.
M	IOAREA1=xxxxxxxx	Name of the document buffer area.
O	ADDAREA=nnn	Additional document buffer area (ADDAREA+RECSIZE=250). If omitted, no area is allotted.
O	ADDRESS=DUAL	Must be included only if the device is a 1419 or 1275 with a dual address adapter.
O	BUFFERS=nnn	Specifies the number of buffers needed. If omitted, 25 is assumed.
O	ERROPT=xxxxxxxx	Name of your error routine. Required only if the CHECK macro is used.
O	EXTADDR=xxxxxxxx	Name of your stacker selection routine. Required only if SORTMDE=ON.
O	IOREG=(nn)	Pointer register number. If omitted, register 2 is assumed. General registers 2-12, written in parentheses.
O	MODNAME=xxxxxxxx	Name of your I/O module. Required only if a nonstandard module is referenced.
O	RECSIZE=nnn	Specifies the maximum record length. If omitted, 80 is assumed.
O	SECADDR=SYSnnn	Specifies secondary symbolic unit assigned to (dual address) 1275 or 1419. Required only if LITE macro is used.
O	SEPASMB=YES	Required only if the DTF is assembled separately; otherwise it should be omitted.
O	SORTMDE=xxx	ON-1255/1259/1270 or program sort mode used; OFF-1419/1275 sort mode used. If omitted, ON is assumed.

M = Mandatory
O = Optional

Figure 7-11. DTFMR macro operands.

Figure 7-12 is a storage map of the document buffer. The minimum number of document buffers you may specify is 12, and the maximum number is 254. Before any data is read into the document buffer, logical IOCS sets the entire buffer, including the status indicators, to binary zeros. The processing macro - GET if your program uses one MICR device, or READ if your program uses more than one MICR device - then engages the device, and documents are read into the I/O area until the MICR device is out of documents, or until the I/O area is filled. The external interrupt routine of the supervisor continually monitors the reading of data so that processing of other document buffers is never disrupted. At the completion of each read for a MICR document, the external interrupt routine interrupts your program to give control to your stacker selection routine which then determines pocket selection for that document.

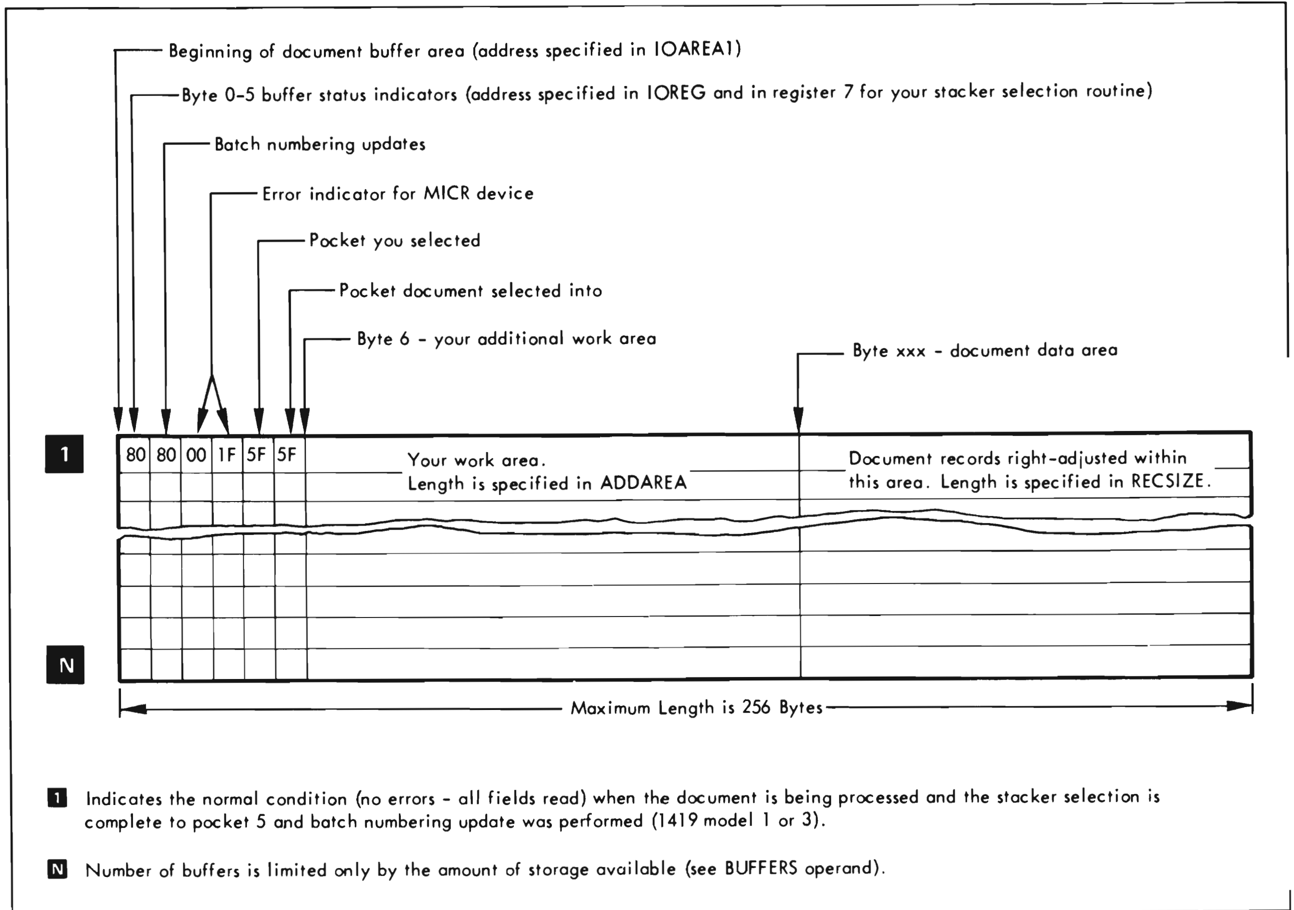
The MICR document buffer format is given in detail in Figure 7-13.

Stacker Selection Routine for MICR

Your stacker selection routine resides in your program area and receives control whenever a document is ready to be stacker selected. This routine determines the pocket (stacker) selected to receive the document and whether batch numbering update is to be performed (1419 only). The entry point is specified in the DTFMR operand EXTADDR=name. All registers are saved upon exiting from, and restored upon returning to, your program. The use of the general registers in this routine is as follows:

Register	Comment
0-4,6,8-15	These registers are available to your stacker selection routine for any purpose. However, because the routine can be interrupted at any time, the contents of these registers are unpredictable. If you want to reference an area outside the stacker select routine, you must reestablish addressability to the referenced area. That is, you must re-load your USING register (by means of, for instance, an address constant placed within the stacker select routine).
5	When your stacker selection routine is entered, this register contains the address of the routine. Register 5 should be utilized as the base register for the routine.
7	This register alway contains the address of the buffer for the document being selected. Bytes 2 and 3 of the buffer (see Figure 7-13) indicate the read status of the document.

Before entering your stacker selection routine, IOCS aids in stacker selection by setting the entire document buffer to binary zeros, reading the document into the document data area, and posting information in bytes 2 and 3. When the stacker selection routine has determined which pocket to select for the document, the actual stacker selection command code for this pocket must be placed into byte 4 of the document buffer pointed to by register 7. The final destination of the document is indicated in byte 5 of the buffer. This indication is the same as byte 4 except in the case of a late stacker select, an auto-selected document, a program malfunction, or a device malfunction. Any of these results in an I/O error. The reject code X'CF', indicating that the document is placed in the reject pocket, is placed in byte 5.



1 Indicates the normal condition (no errors - all fields read) when the document is being processed and the stacker selection is complete to pocket 5 and batch numbering update was performed (1419 model 1 or 3).

N Number of buffers is limited only by the amount of storage available (see BUFFERS operand).

Figure 7-12. MICR document buffer.

The command codes to be used to select pockets are:

Pocket	Code
A	X'AF' (Only for 1419)
B	X'BF' (Only for 1419)
0	X'0F'
1	X'1F'
2	X'2F'
3	X'3F'
4	X'4F'
5	X'5F'
6	X'6F'
7	X'7F'
8	X'8F'
9	X'9F'
Reject	X'CF'

An invalid code placed in byte 4 puts the document into the reject pocket and posts bit 1 of byte 0 of the buffer. Byte 0, bit 2 of the next buffer is posted.

Before returning to a 1419 external interrupt routine via the EXIT macro with the MR operand (required method), you can request a batch numbering update. You can do this only within a your 1419 stacker selection routine by turning on byte 1, bit 0 in the current document buffer. The instruction

```
OI 1(7),X'80'
```

does this for you.

For the 1419 (dual address), you cannot obtain batch numbering update on an auto-selected document (byte 2, bit 6 on). Such requests are ignored by the external interrupt routine.

Timings for Stacker Selection

Because an MICR reader continuously feeds documents while engaged, it is necessary to reinstruct the reader within a certain time limit after a read completion is signaled by an external interrupt. This period is generally called minimum stacker selection time. This available time depends on the reader model, the length of documents being read, single or dual address adapter (1419), and the fields to be read on the 1419 (dual address) only. Refer to the appropriate MICR publications listed in the latest *IBM System/370 and 4300 Processors Bibliography,* for a more complete description of device timings.

Failure to reinstruct the 1255, 1259, or 1419 (single address adapter) within the allotted time causes the document(s) processed after this time to

be auto-selected into the reject pocket (late read condition). Failure to reinstruct the 1419 (dual address adapter) within the allotted time causes the document being processed to be autoselected into the reject pocket (late stacker-select condition).

Programming Considerations for 1419 Stacker Selection

The stacker selection routine operates in the program state with the protection key of its program and with I/O and external interruption disabled. If your stacker selection routine fails to return to the supervisor (loops indefinitely), there is no possible recovery. If such looping occurs, the system must be re-IPLed to continue operation. It is therefore recommended that you thoroughly debug your stacker selection routine in a dedicated environment.

In your stacker selection routine, no system macro other than EXIT MR can be used. The routine runs with an all zero program and system mask, but the machine check interruption is enabled and a program check cancels the program.

Note: Any modification of floating point registers without saving and restoring them may cause erroneous processing by any concurrent program using floating-point instructions.

When processing with the dual address adapter on the 1419, you have more time for your stacker selection routine. The only additional processing you must do within the main line is to check byte 2, bit 0, of the document buffer for stacker selection errors.

Note: Batch numbering is not performed with the stacker selection of auto-selected documents.

Programming Considerations

MICR devices can be operated in any partition. The user is supplied with an extension to the supervisor which monitors, by means of external interrupts, the reading of documents into a user-supplied I/O area (document buffer area).

The user must access all MICR documents through logical IOCS macros. Upon request, IOCS gives a next sequential document and automatically engages and disengages the devices to provide a continuous stream of input. Detected error conditions and information about errors are passed to the user in each docment buffer. Documents are read at a rate dictated by the device rather than by the program. To allow time for necessary processing (including the determination of pocket selection), the device generates an external interruption at the completion of each read operation for each document. The supervisor gives absolute priority to external interrupt processing.

In problem programs, these devices can be controlled by assembler language only, at the LIOCS GET level if one device is attached, or at the LIOCS READ/CHECK/WAITF level if multiple MICR devices are attached.

Within a particular program, you should utilize either the GET macro or the READ, CHECK, WAITF combination.

For a program operating with two or more MICR devices, the READ, CHECK, WAITF combination allows processing to continue within the program when any document buffer is ready for processing. On the other hand, the GET macro (suggested for a program operating with only one MICR device) includes an inherent wait for a document buffer to become available within the file before processing begins. Control always passes to another partition whenever a WAIT condition occurs.

The DTFMR and the MRMOD declarative macros are used to describe the file. For any type of processing you need a document buffer area with a special buffer format. A document buffer must not exceed 256 bytes, including the six-byte buffer status indicators, any additional user work area, and the maximum document data area. You may specify any number of document buffers between 12 and 254; the actual maximum number depends on the amount of virtual storage available.

Before any MICR document processing can be done, the file(s) must be opened. For MICR devices, OPEN sets the entire I/O area to binary zeros.

The first time a GET(or READ) is executed, the supervisor engages the device for continuous reading. Each time thereafter, the GET (or READ) merely points (through IOREG) to the next sequential buffer within each document buffer area. When a buffer for a file becomes available, the user's main line processing continues with the instruction after the GET (or READ, CHECK combination).

When the GET macro detects an end-of-file condition, IOCS branches to your end-of-file routine (specified by EOFADDR). For MICR document processing, you do not regain control until either a buffer becomes filled with a stacker-selected document, or error conditions are posted in the buffer status indicators.

If an unrecoverable I/O error occurs when a GET macro is executed, no more GETs can be issued for the file. If an unrecoverable I/O error occurs when using the READ, CHECK, WAITF combination or when document processing for that file is complete, you can effectively continue by closing the file. Further READ, CHECK, WAITFs treat this file as having no

documents ready for processing (see byte 0, bits 5 and 6 of the document buffer in Figure 7-13).

Each time an end-of-document condition occurs, the user's main line processing routine, or any other routine having control at that time, is interrupted by the supervisor's external interrupt routine. The external interrupt routine branches immediately to the user's stacker selection routine. After selecting a pocket, you exit from your stacker selection routine so that the supervisor can issue the stacker selection command. At this time, the MICR device should be reading document data into its respective document buffer area. The supervisor, in priority order, passes control to your main line processing routine, or to the routine that had been interrupted.

Thus, document processing continues concurrently (see Figure 7-14) within

(1) the user's main line processing routine,
(2) the supervisor's external interrupt routine, and
(3) the user's stacker selection routine.

The order for exiting from these routines is the reverse of the indicated order. Processing and monitor operations continue concurrently until the reader is disengaged, either normally or because of an error.

End-of-file must be detected and handled by the user's main line processing routine. You can use the DISEN macro to stop the feeding of documents through the MICR device: the program proceeds to the next sequential instruction without waiting for the disengagement to complete. You continue to issue GETs or READs until the unit exception bit indicates that all following documents have been processed.

The GET and the READ macros perform the same functions. The GET, however, waits while the document buffer fills, whereas the READ posts an indicator in the buffer for you to examine with the CHECK macro. If this indicator bit is on, the buffer is not ready for processing, and a branch is made to the second operand address of the CHECK macro. Your routine at this operand address can then READ and CHECK another file for document availability. If this buffer is ready for processing, control passes to the next instruction. If a special non-data status exists, you should analyze the conditions in your ERROPT routine and issue a READ to obtain a document unless an I/O error has occurred. If a second operand is not provided within the CHECK macro, control passes to the ERROPT routine address.

The READ filename, MR macro makes the next sequential buffer available to you, but it does not verify that it is ready for processing (the CHECK macro is provided to make that test). If the buffer is not ready for processing, the next READ to that file points to the same buffer. Filename specifies the name of the file associated with the record. It is the same as that specified in the DTFMR header entry. Register notation may be used. MR signifies that the file is for a magnetic ink character reader (MICR).

The CHECK macro examines the buffer status indicators. A READ macro must therefore have been issued to the file before a CHECK macro is issued.

The CHECK macro determines whether the buffer contains data ready for processing, is waiting for data, contains a special nondata status, or the file (filename) is closed. If the buffer has data ready for processing, control passes to the next sequential instruction. If the buffer is waiting for data, or the file is closed, control passes to the address specified for control address, if present. If the buffer contains a special nondata status, control passes to the ERROPT routine for you to examine the posted error conditions before determining your action. (See byte 0, bits 2, 3, and 4, of the document buffer in Figure 7-13.) Return from the ERROPT routine to the next sequential instruction via a branch on register 14, or to the control address in register 0.

If the buffer is waiting for data, or if the file is closed, and the control address is not present, control is given to you at the ERROPT address specified in the DTFMR macro.

If an error, a closed file, or a waiting condition occurs (with no control-address specified) and no ERROPT address is present, control is given to you at the next sequential instruction.

If the waiting condition occurred, byte 0, bit 5 of the buffer is set to 1. If the file was closed, byte 0, bits 5 and 6 of the buffer are set to 1 (see Figure 7-13.)

The WAITF (wait multiple) macro allows processing of programs in other partitions while waiting for document data. If any device within the WAITF macro list has records or error conditions ready to be processed, control remains in the partition and processing continues with the instruction following the WAITF macro.

One WAITF macro must be issued after a set of READ-CHECK combinations before your program attempts to return to a previously issued combination. Thus, the WAITF macro must be issed between successive executions of a particular READ macro.

The DISEN macro stops the feeding of documents through the magnetic character reader. The program proceeds to the next sequential instruction without waiting for the disengagement to complete.

Buffer Status Indicators		
Byte	**Bit**	**Comment**
0	0	The document is ready for processing (you need never test this bit).
	1	Unrecoverable stacker select error, but all document data is present. You may continue to issue GETs and READs.
	2	Unrecoverable I/O error. An operator I/O error message is issued. The file is inoperative and must be closed.
	3	Unit Exception. You requested disengage and all follow-up documents are processed. The LITE macro may be issued, and the next GET or READ engages the device for continued reading.
	4	Intervention required or disengage failure. This buffer contains no data. The next GET or READ continues normal processing. This indicator allows your program to give the operator information necessary to select pockets for documents not properly selected and to determine unread documents.
	5	The program issued a READ, no document is ready for processing, byte 0, bits 0 to 2 are off, or the file is closed (byte 0, bit 6 is on). The CHECK macro interrogates this bit. **Note:** You must test bits 1 through 4 and take appropriate action. Any data from a buffer should not be processed if bits 2, 3, or 4 are on.
	6	The program has issued a GET or READ and the file is closed. Bit 5 is also on.
	7	Reserved.
1	0	Your stacker selection routine turns this bit on to indicate that batch numbering update (1419 only) is to be performed in conjunction with the stacker selection for this document. The document is imprinted with the updated batch number unless a late stacker selection occurs (byte 3, bit 2).
	1-7	Reserved **Note:** If bits 6 or 7 (byte 2) are on, bit 0 is ignored by the external interrupt routine. With the 1419 (dual address) only, batch numbering update cannot be performed with the stacker selection of auto-selected documents.
2[1]	0	For 1419 or 1275 (dual address) only. An auto-select condition occurred after the termination of a READ but before a stacker select command. The document is auto-selected into the reject pocket.
	1-3	Reserved.
	4	Data check occurred while reading. You should interrogate byte 3 to determine the error fields.
	5	Overrun occurred while reading. Byte 3 should be interrogated to determine the error fields. Overruns cause short length data fields. When the 1419 or 1275 is enabled for fixed-length data fields, bit 4 is set.
	6-7	The specific meanings of bits 6 and 7 depend on the device type, the model, and the Engineering Change level of the MICR reader; but if either bit is on, the document(s) concerned is (are) auto-selected into the reject pocket. 1. **1412 or 1270:** Bit 6 on indicates that a late read condition occurred. Bit 7 on indicates that a document spacing error occurred. (Unique to the 1270: both the current document and the previous document are auto-selected into the reject pocket when this bit is on. This previous document reject cannot be detected by IOCS, and byte 5 of its document buffer does not reflect that the reject pocket was selected.) 2. **1275 and 1419 (single address) without engineering change #125358:** Bit 6 indicates that either a late read condition or a document spacing error occurred. Bit 7 indicates a document spacing error for the current document. 3. **1255, 1259, 1275, and 1419 (single or dual address) with engineering change #125358:** Bit 6 indicates that an auto-select condition occurred while reading a document. The bit is set at the termination of the READ command before the stacker select routine receives control. Bit 7 is always zero.
[1] Byte 2 (bits 4, 5, 6, and 7) and byte 3 contain MICR sense information.		

Figure 7-13. MICR document buffer format (Part 1 of 2).

You should continue to issue GETs or READs until the unit exception bit (byte 0, bit 3) of the buffer status indicators is set on.

The LITE macro lights any combination of pocket lights on a 1419 magnetic character reader.

Before using the LITE macro, the DISEN macro must be issued to disengage the device. Processing of the documents should be continued until the unit exception bit (byte 0, bit 3) of the buffer status indicators is set on (see Figure 7-13). When this bit is on, the follow-up documents have been processed, the MICR reader has been disengaged, and the LITE macro can be issued.

The bit configuration for the pocket light switch area is shown in Figure 7-15. The pocket lights that are turned on should have their indicator bits set to 1. If an error occurs during the execution of the pocket lighting I/O commands, bit 7 in byte 1 is set to 1. This error condition normally indicates that the pocket light operation was unsuccessful.

Processing Optical Reader Files

Before processing, files for the 1270/1275 Optical Reader/Sorters must be defined by a DTFMR macro. The operands for this macro were listed in Figure 7-11 in the preceding section on Magnetic Readers.

Programs for the 3881 Optical Mark Reader re-

Buffer Status Indicators		
Byte	**Bit**	**Comment**
3[1]	0	Field 6 valid.[2]
	1	Field 7 valid.[2]
	2	A late stacker selection (unit check late stacker select on the stacker select command). The document is auto-selected into the reject pocket.
	3	Amount field valid (or field 1 valid).[2]
	4	Process control field valid (or field 2 valid).[2]
	5	Account number field valid (or field 3 valid).[2]
	6	Transit field valid (or field 4 valid).[2]
	7	Serial number field valid (or field 5 valid).[2]
		Notes: 1. For the 1270, bits 3-7 are set to zero when the fields are read without error. 2. For the 1255, 1259, 1275, and 1419, bits 3-7 are set on when each respective field, including bracket symbols, is read without error. This applies to bits 0, 1, and 3-7 on the 1259 and 1419 model 32. 3. For the 1255, 1259, 1275, and 1419, unread fields contain zero bits. Errors are indicated when an overrun or data check condition occurs while reading the data field.
4		Inserted pocket code determination by your stacker select routine. Whenever byte 0, bits 2, 3, or 4 are on, this byte is X'00' because no document was read and your stacker selection routine was not entered. Whenever auto-selection occurs, this value is ignored. A no-op (X'03') is issued to the device, and a reject pocket value (X'CF') is placed in byte 5. The pocket codes are (byte 2, bit 6 or 7 on): Pocket A - X'AF'[3] Pocket 5 - X'5F' Pocket B - X'BF'[4] Pocket 6 - X'6F' Except 1270 Pocket 0 - X'0F' Pocket 7 - X'7F' models 1 and 3 Pocket 1 - X'1F' Pocket 8 - X'8F' Pocket 2 - X'2F' Pocket 9 - X'9F' Pocket 3 - X'3F' Reject Pocket - X'CF' Pocket 4 - X'4F'
5		The actual pocket selected for the document. The contents are normally the same as that in byte 4. **Note:** 1. X'CF' is inserted whenever auto-selection occurs (byte 2, bit 6; byte 2, bit 7; byte 2, bit 0; or byte 3, bit 2). These conditions may result from late READ commands, errant document spacing, or late stacker selection. a. Start I/O for stacker selection is unsuccessful (byte 0, bit 1). b. An I/O error occurs (for example, invalid pocket code) on the 1419 (dual address) secondary control unit when selecting this document.
Additional User Work Areas		
This additional buffer area can be used as a work area and/or output area. Its size is determined by the DTFMR ADDAREA operand. The only size restriction is that this area, plus the 6-byte status indicators and data portion must not exceed 256 bytes. This area may be omitted.		
Document Data Area		
The document data area immediately follows your work area. The data is right-adjusted in the document data area. The length of this data area is determined by the DTFMR RECSIZE operand.		
[1] Byte 2 (bits 4, 5, 6, and 7) and byte 3 contain MICR sense information. [2] Only for the 1259 model 34 or 1419 model 32. Bits 0 and 1 are not used for other models. [3] 1275, 1419, and 1270 models 2 and 4 only. [4] 1275 and 1419 only.		

Figure 7-13. MICR document buffer format (Part 2 of 2).

quire its files to be defined with a DTFCD macro, whose operands are listed in Figure 7-1, in the section on Punched Card files.

You must use the DTFDR macro to define each 3886 Optical Character Reader file in your program. This macro defines the characteristics of the file, the format record to be loaded into the 3886 when the file is opened, and the storage areas and routines used. In addition, LIOCS requires you to code the following macros together with the DTFDR:

DRMOD to have the assembler generate the logic module needed to process the file.

DFR to define attributes common to a group of lines described in one format record.

DLINT to describe the individual line in the format record.

The operands for DTFDR, DFR, and DLINT are listed in Figures 7-16, 7-17, and 7-18.

DTFOR is used to define input files to be processed on a 1287 Optical Reader or a 1288 Optical Page Reader; its operands are listed in Figure 7-19.

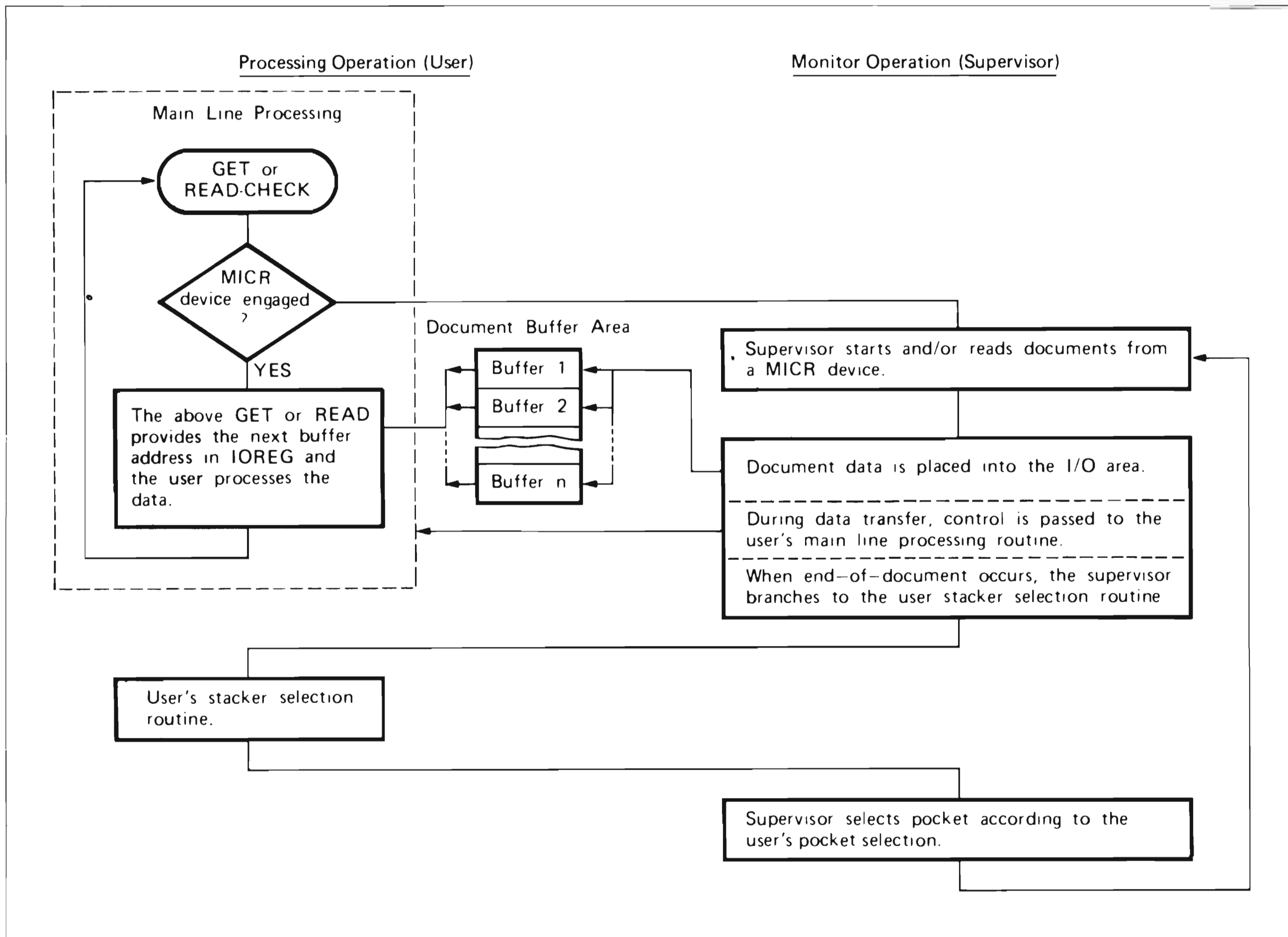


Figure 7-14. MICR/OCR document processing.

Bits	0	1	2	3	4	5	6	7	8	9	A	B	C D E	F
Pocket Lights	A	B	0	1	2	3	4	5	6	7	8	9	Reserved	Error indicator bit

Figure 7-15. Bit configuration for pocket light switch area of the 1419/1275.

M	COREXIT=xxxxxxxx	Name of your error condition routine.
M	DEVADDR=SYSxxx	Symbolic unit assigned to 3886 optical character reader.
M	EOFADDR=xxxxxxxx	Address of your end-of-file routine.
M	EXITIND=xxxxxxxx	Name of completion code return area.
M	FRNAME=xxxxxxxx	Phase name of format record to be loaded upon file opening.
M	FRSIZE=nn	Number of bytes to be reserved in DTF expansion for format records.
M	HEADER=xxxxxxxx	Name of area for header record from 3886.
M	IOAREA1=xxxxxxxx	Name of file input area.
O	BLKSIZE=nnn	Length of area named by IOREG1. If omitted, the maximum length of 130 is assumed.
O	DEVICE=3886	If omitted, 3886 is assumed.
O	MODNAME=xxxxxx	Name of DRMODxx logic module for this DTF. If omitted, IOCS generates standard name.
O	RDONLY=YES	If DTF is to be used with read-only module.
O	SEPASMB=YES	If DTFDR is to be assembled separately.
O	SETDEV=YES	If SETDEV macro is issued in your program to load a different format record into the 3886.

M = Mandatory
O = Optional

Figure 7-16. DTFDR macro operands.

M	FONT=xxxx	Default font for all codes described by format record.
O	BCH=n	Batch numbering is to be performed by 3886. If used, BCHSER is invalid.
O	BCHSER=n	Both batch and serial numbering are to be performed. If specified, BCH is invalid.
O	CHRSET=n	Specifies recognizing character (see Figure 7-16). If omitted, 0 is assumed.
O	EDCHAR=(x, ...,)	Characters that may be deleted from any field that is read. If omitted, no character deletion occurs.
O	ERASE=YES	Group and character erase symbols are to be recognized. If omitted, NO is assumed.
O	NATNHP=YES	European Numeric Hand Printing (ENHP) characters 1 and 7 are used. If omitted, NO is assumed, indicating that Numeric Hand Printing (NHP) characters 1, 7 are used.
O	REJECT=x	Replacement character for any reject character in the data record read by the 3886. If omitted, X'3F' is assumed.

M = Mandatory
O = Optional

Figure 7-17. DFR macro operands.

M	LFR=nn	Line format record for this line.
M	LINBEG=nn	Specifies beginning of a line.
O	IMAGE=YES	Data record is to be in image mode. If omitted, NO (standard mode) is assumed.
O	NOSCAN=(n,n)	Indicates an area on the document line that is to be ignored by the 3886.
O	FLDn=(n,n,NCRIT,xxx)	Describes a field in a line. n in the FLD keyword may be from 1 to 14; if specified, a corresponding EDITn keyword must follow each FLDn keyword.
O	EDITn=(xxxxxx,EDCHAR)	Specifies editing functions to be performed on the data by 3886. A corresponding FLDn keyword must precede each EDITn keyword.
O	FREND=YES	Indicates last DLINT macro for the format record. If omitted, NO is assumed meaning that further DLINT macros follow.

M = Mandatory
O = Optional

Figure 7-18. DLINT macro operands.

For detailed discussions of the operands for these macros, see *VSE/Advanced Functions Macro Reference.*

Applies to					
1287T	1287D	1288			
X	X	X	M	COREXIT=xxxxxxxx	Name of your correction routine.
X	X	X	M	DEVADDR=SYSnnn	Symbolic unit assigned to the optical reader.
X	X	X	M	EOFADDR=xxxxxxxx	Name of your end-of-file routine.
X	X	X	M	IOAREA1=xxxxxxxx	Name of first input area.
X			O	BLKFAC=nn	If RECFORM=UNDEF in journal tape mode.
X	X	X	O	BLKSIZE=nn	Length of I/O area(s). If omitted, 38 is assumed.
X	X	X	O	CONTROL=YES	If CNTRL macro is to be used for this file.
X	X	X	O	DEVICE=xxxxx	(1287D or 1287T). For 1288, specify 1287D. If omitted, 1287D is assumed.
X	X		O	HEADER=YES	If a header record is to be read from the optical reader keyboard by OPEN.
	X	X	O	HPRMTY=YES	If hopper empty condition is to be returned.
X			O	IOAREA2=xxxxxxxx	If two input areas are used, name of second input area.
X			O	IOREG=(nn)	Register number if two input areas or UNDEF records are to be used. If omitted, register 2 is assumed. General registers 2-12, written in parentheses.
X	X	X	O	MODNAME=xxxxxxxx	Name of logic module. If omitted, IOCS generates a standard name.
X	X	X	O	RECFORM=xxxxxx	(FIXBLK, FIXUNB, or UNDEF). If omitted, FIXUNB is assumed.
X	X	X	O	RECSIZE=(nn)	Register number containing record size, if RECFORM=UNDEF. If omitted, register 3 is assumed.
X	X	X	O	SEPASMB=YES	If the DTFOR is to be assembled separately.
X			O	WORKA=YES	If records are to be processed in a work area. Omit IOREG.

M = Mandatory
O = Optional

Figure 7-19. DTFOR macro operands.

Non-Data Device Operations

The CNTRL (control) macro provides commands that apply to physical non-data operations of an I/O unit and are specific to the unit involved.

For optical readers, commands specify marking error lines, correcting a line for journal tapes, document stacker selecting, or ejecting and incrementing documents. The CNTRL macro does not wait for completion of the command before returning control to you, except when certain mnemonic codes are specified for optical readers.

CNTRL usually requires two or three parameters. The first parameter must be the name of the file specified in the DTF header entry. It can be specified as a symbol or in register notation.

The second parameter is the mnemonic code for the command to be performed. This must be one of a set of predetermined codes (see Figure 7-20).

The third parameter, n1, is required whenever a number is needed for stacker selection, immediate printer carriage control, or for line or page marking on the 3886. The fourth parameter, n2, applies to delayed spacing or skipping, or to timing mark check on the 3886. In the case of a printer file, the parameters n1 and n2 may be required.

Whenever CNTRL is issued in your program, the DTF CONTROL operand must be included (except for DTFDR) and CTLCHR must be omitted. If control characters are used when CONTROL is specified, the control characters are ignored and treated as data.

1287 and 1288 Optical Reader Codes

The CNTRL macro for the 1287 and 1288 is used for the non-data functions of marking a journal tape line, incrementing a document, and ejecting and/or stacker selecting a document. It is also used to read data from the 1287 keyboard when processing journal tapes.

When the CNTRL macro is used with the READKB mnemonic, it allows a complete line to be read from the 1287 keyboard when processing journal tapes. This permits the operator to key in a complete line on the keyboard if a 1287 read error makes this type of correction necessary. When IOCS exits to your COREXIT routine, you may issue the CNTRL macro to read from the keyboard. The 1287 display tube then displays the full line and the operator keys in the correct line from the keyboard, if possible. The line

IBM Unit	Mnemonic Code	n_1	n_2	Command
3881 Optical Mark Reader	PS	1 2		Select Stacker 1 or 2
1287 Optical Reader	MARK			Mark Error Line in Tape Mode.
	READKB			Read 1287 Keyboard in Tape Mode.
	EJD			Eject Document.
	SSD	1 2 3 4		Select Stacker A, B, Reject, or Alternate Stacking Mode.
	ESD	14		Eject Document and Select Stacker.
	INC			Increment Document at Read Station.
1288 Optical Page Reader	ESD	1 3		Select Stacker A. Reject Stacker (R).
	INC			Increment Document at Read Station.
3886 Optical Character Reader	DMK	name (r) number		Page mark the document when it is stacker selected as specified in parameter n1.
	LMK	name (r) number, number		Line mark the document when it is stacker selected as specified in parameter n1.
	ESP	1 2	name (r) number	Eject and stacker select the current document to stacker A or B. Perform line mark station tuning mark check as indicated in parameter n2.

Figure 7-20. CNTRL macro command codes.

read from the keyboard is always read left-justified into the correct input area. The macro resets this area to binary zeros before the line is read.

After CNTRL READKB is used, the contents of filename+80 are meaningful only for a wrong-length error indication (X'04'). Therefore, you must determine whether the operator was able to recognize the unreadable line of data. The CNTRL macro with the READKB mnemonic waits for completion of the order before returning control to the user.

When processing journal tapes, the CNTRL macro with the MARK mnemonic marks (under program control) a line on the input tape that results in a data transfer error or is otherwise suspect of error. To ensure that the proper line is marked, the CNTRL macro must be issued in your error correction routine (specified in DTFOR COREXIT). If CNTRL is issued at any other time, the line following the one in error is marked.

When processing is done in document mode on the 1287, each document may be ejected with a CNTRL macro. The EJD mnemomic causes the document to eject and the next document to be fed. Documents may also be stacker selected by using the CNTRL macro with the SSD mnemonic.

The CNTRL macro with the ESD mnemonic combines the ejection and stacker selection functions. To satisfy the alternate ejection and stacker selection functions, the combined mnemonic must not be immediately preceded by an eject or immediately followed by a stacker select.

A document may be directed to stacker A, B, or R (reject stacker) by specifying a selection number of 1, 2, or 3 respectively. Also, documents may be selected into stackers A and B in an alternate stacking mode, with automatic stacker switching when one stacker becomes full. The selection number for alternate mode is 4. If selection number 4 is used in the first stacker selection macro, stacker A is filled first. If selection number 4 is used after other selection numbers, the last preceding selection number determines the first stacker to be filled. Only selection numbers 1 and 3 are available for the 1288.

If a CNTRL macro is issued in a COREXIT routine and a late stacker select or nonrecoverable error condition occurs, IOCS branches to the next sequential instruction. Filename+80 should therefore be tested for these conditions after issuing a CNTRL macro.

The CNTRL macro with the INC mnemonic may be used for document incrementation. For the 1287, this macro is not used with documents having a scannable area shorter that 6 inches (15.24 cm). When this mnemonic is issued, the document is incremented forward 3 inches (7.62 cm). This macro may be used only once per document.

For the 1288, the CNTRL macro with the INC mnemonic can increment only documents with a scannable area longer than 6.5 inches (16.51 cm). The document is incremented to the next stopping point as selected by console switches on the 1288. More than one CNTRL macro can be used per document.

Document ejection and/or stacker selection and document increment functions can also be accomplished be including the appropriate CCW(s) within the CCW list addressed by the READ macro, rather than by using the CNTRL macro. This technique results in increased document throughput.

Note: For processing documents in a multiprogramming environment where the partition containing 1287 support does not have highest priority, the eject and stacker select functions must be accomplished by a single command. However, when processing documents in a dedicated environment, the stacker select command can be executed separately. It must follow the eject command within 270 milliseconds if the document was incremented, or within 295 milliseconds if the document was not incremented. The eject and stacker select function must occur alternately. If the timing requirements are not met, a late stacker selection condition occurs.

3886 Optical Character Reader Codes

When you are using the 3886 Optical Character Reader, you can use the CNTRL macro to perform the following operations:

- Page mark the current document
- Line mark the current document
- Eject and stacker select the current document
- Perform timing mark check.

When the operation has been completed successfully, control is returned to the next instruction in your program. If the operation does not complete successfully, the COREXIT routine receives control. The end-of-file routine receives control when an operation is requested but no documents are available and the end-of-file key has been pressed.

The contents of parameters n1 and n2 vary depending on the mnemonic operation code specified. Therefore, this discussion treats each mnemonic code separately.

DMK,n1: Specifies that the document currently being processed is to be marked when the next eject/stacker-select command is issued. The digits to be printed on the page are specified by the four low-order bits of the field indicated in parameter n1. The sum of the mark digits printed will equal the value specified in the four bits. The high-order four bits of the field are not used. You can specify the digits you want printed in one of three ways:

- *name* specifies the symbolic name of a one-byte field in your program in which the low-order four digits indicate the combination of digits to be printed.
- *(r)* indicates the number of the register that contains the address of the one-byte field used for page marking.
- *number* indicates the sum of the digits to be printed. The decimal number may be any from 1 through 15.

LMK,n1: Specifies a line on the current document that is to be line-marked when the eject/stacker-select command is issued. The digits to be printed and the line on which they should be printed are specified in a two-byte field. The digits to be printed are specified in the low-order four bits of the first byte as in the document marking operation. The line to be marked is specified in binary in the low-order six bits of the second byte of the field. You can specify the mark digits and line number in three ways:

- *name* specifies the symbolic name of a two-byte hexadecimal field in your program that contains the necessary information.
- *(r)* indicates the number of the register that contains the address of the two-byte field with the necessary information.
- *number,number* provides first, the sum of the decimal digits to be printed (any number from 1 to 15) and second, the decimal line number to be marked (any number from 1 to 33).

ESP,n1,n2: n1 specifies that the current document should be ejected immediately and routed to stacker 1 or 2. (The valid entries are 1 and 2). A request for timing mark check can also be made in this parameter. If the number of timing marks on the document disagrees with the number you specify, either a nonrecovery error or timing mark check error occurs. You can specify the number of timing marks, by using parameter n2, in three ways:

- *name* specifies the name of a one-byte hexadecimal field in your program that indicates the number of timing marks that should be on the document.
- *(r)* specifies the number of the register that contains the address of the one-byte hexadecimal field containing the expected number of timing marks.
- *number* is a decimal number from 0 through 33 specifying the number of timing marks that should be on the document.

If the number of timing marks is not specified or if zero is specified, no timing mark check is performed.

3881 Optical Mark Reader Codes
Documents read by the 3881 are directed to the stacker specified in the CNTRL macro or to the stacker specified on the format control sheet. Stacker 1 is the normal stacker and stacker 2 is the select stacker. If you use both the CNTRL macro and the format control sheet to control stacker selection and either specifies stacker 2, data documents are stacked in stacker 2. The DTF SSELECT operand is not valid for the 3881.

Programming Considerations for Optical Readers

There are four parts to this section; they apply to:

- IBM 1270, 1275 Optical Readers/Sorters
- IBM 1287 Optical Reader and IBM 1288 Optical Page Reader
- IBM 3886 Optical Character Reader
- IBM 3881 Optical Mark Reader.

Optical Readers/Sorters (IBM 1270, IBM 1275)

Optical Character Reader/Sorter (OCR) devices can be operated in any partition. The user is supplied with an extension to the supervisor which monitors, by means of external interrupts, the reading of documents into a user-supplied I/O area (document buffer area).

The user must access all OCR documents through logical IOCS macros. Upon request, LIOCS gives a next sequential document and automatically engages and disengages the devices to provide a continuous stream of input. Detected error conditions and information about errors are passed to the user in each document buffer. Documents are read at a rate dictated by the device rather than by the program. To allow time for necessary processing (including the determination of pocket selection), the device generates an external interruption at the completion of each read operation for each document. The supervisor gives absolute priority to external interrupt processing.

In problem programs, these devices can be controlled by assembler language only, at the LIOCS GET level if one device is attached, or at the LIOCS READ/CHECK/WAIT level if multiple OCR devices are attached. In the latter case, you are allowed to continue processing as long as one file has documents ready for processing.

The DTFMR and the MRMOD declarative macros are used to describe the file. For any type of processing you need a document buffer area with a special buffer format. A document buffer must not exceed 256 bytes, including the six-byte buffer status indicators, any additional user work area, and the maximum document data area. You may specify any number of document buffers between 12 and 254; the actual maximum number depends on the amount of virtual storage available.

The first time a GET (or READ) is executed, the supervisor engages the device for continuous reading. Each time thereafter, the GET (or READ) merely points (through IOREG) to the next sequential buffer within each document buffer area. When a buffer for a file becomes available, the user's main line processing continues with the instruction after the GET (or READ CHECK combination).

Each time an end-of-document condition occurs, the user's main line processing routine, or any other routine having control at that time, is interrupted by the supervisor's external interrupt routine. The external interrupt routine branches immediately to the user's stacker selection routine. After selecting a pocket, you exit from your stacker selection routine so that the supervisor can issue the stacker selection command. At this time, the OCR device should be reading document data into its respective document buffer area. The supervisor, in priority order, passes control to your main line processing routine, or to the routine that had been interrupted.

Thus, document processing continues concurrently (see Figure 7-14) within

(1) the user's main line processing routine,

(2) the supervisor's external interrupt routine, and

(3) the user's stacker selection routine.

The order for exiting from these routines is the reverse of the indicated order. Processing and monitor operations continue concurrently until the reader is disengaged, either normally or due to error.

End-of-file must be detected and handled by the user's main line processing routine. You can use the DISEN macro to stop the feeding of documents through the OCR device: the program proceeds to the next sequential instruction without waiting for the disengagement to complete. You should continue to issue GETs or READs until the unit exception bit (byte 0, bit 3), of the buffer status indicators is set on (see Figure 7-13.)

When the IBM 1275 is equipped with the Program Control for Pocket lights (special feature), you can, by means of the LITE macro, light any combination

of pocket lights to indicate that a specified number of documents has entered the pockets.

Before using the LITE macro, the DISEN macro must be issued to disengage the device. Processing of the documents should be continued until the unit exception bit (byte 0, bit 3) of the buffer status indicators is set on (see Figure 7-13). When this bit is on, the follow-up documents have been processed, the reader has been disengaged, and the pocket LITE macro can be issued.

The bit configuration for the pocket light switch area is shown in Figure 7-15. The pocket lights that are turned on should have their indicator bits set to 1. If an error occurs during the execution of the pocket lighting I/O commands, bit 7 in byte 1 is set to 1. This error condition normally indicates that the pocket light operation was unsuccessful.

The GET and the READ macro perform the same functions. The GET, however, waits while the document buffer fills, whereas the READ posts an indicator in the buffer for you to examine with the CHECK macro. If this indicator bit is on, the buffer is not ready for processing, and a branch is made to the second operand address of the CHECK macro. Your routine at this operand address can then READ and CHECK another file for document availability. If this buffer is ready for processing, control passes to the next instruction. If a special non-data status exists, you should analyze the conditions in your ERROPT routine and issue a READ to obtain a document unless an I/O error has occurred. If a second operand is not provided within the CHECK macro, control passes to the ERROPT routine address.

At least one WAITF macro must be issued between two successive executions of any one READ to the same file. The multiple WAITF tests device operation availability or buffer processing availability. If work can be done on any specified file, control remains in the partition. If not, control passes to a lower-priority partition until this partition is ready for processing.

Optical Reader (IBM 1287) and Optical Page Reader (IBM 1288)

The IBM 1287 and 1288 can be operated in any partition. You must access all operations through LIOCS macros or through PIOCS. In your problem program, these devices are controlled by means of the assembler language: for 1287 journal tape processing at the LIOCS GET level, for 1287 and 1288 document processing at the READ/WAITF level. You use the DTFOR macro to describe the input file; the MRMOD macro generates the logic module to process the file. The non-data functions are performed by the CNTRL macro, which is used to increment, eject, and stacker select documents on the 1287 and 1288, as well as to mark error lines and to read keyboard information when reading journal tapes on the 1287.

You supply the name of your own COREXIT correction routine in the DTFOR macro. If an error condition occurs after a GET, WAITF, or CNTRL macro has been executed, COREXIT provides an exit to your error correction routine. In this routine you can reset a number of error conditions and take appropriate actions.

When processing journal tapes on the 1287, the RDLNE macro provides online correction; it causes the reader to read a line in online correction mode while processing in offline correction mode. When processing documents on the 1287 or 1288, you can use the RESCN macro to selectively reread a field on a document when a read error makes this necessary. The DSPLY macro displays a document field on the display screen and allows the 1287 operator to key in a complete field on the keyboard for correction.

When LIOCS is used for processing journal tapes on the 1287 optical reader, OPEN may be issued at the beginning of each input roll.

To process in two or more rolls on the 1287 as one file (when an end-of-tape condition occurs), instruct your operator to run out the tape by pressing the start key on the optical reader instead of the end-of-file key. This creates an intervention-required condition. The next tape can then be loaded and processed as a continuation of the previous tape. However, because OPEN is not reissued, no header information can be entered between tapes.

When processing documents on the 1287, OPEN must be issued to make the file available.

OPEN allows header (identifying) information to be entered at the 1287 keyboard for journal tape or cut documents. When header information is entered, it is always read into IOAREA1, which must be large enough to accomodate the desired header information.

In conjunction with optical reader input, the GET macro can be used only to retrieve records from a journal tape on a 1287.

The READ macro causes the next sequential 1287 or 1288 optical reader (document mode only) record to be read.

To accomplish document ejection and/or stacker selection and document increment functions, include the appropriate CCW(s) within the CCW list addressed by the READ macro. This technique results in increased processing throughput, and is pre-

ferable to using the CNTRL macro for document control.

The WAITF macro must be issued after the READ macro and before the program attempts to process an input record of that file. The program waits until the transfer of data is complete.

The WAITF macro accomplishes all checking for read errors on the 1287 or 1288 file and exits to your COREXIT routine for handling of these conditions, if necessary.

The RESCN macro selectively rereads a field on a document if one or more defective characters make this type of operation necessary. The field is always right-justified into the area (normally within IOAREA1) that was originally intended for this field as specified in the CCW. The macro first resets this area to binary zeros.

Note: For the 1287 models 3 and 4 and the 1288, this macro can only be used with READ BACKWARD commands. If used with READ FORWARD commands, the input area is not cleared. When 1288 unformatted fields are read, the RESCN macro should not be used.

When this macro is used in the COREXIT routine, the address of the load format CCW is obtained by subtracting 8 from the 3-byte address that is right-justified in the fullword location beginning in filename+32. (The high-order fourth byte of this fullword should be ignored.) If the RESCN macro is not used in the COREXIT routine, you must determine the load format CCW address.

When using the RESCN macro, you must ensure that the load format CCW (giving the document's coordinates for the field to be read) is command chained to the CCW used to read that field.

If the reread of the field results in a wrong-length record, incomplete read, or an unreadable character, it is indicated in filename+80.

The DSPLY macro displays the document field on the 1287 display scope. A complete field may be keyboard-entered if a 1287 read error makes this type of correction necessary. An unreadable character may be replaced by the reject character either by the operator (if processing in the on-line correction mode) or by the device (if processing in the off-line correction mode). You may then use the DSPLY macro to display the field error.

The 1287 display tube displays the full field and the operator keys in the correct field from the keyboard, if possible. The field read from the keyboard is always read into the area (normally within IOAREA1) that was originally intended for this field as specified in the CCW. The macro first resets this area to binary zeros. At completion of the operation, the data is left-justified in the area.

When the DSPLY macro is used in the COREXIT routine, the address of the load format CCW can be obtained by subtracting 8 from the 3-byte address that is right-justified in the fullword location beginning at filename+32. (The high-order fourth byte of this full word should be ignored.) If the DSPLY macro is not used in the COREXIT routine, you must determine the load format CCW address. The third parameter specifies a general-purpose register (2 through 12) into which you place the address of the load format CCW giving the coordinates of the reference mark associated with the displayed field.

The contents of filename+80 are meaningful only for X'40' (1287 scanner cannot locate the reference mark) and X'04' (wrong-length record) after the DSPLY macro is issued. Therefore, you must determine whether the operator was able to recognize the unreadable line of data.

Note: When using the DSPLY macro, you must ensure that the load format CCW is command chained to the CCW used to read that field. This provides the document coordinates for the field to be displayed.

The RDLNE macro provides selective on-line correction when processing journal tapes on the 1287 optical reader. This macro reads a line in the on-line correction mode while processing in the off-line correction mode. RDLNE should be used in the COREXIT routine only, or else the line following the one in error will be read in on-line correction mode.

If the 1287 cannot read a character, IOCS first resets the input area to binary zeros and then rereads the line containing the character which could not be read. If the read is unsuccessful, you are informed of this condition via your error correction routine (specified in DTFOR COREXIT). The RDLNE macro may then be issued to cause another attempt to read the line. If the character in the line still cannot be read, the character is displayed on the 1287 display scope. The operator keys in the correct character, if possible. If the operator cannot readily identify the defective character, he may enter the reject character in the error line. This condition is posted in filename+80 for your examination. Wrong-length records and incomplete read conditions are also posted in filename+80.

COREXIT provides an exit to your error correction routine for the 1287 or 1288. After a GET, WAITF, or CNTRL macro is executed (to increment or eject and/or stacker-select a document), an error condition causes an error condition routine to be entered with an error indication provided in filename+80. The byte at filename+80 contains the following

hexadecimal bits indicating the conditions that occurred during the last line or field read. The byte should also be tested after issuing the optical reader macros DSPLY, RESCN, RDLINE, CNTRL READKB, and CNTRL MARK. More than one error condition may be present.

Code Dec	Hex	Meaning
32	X'20'	For the 1288, reading in unformatted mode, the end-of-page (EOP) condition has been detected. Normally, on an EOP indication, the problem program ejects and stacker selects the document. After issuing one of the macros CNTRL ESD, CNTRL SSD, CNTRL EJD in your COREXIT routine, a late stacker selection condition occurred. For the 1287, a stacker select was given after the allotted elapsed time and the document was put in the reject pocket.
1	X'01'	A data check has occurred. Five read attempts for journal tape processing or three read attempts for journal tape processing were made.
2	X'02'	The operator corrected one or more characters from the keyboard (1287T) or a hopper empty condition (see HPRMTY=YES operand) has occurred (1287D).
4	X'04'	A wrong-length record condition has occurred (for journal tapes, five read attempts were made; for documents, three read attempt were made). Not applicable for undefined records.
8	X'08'	An equipment check resulted in an incomplete read (ten read attempts were made for journal tapes or three for documents). If an equipment check occurs on the first character in the record, when processing undefined journal tape records, the RECSIZE register contains zero, and the IOREG (if used) points to the rightmost position of the record in the I/O area. You should test the RECSIZE register before moving records from the work area or the I/O area.
16	X'10'	A nonrecoverable error occurred.
64	X'40'	The 1287D scanner was unable to locate the reference mark (for journal tapes, ten read attempts were made; for documents, three read attempts were made).

The byte filename+80 can be interrogated to determine the reason for entering the error correction routine. Choice of action in your error correction routine is determined by the particular application.

If you issue an I/O macro to any device other than the 1287 and/or 1288 in the COREXIT routine, you must save registers 0, 1, 14, and 15 upon entering the routine, and restore these registers before exiting. Furthermore, if I/O macros (other than the GET, WAITF, and/or READ, which cannot be used in COREXIT) are issued to the 1287 and/or 1288 in this routine, you must also save and later restore registers 14 and 15 before exiting. All exits from COREXIT should be to the address specified in register 14. This provides a return to the point from which the branch to COREXIT occurred. If the command chain bit is on in the READ CCW for which the error occurred, IOCS completes the chain upon return from the COREXIT routine.

Note: Do not issue a GET, READ, OPEN, or WAITF macro to the 1287 or 1288 in the error correction routine. Do not process records in the error correction routine. The record that caused the exit to the error routine is available for processing upon return to the mainline program. Any processing included in the error routine would be duplicated after return to the mainline program.

When processing journal tapes, a nonrecovery error (torn tape, tape jam, etc.) normally requires that the tape be completely reprocessed. In this case, your routine *must not* branch to the address in register 14 from the COREXIT routine or a program loop will occur. Instead, the routine should ignore any output resulting from the document. Following an unrecoverable error:

- the optical reader file must be closed.
- the condition causing the nonrecovery must be cleared.
- the file must be reopened before processing can continue.

If a nonrecoverable error occurs while processing documents (indicating that a jam occurred during a document incremenation operation, or a scanner control failure has occurred, of an end-of-page condition, etc.), the document should be removed either manually or by nonprocess runout. In such cases, your program should branch to read the next document.

If the 1287 or 1288 scanner is unable to locate the document reference mark, the document cannot be processed. In this case, the document must be ejected and stacker selected before attempting to read the following document or a program loop will result.

Eight binary error counters are used to accumulate totals of certain 1287 and 1288 error conditions. Each of these counters occupies four bytes, starting at filename+48. Filename is the name specified in the DTF header entry. The error counters are:

Counter and Address	Contents
1 filename+48	Equipment check (see *Note* below).
2 filename+52	Equipment check uncorrectable after ten read attempts for journal tapes or three read attempts for documents (see *Note* below).
3 filename+56	Wrong-length records (not applicable for undefined records).
4 filename+60	Wrong-length records uncorrectable after five read attempts for journal tapes or three read attempts for documents (not applicable for undefined records).

5 filename+64	Keyboard corrections (journal tape only).
6 filename+68	Journal tape lines (including retried lines) or document fields (including retried fields) in which data checks are present.
7 filename+72	Lines marked (journal tape only).
8 filename+76	Count of total lines read from journal tape or the number of CCW chains executing during document processing.

Note: Counters 1 and 2 apply to equipment checks that result from incomplete reads or from the inability of the 1287 or 1288 scanner to locate a reference mark (when processing documents only).

All previous counters contain binary zeros at the start of each job step. You may list the contents of these counters for analysis at end of file, or at end of job, or you may ignore the counters. The binary contents of the counters should be converted to a printable format.

Optical Character Reader (IBM 3886)

The IBM 3886 Optical Character Reader can be operated in any partition. You must access all operations through LIOCS macros or PIOCS. In problem programs, the device is controlled by means of the assembler language only, at the LIOCS READ/WAITF level.

Two steps are required to use the 3886 as an input device. In one assembly, you must define the documents to be read. Then, in the problem program, you issue the instructions to process the documents. You use the DTFDR macro to define the characteristics of the 3886 file in your problem program, to describe the format record to be loaded into the 3886 when the file is opened, and to specify the storage areas and routines to be used. The DRMOD macro generates the logic module to process the file.

Defining Documents: Two macros are provided for defining documents. One, the DFR macro, defines attributes common to a group of line types. The other, the DLINT macro defines specific attributes of an individual line type. As many as 27 DLINT macros can be associated with one DFR macro as long as the number of line types plus the number of fields is less than or equal to 53.

The DFR and associated DLINT macros are used in one assembly to build a format record. Only one DFR with its associated DLINT macros may be specified in each assembly. The DFR is link-edited into the core image library so that it can be loaded into the 3886 when the field is to be processed. A format record contains information about the documents being read, each individual line on the document, and each field in the line. This information is used to read the line and edit the data before it is passed to the problem program.

When opening a 3886 optical reader file, OPEN loads the appropriate format records (as specified in the DTFDR) into the 3886 control unit.

Document Control and Marking: 3886 support also provides for

- changing format records
- ejecting and stacker selecting documents
- performing timing and mark checks
- line and page marking documents.

All format records are created in separate assemblies; they must be cataloged in the core image library before they can be used for processing documents.

You can change format records during program execution by using the SETDEV macro. It loads a new format record into the 3886 and returns control to the next sequential instruction in your program. If the operation is not successful, control is passed to your COREXIT routine, or the job is canceled. If you issue SETDEV macro when no documents remain to be processed and the end-of-file has been pressed on the device, control is passed to the end-of-file routine.

To perform the other control and marking functions, you can use the CNTRL macro. When the operation is successful, control returns to the next instruction in your program; otherwise, control passes to the COREXIT routine or to the end-of-file routine.

Reading Data Records: Each time a READ macro is issued, one line of data is supplied to your program. Each line read is considered to be a data record. A header record is provided to your program with each data record: it is 20 bytes in EBCDIC and contains the number of the line scanned, the number of times the line was scanned, and other important information about the line and its fields.

The data record passed to your program is a fixed-length record containing up to 130 bytes of data. You specify its length in the DTFDR macro. The data record is in one of two formats as follows:

1. If standard mode is specified (IMAGE=NO in the DLINT macro), the data record contains the EBCDIC character codes for the line of data after the editing functions have been performed. The editing functions are specified in the DLINT macro.
2. If image mode is specified (IMAGE=YES in the DLINT macro), the data record contains two types of information: field length and data from the document. The first 28 bytes contain

14 two-byte entries that indicate the length of each field in the record. If the number of fields is less than 14, the entries for the rightmost unused fields contain EBCDIC zero (X'F0F0'). The data read from the document follows the field length entries, beginning in the 29th byte.

If the number of characters in the data record is less than the space allowed for the input record, the unused rightmost portion of the record is padded with blanks (X'40').

The WAITF macro is used to ensure that an I/O operation is completed before the execution continues. If the operation is not completed when the WAITF macro is issued, the active partition is placed in a wait condition until the I/O operation is completed. The completed operation is then tested for errors. If no errors are detected, control is returned to the next instruction in your program.

Error Handling: If an error occurs during the I/O operation, control is passed to the COREXIT routine. If an I/O operation is requested, no more documents are available, and the end-of-file key has been pressed, control is given to the end-of-file routine.

LIOCS branches to the COREXIT routine whenever an error is indicated in the EXITIND byte. The COREXIT routine and EXITIND are both specified by operands in the DTFDR macro.

EXITIND=name specifies the symbolic name of the 1-byte area in which the completion code is returned to the COREXIT routine for error handling from an I/O operation.

The completion codes are:

Code		Meaning
Dec	Hex	
240	X'F0'	No errors occurred. (This code should not be present when the COREXIT routine receives control.)
241	X'F1'	Line mark station timing mark check error.
242	X'F2'	Nonrecovery error. Do not issue the CNTRL macro to eject the document from the machine. Have the operator remove the document.
243	X'F3'	Incomplete scan.
244	X'F4'	Line mark station timing mark check and equipment check.
249	X'F9'	Permanent error.

Note: If any of these errors occur while the file is being opened, the COREXIT routine does not receive control and the job is canceled.

You can attempt to recover from various errors that occur on the 3886 through the COREXIT routine you provide. Your COREXIT routine receives control whenever one of the following conditions occurs:

- Incomplete scan
- Line mark station timing mark check error
- Nonrecovery error
- Permanent error.

Note: If any of these errors occur while the file is being opened, the COREXIT routine does not receive control and the job is canceled.

Figure 7-21 describes normal functions for the COREXIT routine for the various error conditions and provides the exits that must be taken from the COREXIT routine.

Each time an imperative macro (except OPEN, LBRET, SETL, or SETFL) is issued using a particular DTF, register 13 must contain the address of the save area associated with that DTF. The fact that the save areas are unique or different for each task makes the module reentrant (that is, capable of being used concurrently by several tasks). For more information see "Shared Modules and Files" in the "Multitasking Functions" section.

If a COREXIT routine issues I/O macros using the same read-only module that caused control to pass to either routine, your program must provide another save area. One save area is used for the normal I/O operations, and the second for I/O operations in the COREXIT routine. Before returning to the module that entered the COREXIT routine, register 13 must contain the save area address originally specified for that DTF.

Assembling a Format Record for the 3886 Optical Character Reader: This section describes a use for the IBM 3886 Optical Character Reader. Included are a sample document, a format record assembly and the data provided by the 3886.

A typical application for an optical character reader is processing insurance premiums. Figure 7-22 shows an insurance premium notice for the Standardacme Life Insurance Company. The document has three lines of data to be read (see Figure 7-24 for sample data). The first line contains one field, the name of the policy holder. The second line contains four fields: the policy holder's address, the policy number, the premium amount due, and a code to be hand-printed if the amount paid is different from the amount due. The third line contains one field that contains the amount paid if different from the amount due.

To process documents like that in Figure 7-22, one format record is used. The format record must be created in a separate assembly. The coding necessary to create the format record is shown in Figure

Error	Normal COREXIT Function	Exit to
X'F2'	Eliminate the data that has been read from this document and prepare to read the next input document. (See Note 1).	Routine in your program to read the next document.
X'F4' or X'F9'	Do whatever processing is necessary before the job is canceled. (See Note 1).	Your end-of-job routine.
X'F1'	Do any processing that may be required. The document may have been read incorrectly; you may want to delete all data records from the document. (See Note 2).	Branch to the address in register 14 to return to the instruction following the macro causing the error.
X'F3'	Rescan the line using another format record or using image processing and editing the record in your program (see Note 2).	Branch to the address in register 14 to return to the instruction following the macro causing the error.

Note 1: If, in your COREXIT routine, you issue an I/O macro to the 3886 and an error occurs during that operation, control is returned to the beginning of the COREXIT routine. You must take precautions in the COREXIT routine to prevent looping in this situation. If no errors occur, control returns to the instruction following the I/O macro.
Note 2: If, in your COREXIT routine, you issue an I/O macro to the 3886, control always returns to the instruction following the macro. You should then check the completion code to determine the outcome of the operation.

Figure 7-21. COREXIT routine functions.

STANDARDACME LIFE
INSURANCE COMPANY

NOTICE OF PAYMENT DUE

DUE DATE MO	DAY	YR	ANNIV MONTH	DIST NO	PREMIUM
06	23	72	07	45	249.75

H

DALE E. STUEMKE

1

1363 SE 10TH AVE.

ROCHESTER, MINN 58395404 249.75

POLICY NUMBER $ AMOUNT DUE

INSURED DAWN STUEMKE

If your address is other than shown, please notify the Company. Please make check or money order payable to Standardacme Life and present with notice to your Company Representative or to

PLEASE RETURN WITH YOUR PAYMENT

FOR COMPANY USE ONLY

Figure 7-22. Premium notice example.

7-23. The numbers at the left of the coding form correspond to those in the following text.

1. The job control statements indicate that the job is an assembly. The output of the assembly is to be cataloged as phase FORMAT.

2. The DFR macro specifies the characteristics common to all lines on the document:

   FONT=ANA1: The alphameric OCR-A font is used for reading any fields that do not have another font specified in the DLINT macro field entries.

   REJECT=@: The commercial at sign @ is substituted for any reject characters encountered.

   EDCHAR=(',',.): The comma and period are removed from one or more fields as indicated in DLINT entries (line 2, field 3).

3. The DLINT macro describes one line type in a format record described by the DFR macro.

   The following information is provided about the *first* line:

   LFR=1,LINBEG=4: The first line on the document has a line format record number of 1. The first field read from the line begins four-tenths of an inch (10.16 mm) from the left edge of the document. The data record is in standard mode; editing is performed on the field.

```
// JOB FORMAT
// OPTION CATAL
1  PHASE FORMAT,+0,NOAUTO
// EXEC ASSEMBLY
         TITLE 'DOCLIST-FORMAT'
         START
****************************************
* THIS ASSEMBLY WILL CREATE A FORMAT   *
* RECORD DESCRIBING AN INSURANCE       *
* PREMIUM NOTICE                       *
****************************************
2        DFR   FONT=ANA1,REJECT=@,          X
               EDCHAR=(',',.)
3        DLINT LFR=1,LINBEG=4,              X
               FLD1=(32,20,NCRIT),          X
               EDIT1=HLBLOF
4        DLINT LFR=2,LINBEG=4,              X
               FLD1=(30,20,NCRIT),          X
               EDIT1=HLBLOF,                X
               FLD2=(42,8),                 X
               EDIT2=ALBNOF,                X
               FLD3=(54,6),                 X
               EDIT3=(HLBHIF,EDCHAR),       X
               FLD4=(62,1,NHP1),            X
               EDIT4=ALBHIF
5        DLINT LFR=3,LINBEG=45,             X
               FLD1=(64,7,NHP1),            X
               EDIT1=ALBHIF,                X
               FREND=YES
         END
/*
// EXEC LNKEDT
/*
```

Figure 7-23. Format record assembly example.

FLD1=(32,20,NCRIT),EDIT1=HLBLOF: The first and only field on the line ends 3.2 inches (81.28 mm) from the left edge of the document, the edited data is placed in a 20-character field. The field is not considered critical. All leading and trailing blanks are removed, the data should be left-justified, and the field is padded to the right with blanks.

4. The *second* line on the document is described as follows:

   LFR=2,LINBEG=4: The second line of the document has a line format record number of 2. The first field read begins four tenths of an inch (10.16 mm) from the left edge of the document. The data record is in standard mode; editing is performed on all fields on the line.

   FLD1=(30,20,NCRIT),EDIT1=HLBLOF: The first field on the line ends 3.0 inches (76.2 mm) from the left edge of the document, the edited data is placed in a 20-byte field. The field is not considered critical. All leading and trailing blanks are removed, the data is left-justified, and the field is padded to the right with blanks.

   FLD2=(42,8),EDIT2=ALBNOF: The second field ends 4.2 inches (106.68 mm) from the left edge of the document, the edited data is placed in an eight-byte field, the field is critical. All leading and trailing blanks are removed from the field. The resulting field must be eight digits in length or a wrong-length field indicator is set.

   FLD3=(54,6),EDIT3=(HLBHIF,EDCHAR): The third field ends 5.4 inches (137.16 mm) from the left edge of the document, the edited data is placed in a six-byte field, the field is critical. All leading and trailing blanks are removed, the data is right-justified, and the field is padded to the left with zeros. A comma, if present, and the decimal point are removed from the edited field.

   FLD4=(62,1,NHP1),EDIT4=ALBHIF: The fourth field ends 6.2 inches (157.48 mm) from the left edge of the document, the edited data is placed in a one-byte field, the field is critical and is read using the numeric handprinting normal mode. All blanks are removed, and data is right-justified, and the field is padded to the left with zeros.

5. The *third* line on the document is described as follows:

   LFR=3,LINBEG=45: The third line on the document has a line format record number of 3. The field to be read begins 4.5 inches (114.3 mm) from the left edge of the document. The data record is in standard mode; editing is performed.

   FLD1=(63,7,NHP1),EDIT1=ALBHIF: The field on this line ends 6.3 inches (160.02 mm) from the left edge of the document, the edited data is placed in a seven-byte field, the field is critical and is read using the numeric handprinting normal mode. All blanks are removed, the data is right-justified, and the field is padded to the left with zeros.

   FREND=YES: This is the format record end. No DLINT macros follow this statement.

Optical Mark Reader (IBM 3881)

In general, remarks in the earlier section that applied to processing card files, using DTFCD for file definition, also apply to 3881 files. Some programming considerations for the 3881 and exceptions to the DTFCD general remarks follow.

Block size, or, the I/O area size, for 3881 files must be specified by the DTFCD operand BLKSIZE. It must be sufficient to contain:

- Six bytes of record description information

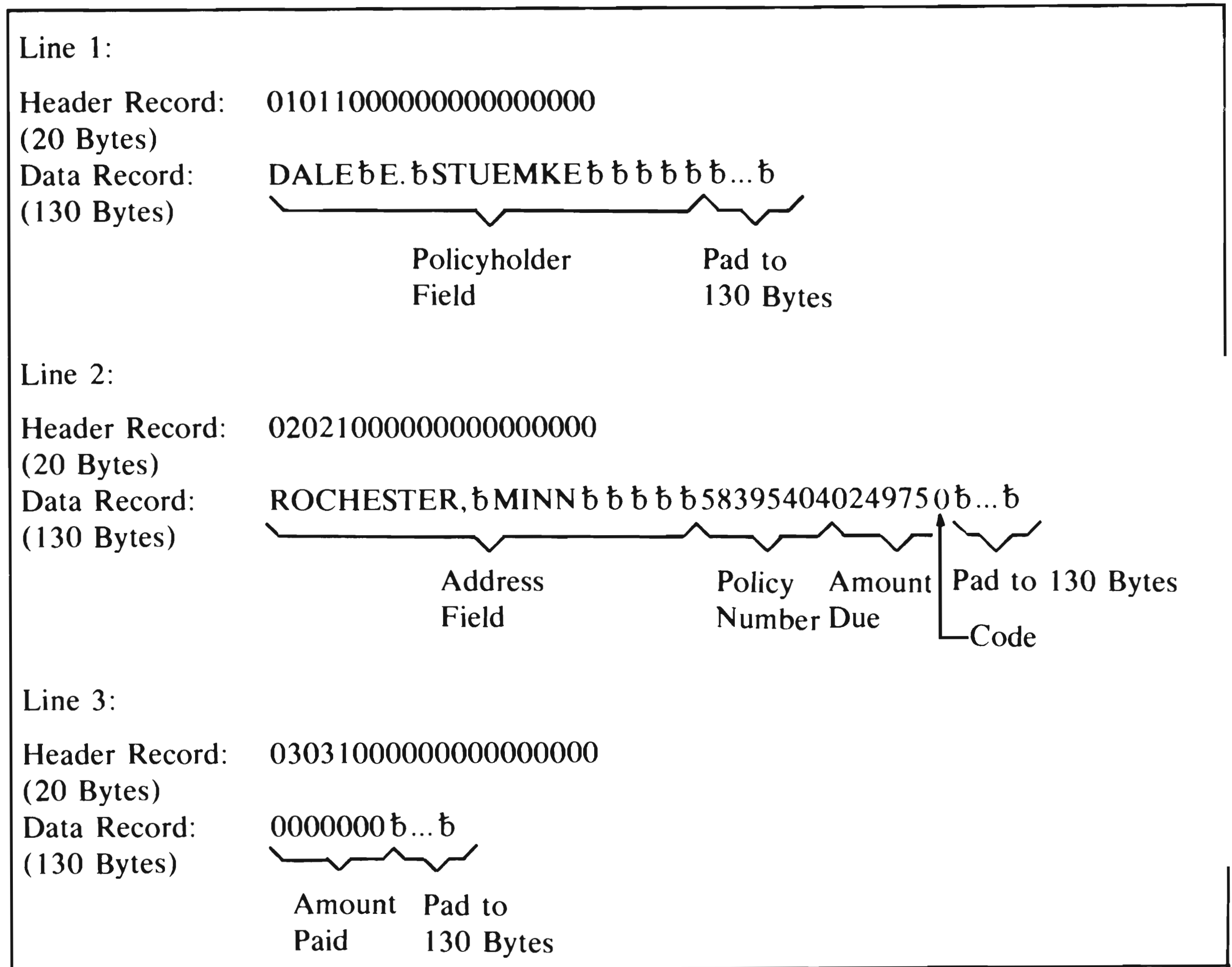


Figure 7-24. Sample data.

- Mark read data
- BCD (binary coded decimal) mark read data if the BCD feature is being used.
- 7 bytes of serial number and batch number data if the serial number feature is being used.

The BLKSIZE operand for the 3881 cannot exceed 900. If specified greater than 900, BLKSIZE defaults to 900. If the BLKSIZE operand is omitted, 900 is assumed.

A device address (DEVADDR operand) of SYSIPT, SYSPCH, or SYSRDR must not be specified.

Use of a work area is not permitted with the 3881; that is, the WORKA=YES operand is forbidden.

Only fixed, unblocked, input records are valid for the 3881.

Processing Paper Tape Files

Before processing paper tape files, they must be defined by means of the DTFPT macro and the PTMOD logic module generation macro. The operands for these macros are listed in Figure 7-25. For details of these macros, see *VSE/Advanced Functions Macro Reference.*

Note that not all of the DTFPT operands are valid for PTMOD; Figure 7-26 shows operands valid for the various paper tape handling devices.

Programming Considerations for Paper Tape

Paper tape I/O routines can only be programmed in assembler language. You specify the characteristics of the file in the DTFPT declarative macro and use GET and PUT macros to perform input and output. Two record formats are acceptable: fixed-length unblocked and undefined. The record format specified does not necessarily apply to the physical for-

Applies to					
Input	Output	DTFPT	PTMOD		
X	X	M		BLKSIZE=n	Length of your I/O areas.
X	X	M		DEVADDR=SYSnnn	Symbolic unit to be associated with this file.
X	X	M		IOAREA1=xxxxxxxx	Name of first I/O area.
	X	O		DELCHAR=x'nn'	Delete character.
X	X	O	O	DEVICE=nnnn	(2671, 1017, 1018). If omitted, 2671 is assumed.
X		O		EOFADDR=xxxxxxxx	Name of your end-of-file routine.
	X	O		EORCHAR=x'nn'	End-of-record character. (For RECFORM=UNDEF).
X	X	O		ERROPT=xxxxxxxx	(IGNORE, SKIP, or error routine name). Prevents job termination on error records.
	X	O		FSCAN=xxxxxxxx	(For shifted codes). Name of your scan table used to select figure groups.
X		O		FTRANS=xxxxxxxx	(For shifted codes). Symbolic address of your figure shift translate table.
X	X	O		IOAREA2=xxxxxxxx	Name of second I/O area.
X	X	O		IOREG=(nn)	Used with two I/O areas. Register (2-12) containing current record address.
	X	O		LSCAN=xxxxxxxx	(For shifted codes). Name of your scan table used to select letter groups.
X		O		LTRANS=xxxxxxxx	(For shifted codes). Name of your letter shift translate table.
X	X	O		MODNAME=xxxxxxxx	For module names other than standard.
X	X	O		OVBLKSZ=n	Used if I/O records are compressed or expanded.
X	X	O	O	RECFORM=xxxxxx	(FIXUNB or UNDEF). If omitted, FIXUNB is assumed.
X	X	O		RECSIZE=(nn)	Register containing the record length.
X		O	O	SCAN=xxxxxxxx	Name of your scan table for shift or delete character.
X	X	O	O	SEPASMB=YES	DTF is assembled separately.
X	X	O	O	TRANS=xxxxxxxx	Name of your table for code translation.
X		O		WLRERR=xxxxxxxx	Name of wrong-length-record error routine.

M = Mandatory
O = Optional

Figure 7-25. DTFPT and PTMOD macro operands.

mat of the data on the paper tape, but to the format of the logical record as it appears in the I/O area. The physical data may have characters embedded that must be deleted, such as delete characters and shift codes.

Paper Tape Input

Data read from paper tape may physically be in any paper tape code the user requires. Logical data in virtual storage is expected to be in internal IBM code (EBCDIC). If some code or shifted code (figure shift and letter shift) must be translated, this can be done automatically.

After a GET has been issued, a logical record is obtained from physical data. During this process, the delete characters and the shift characters are removed from the data. The data that follows such characters is automatically shifted to the left. Attention must be given to the problem of synchronizing the data fields with the program. The paper tape is read character by character, and all characters are placed in subsequent character locations in the input area. If, in some data field, one character too many or too few is specified, all following fields will be out of phase. Therefore one usually adds extra characters with a special bit configuration to the data. These characters are expected to occur in each record in the same character locations. By checking these locations in his program, the user can identify incorrectly formatted records.

Another method of checking whether data that is processed is valid, is to expand data fields with an additional check character which is the result of some calculation, and later repeat this calculation in the program. For example, for numeric data fields you may add all characters on even locations, do the same for all characters in uneven locations, multiply the two sums, and then use the last character of the product as a check character. A data field with the content 85318 would then be represented on paper tape as 853184, the calculation being 8 + 3 + 8 = 19, 5 + 1 = 6, 19 x 6 = 114.

Operand*				Resulting Module
DEVICE=	RECFORM=	SCAN=	TRANS=	
2671**	FIXUNB**			Does not handle translation or shift or delete characters.
2671**	FIXUNB**		YES	Handles translation of unshifted codes, but not delete characters.
2671**	FIXUNB**	YES		Handles shift and delete characters for records of fixed unblocked format
2671**	UNDEF	YES		Handles shift and delete characters for records of undefined format.
1017	FIXUNB**			Does not handle translation or shift or delete characters.
1017	FIXUNB**		YES	Handles translation of unshifted codes, but no delete characters.
1017	FIXUNB**	YES		Handles shift and delete characters for records of fixed unblocked format.
1017	UNDEF	YES		Handles shift and delete characters for records of undefined format.
1018	FIXUNB**		YES	Handles translation of unshifted codes, if specified in DTFPT, for records of fixed unblocked format.
1018	FIXUNB**			
1018	UNDEF			Handles translation of unshifted codes, if specified in DTFPT, for records of undefined format.
1018	UNDEF		YES	
1018	FIXUNB**	YES		Handles shift characters for records of fixed unblocked format.
1018	UNDEF	YES		Handles shift characters for records of undefined format.

* In all cases, SEPASMB=YES may either be specified or omitted.
** Specified explicitly or by default.

Figure 7-26. PTMOD operand combinations.

Undefined Record Format on Input: Each record must be followed by an end-of-record character, which is specified in a translation table set up by the user. The input area must be at least one position longer than the longest record anticipated, including the EOR character and any delete characters and shift codes embedded in the data. If an input area is filled completely, the record is assumed to be too long, and the wrong-length record routine of IOCS will become active. After a GET, a read is performed, count-controlled by the BLKSIZE operand (length of the I/O area). Reading stops when an EOR character is sensed.

After data has been read up to an EOR character, the delete and shift characters are removed by the translation process. The translated and compressed record is presented to the user; its length is communicated in a register. Consecutive EOR characters are skipped; the system will never return a data length of zero.

Fixed-Length Unblocked Format on Input: The term 'fixed-length' applies to the format of the logical record in the input area after it has been translated (if necessary) and compressed. It does not apply to the format of the data as it appears physically on the paper tape. A paper tape file consistes of one continuous string of data characters, and it is IOCS that establishes boundaries between the records by means of the BLKSIZE operands specified by the user. The physical data may have embedded delete characters and shift codes. It may therefore be necessary to read more characters than the size of the logical record seems to indicate. The number of characters that must be read is specified by the OVBLKSZ operand in the DTFPT macro.

After a GET, IOCS starts a count-controlled read until the input area contains the number of characters specified in OVBLKSZ. Then the translation process starts, eliminating shift codes and delete characters from the data. If the resulting record is shorter than BLKSIZE, additional reads are performed until IOCS has obtained a logical record with a size equal to BLKSIZE. As a result, some characters may be read which belong to the next logical record. These characters are moved to the beginning of the input area when the next GET is issued. Therefore, do not clear the input area beyond the size of the logical record as defined in BLKSIZE. If you do, you will destroy part of the following logical record.

Undefined Format versus Fixed-Length Format: The main difference between the processing of the two formats is that IOCS can recognize record

boundaries of undefined records, but not of fixed-length records. An incorrect specification of the number of characters is more serious with fixed-length records than with undefined records. With undefined records, only one record would be out of phase, whereas with fixed-length records all records following the wrong record would be out of phase. Therefore, it is usually better to use the undefined record format, even if the logical records have a fixed-length format. When you do this, make sure that the input area is large enough to contain all physical data of one record.

Code Translation on Input: The TRANS operand in the DTFPT macro is used for the translation of non-shifted code directly into internal IBM code (EBCDIC). If the input is in EBCDIC, no translation is required, and the TRANS operand may be omitted. The SCAN operand may be used alone or in conjunction with TRANS to delete characters from records that do not contain shifted code.

If the input contains shifted code, the FTRANS, LTRANS, and SCAN operands must be specified in the DTFPT macro. Translation of shifted code is accomplished by IOCS as follows:

1. The data is scanned for shift characters. The segments between shift characters are translated, using the appropriate shift table.
2. The translated segments are moved to the left to remove the shift characters.
3. Steps 1 and 2 are repeated for each segment until the complete record has been translated and compressed.

These steps result in a translated and compressed record, left-justified in the input area. The record length is communicated to the user in a register, which is designated in the RECSIZE operand.

The EOR character at the end of undefined records must be shift-independent. That is, it must be effective whether the coding be in letter shift or figure shift. If there is valid code in either shift that corresponds to the coding of the EOR character established for a particular job, then this shift code must not be included in the input.

IOCS assumes that the first record read from paper tape starts with figure shift coding. Therefore, if the first record starts with letter shift code, you must make sure that the first character in the first physical block is a letter shift character. The shift status is carried from one record to the next and remains unchanged until another shift character is encountered.

EOF Condition (Input Only): The EOF condition occurs with an end-of-tape condition when the EOF switch is on. When IOCS detects this EOF condition (unit exception flag on in the first CSW status byte), it automatically branches to your end-of-file routine. However, at the end of the routine, you can choose to return to IOCS to read a new tape by branching to the address in register 14. If any IOCS macro is contained in this routine, the contents of register 14 must be saved and restored.

If an end-of-tape condition is detected while reading characters other than blanks or deletes (all punched holes), the unit check bit in the first CSW status byte is set on. This applies only to the 1017, and causes the broken tape bit (bit 7) to appear in the sense byte. The broken tape condition may occur in addition to the EOF condition if the EOF switch is on.

Trailer Length (Input Only): To avoid a broken tape condition that would result if the tape trailer is too short, make sure that the length of the trailer is as shown below:

- For undefined records (read-EOR command): longer than two inches (5.08 cm).
- For fixed unblocked records (read command): longer than

  $\frac{\text{byte count}}{10}$ + 2 inches, or 0.254 (byte count) +5.08 cm

Note: Byte count is either the count specified in BLKSIZE (record without shifted codes or records with shifted codes but without using the OVBLKSZ operand in the DTFPT), or the count specified in OVBLKSZ (records with shifted codes using the OVBLKSZ operand).

However, when your program processes undefined records, and a trailer longer than

$\frac{\text{byte count}}{10}$ + 2 inches, or 0.254 (byte count) +5.08 cm

is read, this trailer will be mistaken for a wrong-length record.

Paper Tape Output

Data may be written in any code the user requires. Translation of shifted and non-shifted code can be done automatically.

After a PUT has been issued, the logical record is expanded by IOCS during the translation process, if required, and shift characters are added when necessary.

Undefined Record Format on Output: You specify the end-of-record character in the EORCHAR operand of the DTFPT macro. IOCS writes this character after the last character in each record. The size of

the output area (specified in the BLKSIZE operand) must be at least equal to that of the longest record anticipated, including all shift characters that are to be inserted.

Fixed-Length Record Format on Output: The output area must be the same size as each logical record before translation and insertion of shift characters. Its length is specified in the BLKSIZE operand. Logical records are translated by IOCS if necessary. As a result of PUT, a count-controlled write causes the specified number of characters to be written. For shifted-code files, the records are expanded by IOCS with shift characters. IOCS performs additional writes to construct the required physical block.

Code Translation on Output: The TRANS operand in the DTFPT macro is used for translation on non-shifted code, from internal IBM code (EBCDIC) into any other code required by the user. If the output is to be punched in EBCDIC, no translation is required, and the TRANS operand may be omitted.

If the output contains shifted code, the TRANS, FSCAN, and LSCAN operands must be included in the DTFPT macro. The OVBLKSZ operand may be used or omitted. If omitted, the records are written segment by segment, and IOCS adds a shift character in front of each segment. If OVBLKSZ is used, however, the segments are moved to the right, while shift characters are inserted in the data before each segment. If OVBLKSZ is specified too low, additional writes are performed to produce the physical data.

I/O areas

The BLKSIZE operand specifies the length of the input or output area. The maximum block size is 32,767 bytes.

Input Area: For undefined records, this area must be at least one byte larger than the longest record including all shift and delete characters included in the record. For fixed-length records, this area must be the same size as the record. If shift and delete characters are included in the record (the SCAN entry is specified), BLKSIZE indicates the number of characters required by the program after translation and compression. OVBLKSZ contains the number of characters to be read in to produce the BLKSIZE number.

Output Area: For undefined records, the area must be at least as large as the longest record, including all shift characters that are to be included in the record. For fixed-length records, the area must be the same size as the record. For shifted codes (when the FSCAN and LSCAN entries are specified), BLKSIZE must contain the number of characters after translation and insertion of shift characters.

Error Conditions

The paper tape reader or punch stops immediately on an error condition. If the error cannot be corrected and the job is not terminated, IOCS causes the entire record containing the error to be:

- Translated and compressed, if needed (for input records).
- Translated, expanded, and punched, before taking the error option specified by the problem program (for output records).

Wrong Length: For input file, the only wrong-length condition that can be detected is an over-length undefined record. This should be reflected in the BLKSIZE operand.

When IOCS finds a wrong-length record, it branches to the symbolic name specified in the WLRERR entry. If this entry is omitted and the ERROPT entry is included, IOCS considers the error uncorrectable and uses the ERROPT option specified. Absence of both ERROPT and WLRERR entries causes the wrong-length record to be accepted as normal records. Wrong-length checking is not performed for fixed-length records because a fixed number of characters is read in each time. IOCS detects over-length undefined records when the incoming record fills the input area. The input area must, therefore, be at least one point longer than the longest record anticipated.

At the end of the WLRERR routine, return to IOCS by branching to the address in register 14. The next IOCS read operation will normally cause the remainder of the overlength, undefined record to be read. If any other IOCS macros are included in the record-length error routine, the contents of register 14 must be saved and restored.

Note: A wrong-length condition appears during the first read operation on a 1017 if the combined length of the tape reader and the first record is greater than the length of the longest record anticipated (the length specified in BLKSIZE).

Wrong-length record indication is impossible with fixed unblocked records, because each record is a sequence of a specified number of characters. Use the FIXUNB record format carefully, because specifying one character too few or too many in any record causes all subsequent records to be out of phase. The problem program should specify the RECSIZE operand to check for the correct length of the last record of any file. A record must be entirely on one reel of input tape.

Data Check: Figure 7-27 shows the decision taken by LIOCS, or possible operator actions, after an unrecoverable data check occurs.

Type of Record Processed	Input Operation	Output Operation
Fixed unblocked record in shifted code	Action 1	Action 1
Fixed unblocked record in nonshifted code	Action 2	Action 2
Undefined record in non-shifted code	Action 2	Action 2
Undefined record in shifted code	Action 2	Action 1

Action 1: The system automatically cancels the job.

Action 2: The operator may choose to:
- cancel the job
- ignore the error
- retry the operation (for 2671 only).

Figure 7-27. LIOCS decision on a paper tape data check.

Following an ignore decision, logical IOCS takes action in accordance with the parameter specified in ERROPT. If

ERROPT=IGNORE:

The record is handled as if no errors were detected.

ERROPT=SKIP:

The erroneous record is skipped and the next record is read in.

ERROPT=name:

The record is handled as if no error were detected, and control is given to your error routine. At the end of this routine, return to IOCS by branching to the address in register 14. The next record is then read in or written out.

If ERROPT was not specified, the job is canceled.

Notes:

1. The character in error is repunched preceded by its corresponding shift character:
   - For output records expressed in a paper tape code where the delete character and one of the shift characters have the same configuration.
   - Following a data check.
2. The entire erroneous record is repunched as if no errors were detected:
   - If an irrecoverable error occurs and ERROPT=name or ERROPT=IGNORE was specified in the DTFPT.
   - In the case of output records with two I/O areas, the CLOSE macro checks the successful completion of the last operation.
3. No error condition occurs on the 1018 if the setting of the tape width selector does not match the tape code specified in the problem program.
4. When reading paper tape with physical IOCS, restore the CCW address in the CCB before using the EXCP macro.

Chapter 8: Processing Device-Independent System Files

Device independence allows you to program as if a certain device were always available. When the program is actually run and the device happens not to be available, the symbolic device name can be assigned easily to some other device. In some cases, the other device may even be of a different type.

The DTFDI macro provides device independence for system logical units. If several DTFDI macros are assembled within one program and all of them have the same RDONLY condition, only one logic module (DIMOD) is required. Therefore, DTFDI processing requires fewer parameters and less storage than device dependent LIOCS macros.

If you are using a DASD device or advanced printer buffering on an IBM 3800 Printing Subsystem, you do not need to specify DIMOD. Support for DASD devices and for the advanced printer buffering includes pre-assembled logic modules that automatically are loaded into the SVA (system virtual area) at IPL time and are linked to the problem program when the assigned file is opened. To maintain device independence, however, you may choose to include a DIMOD specification in your program if, when you write the program, you are not certain which device will be assigned to the file at execution time. When the file is opened, the OPEN routines for DASD devices or the 3800 will override the DIMOD linkage if the file is assigned to either a DASD device or a 3800.

The DTFDI macro should always be used to read SYSIPT data if the program might be invoked by a catalogued procedure.

The restrictions on DTFDI processing are:

- Only fixed unblocked records are supported.
- Only forward reading is allowed.
- In a multivolume diskette file, new volumes are fed automatically.
- The last volume of a multivolume diskette *output* file will be ejected automatically, but the last volume of a multivolume diskette *input* file will not.
- If DTFDI is used with diskettes, special records (deleted or sequentially relocated records) on input files are skipped and not passed to the user.
- Rewind options are not provided, that is, no repositioning is done at OPEN and CLOSE time.
- Combined file processing is not supported for reader-punches.
- Reading of cards is restricted to the first 80 bytes per card.
- The CNTRL and PRTOV macros cannot be used with this macro.
- Reading, writing, or checking of standard or user-standard labels for tape/disk is not supported.
- If ASA control character code is used in a multitasking environment and more than one DTF uses the same module with RDONLY=YES, overprinting may occur.
- If a DASD device is assigned to a system logical unit and ERROPT or WLRERR specified, no LIOCS macros other than ERET may be issued within the user-written error handling routine.

The operands for DTFDI are listed in Figure 8-1. For more details about DTFDI and the logic module generation macro, DIMOD, see *VSE/Advanced Functions Macro Reference.*

Record Size

For input files, (SYSIPT and SYSRDR), the maximum allowable record size is 80 bytes. For output files, the record must include one byte for a control character. The maximum record size for SYSLST is 121 bytes and 81 bytes for SYSPCH. For printers and punches, the logic module assumes a S/370-type control character if the character is not a valid ASA character. The program checks ASA control characters before S/370-type control characters. Therefore, if it is a valid ASA control character (even though it may also be a S/370-type control character), it is used as an ASA control character. Otherwise, it is used as a S/370-type control character.

Control character codes are listed in Appendix A, except for the following:

- 2520 stacker selection codes must be used for the 1442.
- 2540 stacker selection 3 must not be used if device independence is to be maintained.

The record size is specified by the RECSIZE operand. If this operand is omitted, the following is assumed:

80 bytes for SYSIPT.
80 bytes for SYSRDR.
81 bytes for SYSPCH.
121 bytes for SYSLST.

The use of assumed values for the RECSIZE operand assures device independence. For disk and

M	DEVADDR=SYSxxx	SYSIPT, SYSLST, SYSPCH, or SYSRDR. System logical unit.
M	IOAREA1=xxxxxxxx	Name of first I/O area.
O	CISIZE=nnnnn	Size of FBA DASD Control Interval.
O	EOFADDR=xxxxxxxx	Name of your end-of-file routine.
O	ERROPT=xxxxxxxx	IGNORE, SKIP, or name of your error routine. Prevents termination on errors.
O	FBA=YES	Specifies a Fixed Block Architecture DASD file.
O	IOAREA2=xxxxxxxx	If two I/O areas are used, name of second area.
O	IOREG=(nn)	General registers 2-12, written in parentheses. If omitted and two I/O areas are used, register 2 is assumed.
O	MODNAME=xxxxxxxx	DIMOD name for this DTF. If omitted, IOCS generates a standard name. Not needed with DASD or 3800 advanced printer buffering.
O	RDONLY=YES	Generates a read-only module. Requires a module save area for each task using the module.
O	RECSIZE=nnn	Number of characters in record. Assumed values: 121(SYSLST), 81(SYSPCH), 80 (otherwise).
O	SEPASMB=YES	DTFDI to be assembled separately.
O	TRC=YES	For 3800, output data lines include Table Reference Character.
O	WLRERR=xxxxxxxx	Name of your wrong length record routine.

M = Mandatory
O = Optional

Figure 8-1. DTFDI macro operands.

diskette files, the assumed values are required to assure device independence.

Error Handling

By means of two DTFDI operands, ERROPT and WLRERR, IOCS assists you in processing I/O and record-length errors. The WLRERR operand applies only to input files on devices other than diskette units. It specifies the name of your routine to which IOCS branches if a wrong-length record is read on a tape or disk device.

Because only fixed-length records are allowed, a wrong-length record error condition results when the length of the record read is not equal to that specified in the RECSIZE operand. If the length of the record is less then that specified in the RECSIZE operand, the first two bytes of the CCB (first 16 bytes of the DTF) contain the number of bytes left to be read (residual count). If the length of the record to be read is larger than that specified in the RECSIZE operand, the residual count is set to zero and there is no way to compute its size. The number of bytes transferred is equal to the value of the RECSIZE operand, and the remainder of the record is truncated.

The address of the record is supplied in register 1. In your routine, you can perform any operation except issuing another GET for this file. Also if you use any other IOCS macros in your routine for a file assigned to a DASD, you must save the contents of register 14. If RDONLY=YES, you must save the contents of register 13 as well. For a file assigned to a DASD, use of a LIOCS macro other than ERET will cause the task to be terminated (for the file in error).

At the end of the routine, you must return to IOCS by branching to the address in register 14. When control returns to your program, the next record is made available. If this operand is omitted but a wrong-length record is detected by IOCS, the action depends on whether the ERROPT operand is included:

- If the ERROPT operand (always assumed for DASD) is included, the wrong-length error record is treated as an error record and handled according to the ERROPT parameter.
- If the ERROPT operand is omitted, IOCS ignores wrong-length errors and the record is made available to you. If, in addition to a wrong-length record error, an unrecoverable parity error occurs, the job is terminated.

The ERROPT operand does not apply to output files. For output files for most devices, the job is automatically terminated after IOCS has attempted to retry writing the record; for 2560 or 5424/5425 output files, normal error recovery procedures are followed.

ERROPT applies to wrong-length records if WLRERR is omitted. If both ERROPT and WLRERR are omitted and wrong-length records occur, IOCS ignores the error.

ERROPT specifies the function to be performed for an error block. If an error is detected when reading a magnetic tape, a disk pack, or a diskette volume,

IOCS attempts to recover from the error. If the error is not corrected, the job is terminated unless this operand is included to specify other procedures to be taken. The three specifications are described below.

IGNORE
: Indicates that the error condition is to be ignored. The address of the error record is made available to you for processing (see "CCB Macro" in "Chapter 9. Processing Files with PIOCS (Physical IOCS)").

SKIP
: Indicates that the error block is not to be made available for processing. The next record is read and processing continues.

name
: Indicates that IOCS is to branch to your routine when an error occurs, where you may perform whatever functions are desired or simply note the error condition. The address of the error record is supplied in register 1. The contents of the IOREG register may vary and should not be used for error records. Also, you must not issue any GET instructions in your error routine. If you use any other IOCS macros for a file assigned to a DASD, you must save the contents of register 14. If RDONLY=YES is specified, you must also save the contents of register 13. For a file assigned to a DASD, use of a LIOCS macro (other than ERET) for the file in error will cause the task to be terminated. At the end of the error routine, return to IOCS by branching to the address in register 14. The next record is then made available for processing.

End-Of-File Handling

The EOFADDR operand specifies the name of your end-of-file routine. It is required only if SYSIPT or SYSRDR is specified.

IOCS branches to this routine when it detects an end-of-file condition. In this routine, you can perform any operations necessary for the end-of-file condition (you generally issue the CLOSE macro).

IOCS detects the end-of-file condition by recognizing the characters /* in positions 1 and 2 of the record for cards, a tapemark for tape, and an end-of-file record for disk. If the system logical units SYSIPT and SYSRDR are assigned to a 5424/5425, IOCS requires that the /* card, indicating end-of-file, be followed by a blank card. An error condition results if the records are allowed to run out without a /* card (and without a /& card, if end-of-job). IOCS detects the end-of-file condition on diskette units by recognizing that end-of-data has been reached on the current volume and that there are no more volumes available.

Chapter 9: Processing Files with PIOCS (Physical IOCS)

When your program processes magnetic tape, DASD, or diskette files by means of the PIOCS macros (such as EXCP and WAIT), the files must first be defined by the DTFPH declarative macro. No logic module generation macro is needed. The DTFPH macro must also be used for a checkpoint file on disk.

Figures 9-1 and 9-2 list the DTFPH operands; for details of these operands, refer to *VSE/Advanced Functions Macro Reference.*

Operand	Optional	Required
CCWADDR=name	X	
CISIZE=nnnnn	X	
DEVADDR=SYSnnn	X	
DEVICE=2311, 2314, 3330, 3340, 3350, 3540, FBA*, DISK**		X
LABADDR=name	X	
MOUNTED=SINGLE		X
TYPEFLE=OUTPUT		X
* Specify FBA for 3310 or 3370.		
** Specify DISK to indicate any DASD device. The actual one to be determined at OPEN time.		

Figure 9-1. Operands to define a checkpoint file on disk.

After the files are defined by the DTFPH macro, the imperative macros can be used to operate on the files. The imperative macros are divided into three groups: those for initialization, processing, and completion.

Initialization

The OPEN macro activates files processed with the DTFPH macro. The macro associates the logical file declared in your program with a specific physical file on a DASD. The association remains in effect throughout your processing of the file until you issue a CLOSE macro.

If OPEN attempts to activate a logical IOCS file (DTF) whose device is unassigned, the job is terminated. If the device is assigned IGN, OPEN does not activate the file and turns on DTF byte 16, bit 2, to indicate the file is not activated. If the file is not activated, do not attempt I/O operations, as unpredictable results may occur.

Enter the symbolic name of the file (DTF filename) in the operand field. A maximum of 16 files may be opened with one OPEN by entering the filenames as additional operands. Alternatively, you can load the address of the DTF filename into a register and specify the register using ordinary register notation. The address of filename may be preloaded into register 0 or any of the registers 2 through 15. The high-order 8 bits of this register must contain zeros or unpredictable results may occur.

Note: If you use register notation, we recommend that you follow the practice of using only registers 2 through 12.

Whenever an input/output DASD or magnetic tape file is opened and you plan to process user-standard labels (UHL or UTL), or nonstandard tape labels, you must provide the information for checking or building the labels. If this information is obtained from another input file, that file must be opened ahead of the DASD or tape file. Do this by specifying the input file ahead of the tape or DASD file in the same OPEN, or by issuing a separate OPEN preceding the OPEN for the file.

If an output tape specified to contain standard labels is opened and does not contain a volume label, a message is issued to the operator. He can then enter a volume serial number allowing the volume label to be written on the output tape.

Single Volume Mounted - Output

When processing output files with physical IOCS, OPEN is used only if you want to build standard labels. When the first OPEN for the volume is issued, OPEN checks the standard VOL1 label and the extents specified in the EXTENT job control statements for the mounted volume:

1. The extents must not overlap each other.
2. If user standard header labels are written, the first extent must be at least two tracks long.
3. Only type 1 and type 8 extents are valid. (For files assigned to FBA DASD, only type 1 extent is valid.)

OPEN checks all the labels in the VTOC to ensure that the file to be created does not destroy an existing file whose expiration date is still pending. After this check, OPEN creates the standard label(s) for the file and writes the label(s) in the VTOC.

If you wish to create your own user standard header labels (UHL) for the file, you must include the LABADDR operand in the DTF. OPEN reserves the first track of the first extent for these labels and gives control to your label routine. After this, the first extent of the file can be used. Each time you

determine that all processing for an extent is completed, issue another OPEN for the file to make the next extent available. When the last extent on the last volume of the file is processed, OPEN issues a message. The system operator has the option of canceling the job, or typing in an extent on the printer-keyboard and continuing the job. If the system provides DASD file protection, only the extents opened for the mounted volume are available to you.

Single Volume Mounted - Input

When processing input files with physical IOCS, OPEN is used only if you want to check standard labels.

All Volumes Mounted - Output

If all output volumes are mounted when creating an output file with physical IOCS, each volume is opened before the file is processed. OPEN is used only if standard labels are checked or written.

For each volume, OPEN checks the standard VOL1 label and checks the extents specified in the EXTENT job control statements:

1. The extents must not overlap each other.
2. Only type-1 extents can be used.
3. If user standard header labels are created, the first extent must be at least two tracks long.
4. For 3340, all data modules must be of the same type.

OPEN checks all the labels in the VTOC to ensure that the created file does not write over an existing file with an expiration date still pending. After this check, OPEN creates the standard label(s) for the file and writes the label(s) in the VTOC.

When the mounted volume is opened for the first time, OPEN checks the extents specified in the extent cards (for example, checks that the extent limit address for the device being opened is valid). OPEN also checks the standard VOL1 label and then checks the file label(s) in the VTOC. If the system provides DASD file protection, only the extents opened for the mounted volume are available for use.

If LABADDR is specified, OPEN makes the user standard header labels (UHL) available to you one at a time for checking. Then, OPEN makes the first extent available for processing.

Each time you determine that all processing for an extent is completed, issue another OPEN for the file to make the next extent available if MOUNTED=SINGLE is specified. If another extent is not available, OPEN stores the character F (for EOF) in byte 31 of the DTFPH table. You can determine the end of file by checking the byte at filename +30.

If you wish to create your own user standard header labels for the file, include the LABADDR operand in the DTF. OPEN reserves the first track of the first extent for these labels and gives control to your label routine.

If the XTNTXIT operand is specified, OPEN stores the address of a 14-byte extent information area in register 1. Then, OPEN gives control to your extent routine. You can save this information for later use in specifying record addresses. If your DASD file is file protected, you cannot write on any extents while in the XTNTXIT routine. When checking is complete, return control to OPEN by issuing the LBRET 2 macro

M	TYPEFLE=xxxxxx	INPUT or OUTPUT. Specifies type of file.
O	ASCII=YES	ASCII file processing is required.
O	CCWADDR=xxxxxxxx	To be used if CCB is generated by DTFPH.
O	CISIZE=nnnnn	For Fixed Block Architecture DASD; Control Interval size.
O	DEVADDR=SYSxxx	Symbolic unit required only when not provided on an EXTENT statement.
O	DEVICE=xxxx	(TAPE, DISK, FBA, 2314, 3330, 3340, 3350, 3540). Specify FBA for 3310 or 3370. If omitted, TAPE is assumed.
O	HDRINFO=YES	Print header label information.
O	LABADDR=xxxxxxxx	Routine to check or build user standard labels.
O	MOUNTED=xxxxxx	ALL or SINGLE. Required for DASD files only; for diskette files, specify SINGLE.
O	XTNTXIT=xxxxxxxx	If EXTENT statements are to be processed, DASD only.

M = Mandatory
O = Optional

Figure 9-2. DTFPH macro operands.

which opens the next volume. After all volumes are opened, the file is ready for processing.

All Volumes Mounted - Input

When all volumes containing the input file are on-line and ready at the same time, each volume is opened, one at a time, before any processing is done. OPEN is used only when standard labels are to be processed. For each volume, OPEN checks the extents specified in the EXTENT job control statements, and checks the standard VOL1 label on track 0 and the file label(s) in the VTOC. If LABADDR is specified in the DTF, OPEN makes the user standard labels available, one at a time, for checking.

If XTNTXIT is specified, OPEN stores the address of a 14-byte extent information area into register 1. Then OPEN gives control to your extent routine. For example, you can save this area and use the information later on for specifying record addresses. If the DASD file is file protected, you cannot write on any extents while in the XTNTXIT routine.

Diskette Volumes - Output

When processing output files on diskettes with physical IOCS, OPEN is used to build standard labels. When OPEN is issued for the first volume, it checks the VTOC on the diskette, and

- ensures that the file to be created does not have the same name as an existing unexpired file;
- ensures there is at least one track available to be allocated;
- allocates space for the file, starting at the track following the last unexpired or write-protected file on the diskette.

After this check, OPEN creates the format-1 label for the file and writes the label in the VTOC. Each time you determine that all processing for an extent is complete, you must feed to make the next diskette available and then issue another OPEN for the file, to make the next extent available. CLOSE will automatically cause the last volume to be fed out. If the last extent of the file is completely processed before a CLOSE is issued, OPEN assumes an error condition and the job is canceled.

Diskette Volumes - Input

When processing input files on diskettes with physical IOCS, OPEN is used to check standard labels.

When the first volume is opened, OPEN checks the VTOC on the diskette and determines the extent limits of the file from the file label.

After the label is checked, OPEN makes the first extent available for processing. Each time you determine that all processing for a diskette is complete, you must feed to make the next diskette available, and then issue another OPEN for the file, to make the next extent available. If another extent is not available, OPEN stores the character F (for EOF) in byte 31 of the DTFPH table. You can determine the end of file by checking the byte at filename +30.

For a programmer logical unit, the last diskette will always be fed out; for a system logical unit, the last diskette will not be fed out.

Processing

In order to process a file by means of physical IOCS, you must provide either a Command Control Block (CCB) or an I/O Request Block (IORB) for each I/O device. These control blocks are used to maintain communications between your program and PIOCS about such things as determining the status of the device in use and specifying the operations that you want performed.

Using the operands that you specify, the CCB *macro* generates a Command Control Block of either 16 or 24 bytes. See Figure 9-3 for the format and contents of the CCB. Similarly, the IORB and GENIORB macros generate an I/O Request Block, which is the same as a CCB except that, in the IORB, bytes 6-12 and 16-23 are reserved for use by PIOCS. Using the IORB or GENIORB macros instead of the CCB macro allows you additional options, such as specifying areas to be page-fixed. This frees the system from having to determine which areas are to be fixed.

The macros differ from one another further in that issuing a CCB or IORB macro generates the block when the program is assembled, while GENIORB generates it at execution time. Otherwise, depending on the requirements of your program, the macros may be interchangeable.

For details of the CCB, IORB, and GENIORB macros and their operands, see *VSE/Advanced Functions Macro Reference.*

The significance of bits in bytes 2 and 3 of the Command Control Block, the 'transmission information' or 'user option bits,' is listed in Figure 9-4. The CCB macro can set any or all of these bits; if more than one bit must be set, the sum of the values is used. For example, to set on user option bits 3, 5, and 6 of byte 2, X'1600' is used.

(X'1600'=X'1000' + X'0400' + X'0200')

Only certain of these bits can be set on when you use the IORB or GENIORB macros; you do this by specifying the IOFLAG operand.

When certain I/O devices are used, user option bits must be set. For instance, if the CCB (or IORB) is for an IBM 2560 or 5424/5425, option bit 5 of CCB byte 2 (post at device end) must be set on. If command chaining is used in the channel programs for these devices, or for the 3895, option bit 7 of CCB byte 3 (command chain retry) must also be set on.

Also, if your program has its own user error routine (byte 2, bit 7 of the CCB is on, or X'nnnn' in the CCB macro = X'0100') but has not specified a sense address in the CCB macro, the sense information is cleared by the supervisor in order to prevent deadlocks in the control unit. If the user then issues an EXCP with the CCW address for SENSE from the error routine, the information has already been destroyed.

(When under control of LIOCS, the CCB macro is generated as a result of the DTFxx macro).

The EXCP (EXecute Channel Program) macro is used for including an I/O request to PIOCS. It is translated into an SVC instruction (which calls the channel scheduler) and a reference to the CCB or IORB. This reference can be given as a symbolic reference by means of a *ccbname* or a reference to a general register which contains its address.

Physical IOCS determines the device from the CCB specified by blockname, places the CCB in a queue of such CCBs for this device, and returns control to your program. Physical IOCS causes the channel program to be executed as soon as the channel and device are available. I/O interruptions are used to process I/O completion and to start I/O for requests if the channel or device was busy at the time the EXCP was executed. You can determine the actual device type to which a logical unit is assigned by means of the EXTRACT ID=PUB macro.

PIOCS does not wait for the completion of an I/O operation after the operation has been started. Instead, control is returned to the problem program, which must make sure that it does not start processing data that has not been completely read, or start overwriting an output area before the previous block has been completely written. A problem program can wait for the completion of an I/O operation by issuing a WAIT macro that refers to a CCB or IORB. The reference can be made in either way described above for the EXCP macro. The effect of a WAIT macro is another SVC instruction which checks, in the interrupt routine, the status of the I/O operation in the process.

When WAIT is executed, the supervisor gives control to another program until the traffic bit (byte 2, bit 0) of the related CCB or IORB is turned on.

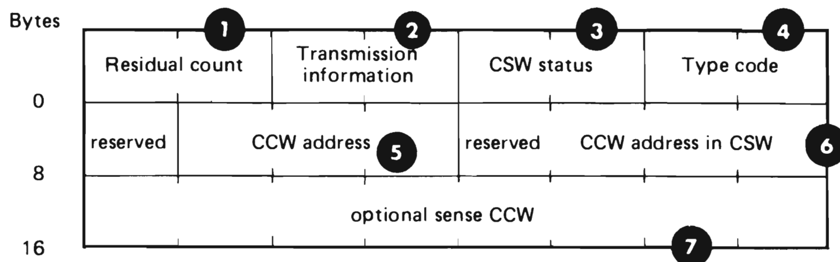
The relationship between the three PIOCS macros CCB, EXCP, and WAIT is shown in Figure 9-5. Note that in this figure the assembler instruction CCW is illustrated as well. The CCW instruction is not a macro. It is included in the figure only for clarification. Note also that a IORB or GENIORB macro may be substituted for the CCB, in some cases.

The EXTRACT ID=PUB macro retrieves partition-related device information, that is, for a given logical unit PUB information can be obtained. The PUB information retrieved can be interpreted using the mapping DSECT generated by the IJBPUB macro.

The SECTVAL macro calculates the sector value of the address of the requested record on the track of a disk storage device when RPS is used. The macro returns this value in register 0.

The sector value is calculated from data length, key length, and record number information. Values are calculated for fixed or variable length and for keyed and non-keyed records.

The LBRET macro is issued in your subroutines when processing is completed and you wish to return control to IOCS. LBRET applies to subroutines that write or check tape nonstandard labels, or check DASD extents. The operand used depends on the function to be performed (see the section "Label Processing").



1 After a record has been transferred, IOCS places the residual count from the CSW into these two bytes. By subtracting the residual count from the original count in the CCW, your program can determine the length of the transferred record. The field is set to zero for negative values.

2 Used for transmission of information between physical IOCS and your program. For detailed information on the use and purpose of the individual bits in this field, see Figure 9-4. Your program can test any of the bits in this field using the mask given in the last column of Figure 9-4. Your program may test more than one bit by the hexadecimal sum of the test values.

All bits are set to 0 when your program is assembled unless the X'nnnn' operand is specified. If this operand is specified, it is assembled into these two bytes. When your program is being executed, each bit may be set to 1 by your program (to request certain functions or specific feedback information) or by physical IOCS (as a result of having detected a particular condition). Any bits that can be turned on by physical IOCS during program execution are reset to zero by PIOCS the next time an EXCP macro is executed against the same CCB.

3 Byte 4 is set to X'00' whenever an EXCP macro is issued against the CCB. For non-teleprocessing devices, a program-controlled interruption (PCI) is ignored.

The meaning of the bits in these two bytes is as follows:

Byte 4:	Byte 5:
0 = attention	0 = program-controlled interruption
1 = status modifier	1 = incorrect length
2 = control unit end	2 = program check
3 = busy	3 = protection check
4 = channel end	4 = channel data check
5 = device end	5 = channel control check
6 = unit check	6 = interface control check
7 = unit exception	7 = chaining check

If bit 5 of CCB byte 2 is set to 1 and device end results as a separate interrupt, device end will be posted.

4 Contents of byte 6:

X'0u' = original CCB
X'4u' = BTAM-ES CCB
X'8u' = user-translated CCB in virtual partition

Note: if u = 0: the address in byte 7 refers to a system logical unit.
if u = 1: the address in byte 7 refers to a programmer logical unit

Contents of byte 7:
Hexadecimal representation of SYSnnn as follows:

SYSRDR = 00	SYS000 = 00
SYSIPT = 01	SYS001 = 01
SYSPCH = 02	SYS002 = 02
SYSLST = 03	•
SYSLOG = 04	•
SYSLNK = 05	•
SYSRES = 06	SYS254=FE
SYSSLB = 07	
SYSRLB = 08	
SYSUSE = 09	
SYSREC = 0A	
SYSCLB = 0B	
SYSDMP = 0C	
SYSCAT = 0D	

5 Address of CCW (or of the first of a chain of CCWs) associated with the CCB:
This is a real address if CCB byte 6 = X'8u'.
This is a virtual address if CCB byte 6 = X'0u'.

6 Either of the following:
The CCW address contained in the CSW at channel-end interrupt for the I/O operation involving the CCB; or the address of the associated channel appendage routine if CCB byte 12 contains X'40'.

7 Bytes 16 to 23 are provided only if the sense operand was specified in the CCB macro. They accommodate the CCW for returning sense information to your program.

Figure 9-3. Layout and contents of Command Control Block (CCB).

Byte	Bit	Condition Indicated		On Values for Third Operand in CCB Macro	Mask for Test Under Mask Instruction
		1 (ON)	0 (OFF)		
2	0 Traffic Bit (WAIT)	I/O Completed. Normally set at Channel End. Set at Device End if bit 5 is on.	I/O requested and not completed.		X'80'
	1 End of File on System Input 3211 UCB Parity Check (line complete)[8]	/* or /& on SYSRDR or SYSIPT. Byte 4, Unit exception Bit is also on. Yes	 No		X'40'
	2 Irrecoverable I/O Error	I/O error passed back due to program option or operator option.	No program or operator option error was passed back.		X'20'
	3[1] Accept Irrecoverable I/O Error (Bit 2 is ON)	Return to user after physical IOCS attempts to correct I/O error.[2]	Operator Option: Dependent on the Error	X'1000'	X'10'
	4[1] Return: DASD data checks, 3540 data checks, 2671 data checks, 1017/1018 data checks. 5424/5425 not ready. Indicate action type messages for DOC	Operator Options: Ignore, Retry, or Cancel. Ignore or Cancel. Return to user.	Operator Options: Retry or Cancel. Cancel.	X'0800'	X'08'
	5[1] Post at Device End. Specify this bit to be set on for a 2560 or 5424/5425.	Device End condition is posted: that is, byte 2, bit 0 and byte 3, bits 2 and 6 set at Device End. Also byte 4, bit 5 is set.	Device End conditions are not posted. Traffic bit is set at Channel End.	X'0400'	X'04'
	6[1] Return: Uncorrectable tape read data check (2400 series or 3420); 1018, 2560 data check, 2520 or 2540 punch equipment check; 2560, 5424/5425 read, punch, print data, and print clutch equipment checks; 3881 equipment check; 3504, 3505, or 3525 permanent errors; DASD read or read verify data check; 3211 passback requested; 3895 error codes requested. (Data checks on count not retained)[8]	Return to user; after physical IOCS attempts to correct 3211[8], tape, or DASD error; when 1018 or 2560 data check[4]; when 2560 or 5424/5425 equipment check; when 3504, 3505, 3525 permanent error (byte 3, bit 3 is also on).[4,8]	Operator Option: Ignore or Cancel for tapes, paper tape punch (1018), card punches other than 2560 and 5424/5425. Retry or Cancel for DASD, 2560, or 5424/5425.	X'0200'	X'02'
	7[1] User Error Routine	User handles error recovery.[3]	A physical IOCS error routine is used unless the CCB sense address operand is specified. The latter requires user error recovery.	X'0100'	X'01'

Figure 9-4. Conditions indicated by CCB bytes 2 and 3 (Part 1 of 3).

Byte	Bit		Condition Indicated 1 (ON)	Condition Indicated 0 (OFF)	On Values for Third Operand in CCB Macro	Mask for Test Under Mask Instruction
3	0	Data check in DASD count field. Permanent error for 3330, 3340, or 3350.	Yes-Byte 3, bit 3 is off; Byte 2, bit 2 is on.	No		X'80'
		Data check - 1287 or 1288.	Yes	No		
		MICR - SCU not operational.	Yes	No		
		3211 Print Check (equipment check).[7,8]	Yes	No		
		3540 special record transferred.	Yes	No		
	1	DASD Track overrun.	Yes	No		X'40'
		1017 broken tape.	Yes	No		
		Keyboard correction 1287 in Journal Tape Mode	Yes	No		
		3211 print quality error (equipment check)[8].	Yes	No		
		MICR intervention required.	Yes	No		
	2	End of DASD Cylinder.	Yes	No		X'20'
		Hopper Empty 1287/1288 Document Mode.	Yes	No		
		MICR - 1255/1259/1270/1275/1419, disengage.	Document feeding stopped.	No		
		1275/1419D, I/O error in external interrupt routine.	Channel data check or Bus-out check.			
		3211/2245 line position error.[5,8]	Yes	No		
	3	Tape read data check (2400 series); 2520, 2540 or 3881 equipment check; any DASD data check.	Operation was unsuccessful. Byte 2, bit 2 is also on. Byte 3, bit 0 is off.	No		X'10'
		1017, 1018 data check.	Yes	No		
		1287, 1288 equipment check.	Yes	No		
		2560, 3203, 5203, 5424/5425 read, punch, print data, and print clutch equipment checks.	Byte 2, bit 6 is also on.	No		
		3504, 3505, 3525 permanent errors.	Byte 2, bit 6 is also on.	No		
		3211 data check/print check.[8]	Yes	No		
		3540 data check.	Yes	No		
	4	Nonrecovery Questionable Condition.	Card: unusual command sequence. For DASD, no record found. 1287, 1288 document jam or torn tape. 3211 UCB parity check (command retry). 5424/5425 not ready.			X'08'
	5[1]	No record found condition (retry on 2311, 2314, 2319, 3330, 3340, or 3350).	Retry command if no record found condition occurs (disk).	Set the nonrecovery questionable condition bit on and return to user.	X'0004'	X'04'

Figure 9-4. Conditions indicated by CCB bytes 2 and 3 (Part 2 of 3).

Byte	Bit		Condition Indicated		On Values for Third Operand in CCB Macro	Mask for Test Under Mask Instruction
			1 (ON)	0 (OFF)		
3	6	Verify error for DASD or Carriage Channel 9 overflow.	Yes. (Set on when Channel 9 is reached only if Byte 2, bit 5 is on).	No		X'02'
		1287 document mode: late stacker select.	Yes	No		
		1288 End-of-Page (EOP).	Yes	No		
	7[1]	Command Chain Retry. Specify this bit to be set on if command chaining is used for a 2560 or 5424/5425.	Retry begins at last CCW executed.[6]	Retry begins at first CCW or channel program.	X'0001'	X'01'

Notes:

1 User Option Bits. Set in CCB macro. Physical IOCS sets the other bits off at EXCP time and on when the specified condition occurs.
2 I/O program check, command reject, or tape equipment check always terminates the program.
3 You may not handle Channel Control Checks and Interface Control Checks. The occurrence of a channel data check, unit check, or channel chaining check cause byte 2, bit X'20' of the CCB to turn on, and completion of posting and dequeuing to occur. I/O program and protection checks always cause program termination. Incorrect length and unit exception are treated as normal conditions (posted with completion). Also, you must request device end posting (CCB byte 2, bit X'04') in order to obtain errors after channel end.
4 Error correction feature for 1018 is not supported by physical IOCS. When a 1018 data check occurs and CCB byte 2, bit X'02' is on, control returns directly to you with CCB byte 3, bit X'10' turned on.
5 A line position error on the 3211 can occur as a result of an equipment check, data check, or FCB parity check.
6 If an error occurs, physical IOCS updates the CCW address in bytes 9 through 11 of the CCB that is used for the pertinent I/O operation. The original CCW address must therefore be restored before another I/O operation using the same CCB is issued.
7 A deleted or bad spot record has been read on a 3540 diskette. CCW chain broken, after CCW reads special record.
8 3211 remarks apply also to 3211-compatible printers (that is, with device type code of PRT1). 3895 error codes are returned in CCB byte 8. Refer to the 3895 Document Reader/Inscriber manuals for information on these error codes.

Figure 9-4. Conditions indicated by CCB bytes 2 and 3 (Part 3 of 3).

Processing Labels and Extents

Checking User Standard DASD Labels: IOCS passes labels to you one at a time until the maximum allowable number is read and updated, or until you signify you want no more. Use LBRET 3 in your label routine if you want IOCS to update (rewrite) the label read and pass you the next label. Use LBRET 2 if you want IOCS to read and pass you the next label. If an end-of-file record is read when LBRET 2 or LBRET 3 is used, label checking is automatically ended. If you want to eliminate the checking of one or more remaining labels, use LBRET 1.

Writing User Standard DASD Labels: Build the labels, one at a time, and use LBRET to return to IOCS to write the labels. Use LBRET 2 if you wish to regain control after IOCS wrote the label. If however, IOCS determines that the maximum number of labels has been written, label processing is terminated. Use LBRET 1 to stop writing labels before the maximum number is written.

Checking DASD Extents: When processing an input file with all volumes mounted, you can process your extent information. After each extent is processed, use LBRET 2 to receive the next extent. When extent processing is complete, use LBRET 1 to return control to IOCS.

Checking User Standard Tape Labels: IOCS reads and passes the labels to you one at a time, until a tapemark is read, or until you indicate that you want no more labels. Use LBRET 2 if you want to process the next label. Use LBRET 1 if you want to bypass any remaining labels.

Writing User Standard Tape Labels: Build the labels one at a time and return to IOCS, which writes the labels. You are responsible for accumulating the block count, if desired, and supplying it to IOCS for inclusion in the standard trailer label; for this, the count (in binary form) must be moved to the 4-byte field named filenameB. When LBRET 2 is used, IOCS returns control to you (at the address specified in LABADDR) after writing the label. LBRET 1 must be used to terminate the label set.

Writing or Checking Nonstandard Tape Labels: You must process all your nonstandard labels at once. LBRET 2 is used after all label processing is completed and you want to return control to IOCS. For an example see Appendix C.

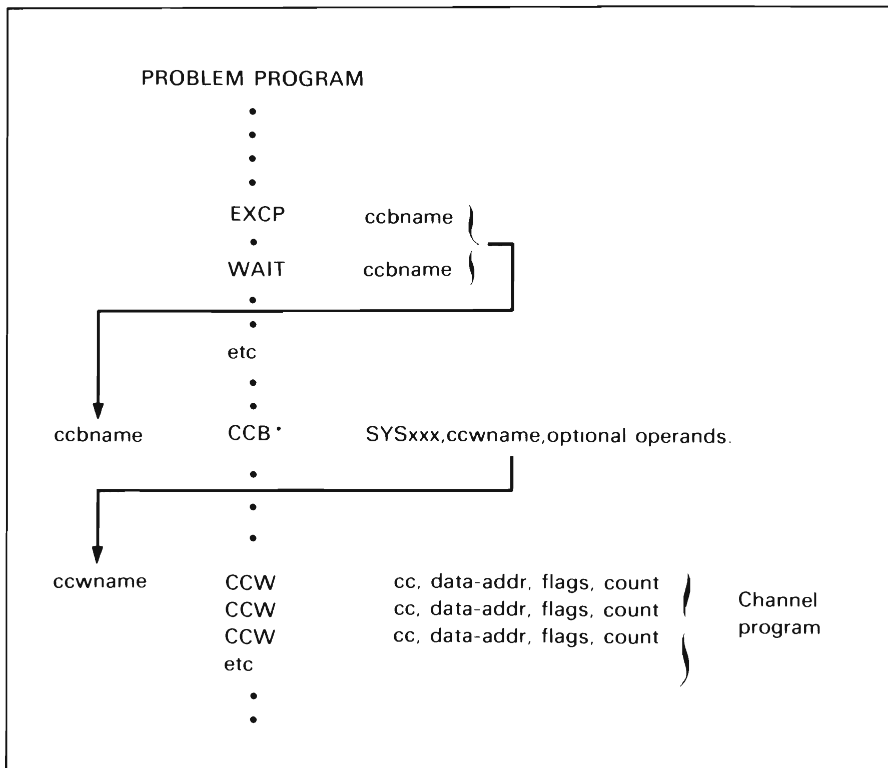


Figure 9-5. Relationship between the PIOCS macros.

Forcing End-of-Volume

The FEOV (forced end-of-volume) macro is used for files on magnetic tape (programmer logical units only) to force an end-of-volume condition before sensing a reflective marker. This indicates that processing of records on one volume is considered finished, but that more records for the same logical file are to be read from, or written on, the following volume.

When physical IOCS macros are used and DTFPH is specified for standard label processing, FEOV may be issued for output files only. In this case, FEOV writes a tapemark, the standard trailer label, and any user-standard trailer labels if DTFPH LABADDR is specified. When the new volume is mounted and ready for writing, IOCS writes the standard header label and user-standard header labels, if any.

The SEOV (system end-of-volume) macro must only be used with physical IOCS to automatically switch volumes if SYSLST or SYSPCH are assigned to a tape output file. SEOV writes a tapemark, rewinds and unloads the tape, and checks for an alternate tape. If none is found, a message is issued to the operator who can mount a new tape on the same drive and continue. If an alternate unit is assigned, the macro fetches the alternate switching routine to promote the alternate unit, opens the new tape, and makes it ready for processing. When using this macro, you must check for the end-of-volume condition in the CCB.

Termination

The CLOSE macro is used to deactivate any file that was previously opened. Console files, however, cannot be closed. The macro ends the association of the logical file declared in your program with a specific physical file on an I/O device. A file may be closed at any time by issuing this macro. No further commands can be issued for the file unless it is opened.

A maximum of 16 files may be closed by one macro by entering additional filename parameters as operands. Alternatively, you can load the address of the filename in a register by using ordinary register notation. The address of the filename may be preloaded into register 0 or any of the registers 2 through 15. The high-order eight bits of this register must be zeros.

Notes:

1. If you use register notation, we recommend that you follow the practice of using only registers 2 through 12.
2. If CLOSE is issued to an unopened tape input file, the option specified in the DTF rewind option is performed. If CLOSE is issued to an unopened tape output file, no tapemark or labels are written.

PIOCS Programming Considerations

The following paragraphs indicate a few programming problems and explain how they can be solved. Some restrictions are also mentioned.

Situations Requiring LIOCS Functions in PIOCS Processing

In explaining PIOCS it was said that the programmer is responsible for providing all of the logical functions that are normally provided by LIOCS routines. There are, however, two exceptions to this rule. In fact, a programmer *must* use some of the logical functions of LIOCS for:

1. DASD files that are file-protected.
2. Magnetic tape files, diskette files, or DASD files that require standard label processing.

These two situations are closely related to access control and data integrity control, for which a system can accept responsibility only if it is recognized as the only authority having access to system information.

In either of the two situations above, files must be defined to LIOCS by means of the DTFPH macro. This macro, like any other DTFxx macro for LIOCS, specifies the characteristics of the file. The logic module will provide the minimum facilities necessary for label processing and protecting files, where applicable. Label processing will be performed, as usual, in response to the OPEN and CLOSE macros.

In addition, the FEOV macro can be used for volume switching on magnetic tape output files.

If the DTFPH macro is used, a program may look slightly different from the example given in Figure 9-5. As was explained before, the DTF table contains the CCB in the first 16 bytes, so that the EXCP and WAIT macros can now refer to the name of the DTFPH macro. The DTFPH macro in turn contains a parameter CCWNAME=ccwname, so that the CCB has the proper reference to the first CCW in the appropriate channel program (see Figure 9-6).

For information about the CCW format and the concepts of data chaining and command chaining, refer to *System/370 Principles of Operation* or to the *IBM 4300 Processors Principles of Operation,* as listed in the Preface.

Command Chaining Retry

If a system has been generated to support command chaining retry, you can use this option for your PIOCS channel programs by setting the *command chaining retry bit* in the CCB on. If an error that involves retry occurs, the retry will then begin with the last CCW executed. If this bit is off, the entire channel program will be re-executed.

When the command chaining retry bit is on, you must move the address of the first CCW in the channel program to bytes 9-11 of the CCB before an EXCP is issued. This ensures that the CCB always contains the correct CCW address; bytes 9-11 are modified by PIOCS for a retry after an error with the address of

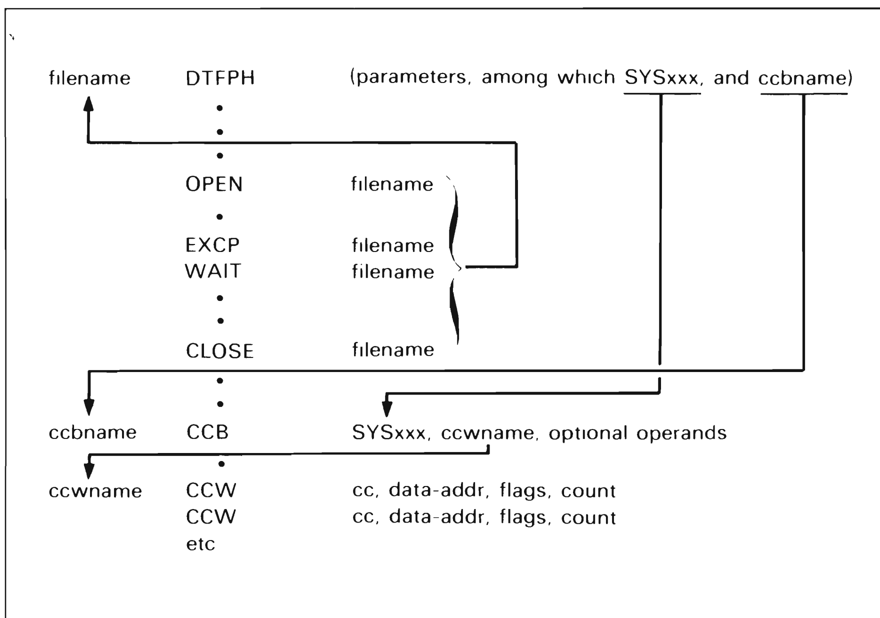


Figure 9-6. Relationship between DTFPH and other PIOCS macros.

the CCW to be re-executed, and is not reset to its original value.

If a command chain is broken by some exceptional condition (for example, wrong-length record, or unit exception) that does not result in device error recovery by IOCS, you can determine the address of the last CCW executed and, if necessary, restart the channel program at that point. To obtain the address of the last CCW executed, subtract 8 from the address in bytes 13-15 of the CCB. On a 1403 printer, a command chain is broken after sensing channel 9 or 12. When using command chaining on a printer, the program should therefore always check if the entire CCW chain has been executed.

The command chaining retry bit must not be used to read multiple blocks from SYSIPT or SYSRDR. Moreover, this bit should *never* be on for DASD or diskette channel programs.

Restrictions for the 3800 Printing Subsystem

Before using PIOCS on a 3800, consult the *DOS/VS IBM 3800 Printing Subsystem Programmer's Guide*, to learn about channel programming restrictions that should be followed to prevent errors for subsequent jobs.

Data Chaining

When performing data chaining, the CCWs in a channel program should all contain the proper command code of the operation to be executed, in order to ensure proper I/O error recovery. In normal cases where nothing goes wrong, the command code is not used if a preceding CCW has the data chaining bit on. In case of an error, however, recovery frequently depends on the command being executed and the command code in the last CCW is often examined. In such a case, a 'dummy' command code might prevent error recovery.

CKD DASD Channel Programs

The user should begin a DASD channel program with a full seek (command code X'07'); if the channel program contains embedded seeks, they should be full seeks as well.

If embedded full seeks are used, a program cannot run under DASD file protection, nor can it take full advantage of the seek separation feature. With DASD file protection, an embedded full seek causes cancellation of the program in error.

The seek separation feature initiates a seek and separates it from the channel program chain. Thus, the channel is available for other input or output operations on the same channel. The seek separation feature, however, applies only to the first seek in a channel command chain.

When executing a channel program (Figure 9-7), the supervisor sets up a channel program with three commands:

1. A *Seek* that is identical to the user's seek.
2. A *Set File Mask* that prevents other X'07' seeks from being executed.
3. A *Transfer in Channel* (TIC) command that transfers control to the command following the user's seek.

RPS (Rotational Position Sensing)

If a system has been generated to support RPS, the user can include the channel commands *set sector* and *read sector* for DASDs supporting the feature (standard on the IBM 3330/3333/3350; optional on the IBM 3340). These commands can be used to establish the angular position of the requested record relative to the read/write head and to free the channel for other operations until the requested record is under the read/write head.

The SECTVAL macro can be used to calculate a sector value for a specified record.

Channel Programs for FBA Devices

Access to data on FBA devices occurs via the execution of a channel program with three basic steps:

1. The channel program for an FBA device must start with a DEFINE EXTENT command (command code X'63') — there is no SEEK CCW for FBA devices. There should be only one DEFINE EXTENT command in the channel program.

   The DEFINE EXTENT command defines the location and size of a data extent; that is, it establishes bounds on the storage medium within which subsequent chained commands are permitted to operate. The command also contains a file protect mask for controlling the execution of following commands.
2. A LOCATE command (command code X'43') specifies the location and number of addressable blocks of the data space to be processed, and the operation (read or write) to be performed.
3. When the storage device is positioned for a data transfer, a READ or WRITE command causes the transfer of data between the storage device and main storage. A READ or WRITE command must be chained from a LOCATE command. If the chaining prerequisite is not satisfied, the

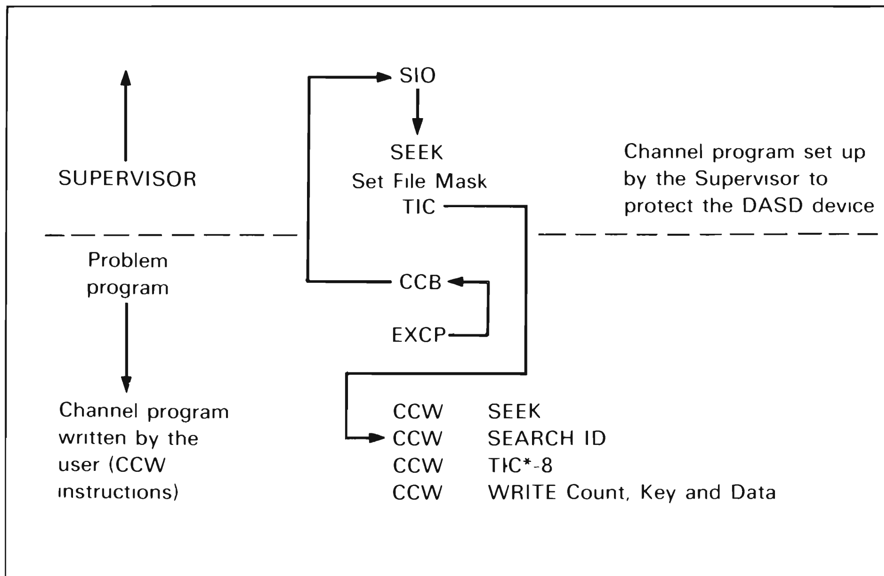


Figure 9-7. Example of channel programming a file-protected CKD DASD file.

command is rejected with a unit check (command reject). Although READ (WRITE) commands may not be command chained from another READ (WRITE), data chaining of READ (WRITE) CCWs is permitted. However, data chaining within a block may cause unpredictable overruns or chaining checks.

Diskette Channel Programs

The user must begin a diskette channel program with a *define operations* command (command code X'2F'). This command is intended for use during the program initiation, and sets the operating mode for a file during program execution. It defines whether read or write operations will be done. If write operations are to be done, the define operations command determines how many writes will be done between seeks. This command must be reissued to change the mode of operations on the file.

Following the define operation should be a seek (command code X'07').

Following the seek should be the read or write CCWs. You can chain 1, 2, 13, or 26 read/writes. You have to check however, where your chaining begins. With 26 chained records, for instance, you have to start chaining on a track boundary. Record length can be chosen freely up to 128 bytes. If write operations are being performed, a NOP command should be chained to the last write command to ensure that any errors occurring on this channel program are returned.

Console (Printer-Keyboard) Buffering

If the console buffering option is specified at system generation and if the printer-keyboard is assigned to SYSLOG, throughput on *output* under PIOCS can be increased for physical blocks that do not exceed 80 characters. This is accomplished by starting the I/O command and returning to the problem program before the output is completed.

Blocks are always printed in a FIFO (first-in-first-out) order, regardless of whether the output blocks are buffered (queued on an I/O completion basis).

Console buffering is performed on output only if the following conditions are maintained:

- The actual block to be written must not exceed 80 characters.
- No data chaining or command chaining must be performed.
- The acceptance of unrecoverable I/O errors, of posting at device end, or of user error routines must not be indicated in the CCB associated with the operation.
- Sense information must not be requested by the CCB.

Alternate Tape Switching

Alternate tape drives cannot be used on *input* processed by PIOCS.

On output, automatic alternate tape drive switching can be done through the DTFPH and FEOV macros. The FEOV (Force End of Volume) macro writes the trailer label sets (standard labels and any desired user labels), and deactivates the current volume.

The next volume is then mounted on the alternate tape drive, and IOCS writes the header label sets (standard labels and any desired user labels) on the new volume.

Bypassing Embedded Checkpoint Records on Magnetic Tape

Checkpoint information is written as a set of magnetic tape records:

- One 20-byte header record;
- One status descriptor record in which the status of the system is saved;
- As many core-image records as are needed to save the required parts of virtual storage;
- One 20-byte trailer record which is identical to the header record.

Depending on whether the file is processed forward or backward, the header or trailer record can be used to recognize and bypass checkpoint sets. The format of both header and trailer record is:

Bytes	Contents
00-11	///bCHKPTb// (note the two space characters: one before, and one after 'CHKPT')
12-13	The number (in binary) of checkpoint records containing program data.
14-15	The total number (in binary) of checkpoint records.
16-19	The serial number of the checkpoint taken.

Checkpoint data sets can always be identified by the first 12 bytes of the header record or trailer record (depending on whether the file is read forward or backward).

When the file is read forward, the checkpoint header record occupies the 20 high-order bytes of the I/O area; when the file is read backward, the checkpoint trailer record occupies the 20 low-order bytes of the I/O area.

These methods may be used to bypass checkpoint sets:

1. Go into a read loop, until a checkpoint trailer record (reading forward) or header record (reading backward) is encountered.
2. Extract bytes 14-15 from the header or trailer record and space forward or backward that number of records.

Method 2 allows the use of read commands instead of forward space commands, except when the checkpoint data sets are on a 7-track-tape. Using read commands in that case results in data checks.

Program Loading

Phases are normally loaded by the supervisor in response to a // EXEC phasename job control statement. However, through the use of a FETCH, LOAD, or CDLOAD macro, an executing phase can load another phase. The phase to be loaded may be in the system core image library, or a private core image library, or it may be in the SVA (shared virtual area). The FETCH macro gives control to the phase just loaded. With the LOAD or CDLOAD macro, control remains with the phase that issued the macro. The load and entry point for the requested phase varies as described below for each of the three macros.

Fetch Macro

The load point and the entry point of the requested phase are the addresses determined when the phase was link-edited. A different entry point may be specified in the FETCH macro. The load point must be in the same partition as the requesting phase. The FETCH macro may not load self-relocating phases.

Load Macro

The load point and entry point of the requested phase may be the addresses determined when the phase was link-edited. The entry point is returned to you in register 1, and it is up to you to transfer control to the newly loaded phase. The LOAD macro permits you to override the link-edited load point of the subject phase. When you override the link edit load point, the entry point address is also relocated. For non-relocatable or self-relocating phases no other addresses will be relocated by the supervisor. A relocatable phase, however, will have all addresses relocated to reflect the current load point of the phase.

CDLOAD Macro

The CDLOAD macro requests a phase to be loaded into the partition's GETVIS area, an area available in the partition for dynamic allocation of storage. For more information about the partition GETVIS area, see *VSE/Advanced Functions System Management Guide.* CDLOAD determines the size of the phase, acquires the appropriate amount of GETVIS storage, and loads the phase. The entrypoint address of that phase is returned to you in register 1. The entry point will be at the same relative distance from the actual load point in the partition GETVIS area as it was from the link-edited load point. It is up to you to transfer control to the newly loaded phase.

Shared Virtual Area Considerations for Program Load Macros

A requested phase may reside in the shared virtual area (SVA). In this case, no actual load operation takes place. For a FETCH macro, the system transfers control to the entry point in the SVA and continues executing. For the LOAD and CDLOAD macros, the appropriate addresses are returned into register 1. When the CDLOAD macro is issued from a program running in real mode, the phase will be loaded into the partition GETVIS area even if it exists in the SVA. This is done to conform with the requirements of execution in real mode.

The systems directory list (SDL), located in the SVA, contains copies of system core image library directory entries for frequently used phases. Parameters in the FETCH and LOAD macros cause the SDL to be searched prior to any private core image library directory. By placing the directory entry of a frequently used phase in the SDL and by coding the FETCH and LOAD macros appropriately, you can reduce the time needed for locating the requested phase.

Fast Loading of Frequently Used Phases

For a phase which is being loaded frequently during execution of one program, it is preferable to use the GENL macro rather than to include the phase's directory entry in the SDL. The GENL macro generates a local directory list within the partition. On the first LOAD or FETCH, the supervisor supplies each entry with information that helps to reduce access time on any subsequent LOAD or FETCH.

The following example illustrates how to use the LOAD macro in connection with a local directory list. Note that the first issuance of the LOAD macro serves only to locate a particular entry (xxxxxxx) within the directory list; the contents of the phase are not being transferred into storage (TXT=NO):

```
         •
         •
         •
         LOAD   XXXXXXX,LIST=GENLIST,TXT=NO
         LR     2,0              GET PTR TO DIRECTORY ENTRY
         TM     16(2),X'06'      PHASE FOUND ?
         BO     NOTFOUND         NO
         TM     16(2),X'10'      PHASE IN SVA ?
         BO     NOLOAD           YES, BRANCH AROUND LOAD
         LA     0,LOADPT
         LOAD   (2),(0),DE=YES
NOLOAD   EQU    *                RFG.1 POINTS TO ENTRY POINT
         •
         •
         •
GENLIST  GENL   XXXXXXX,....
         •
         •
LOADPT   DS     0D               LOAD POINT OF OVERLAY PHASE
```

Virtual Storage Control

The macros designed for use by virtual-mode programs, which are discussed in this section, perform the following services:

- fix pages in real storage (PFIX macro) and later free the same pages for normal paging (PFREE macro).
- allocate and release virtual storage dynamically.
- determine the mode of execution of a program (RUNMODE macro).
- extracting partition related information, such as partition boundaries.
- influence the paging mechanism in order to reduce the number of page faults, minimize the page I/O activity, and control the page traffic within a specific partition.

The discussion of the available virtual storage control macros in this section assumes that you are familiar with the virtual storage concept implemented in VSE and described in *Introduction to the VSE System.*

Fixing and Freeing Pages in Real Storage

Parts of virtual-mode programs must be in real storage only at certain times. These parts include not only the instructions and data being processed at any one moment by the CPU, but also data areas for use by channel programs. Instructions and data are always in real storage when being used. Data areas could be paged out during an I/O operation if something were not done to keep them in real storage during the entire operation. The supervisor fixes I/O areas in real storage for the duration of the I/O operation.

There are other parts of a program which cannot tolerate paging, and these parts are not necessarily kept in storage by the system. For instance, programs that control time-dependent I/O operations cannot tolerate paging. A familiar example is a MICR (Magnetic Ink Character Reader) stacker select routine. If a page fault were to occur during the execution of one of these programs, the results would be unpredictable. A page fault in one of these programs can be avoided by fixing the affected pages in real storage, that is, by using the PFIX macro. Using the PFIX macro, you may request that specific pages of your program be fixed in processor storage. Associated with each partition is a defined number of page frames that are available for fixing. The ALLOCR command determines how many page frames are available for the PFIX macro.

The PFIX macro fixes the pages in real storage, regardless of whether these pages are contained in contiguous page frames or not. The supervisor keeps count of the number of times a PFIX was issued for a specific page without this page having been freed. A page that is fixed more than once without having been freed (via the PFREE macro) is not brought in repeatedly and given additional page frames. Instead, the page-fix counter for that page is increased by one each time, and the page remains in the same page frame. If more than 255 PFIX requests were issued for the same page (without having issued PFREE), the issuing task is canceled.

The PFREE macro does not directly free a page for paging out, but each time it is issued, the counter of fixes is reduced by one. As soon as the counter for a page reaches zero, the page can be paged out. At the end of a job step, all pages that have been fixed during the job step are freed. The PFREE macro should be used as soon as possible to make the page frames available to all programs running in virtual mode.

Figure 10-1 is an example using the PFIX and PFREE macros. After the execution of a PFIX macro,

a return code is given in register 15. The meaning of the return codes are:

0- The pages were fixed successfully.

4- You requested more page frames than can be PFIXed in the allocated address space.

8- Insufficient free page frames were available.

12- You specified invalid addresses in your macros.

Note in the example how the return code can be used to establish a branch to parts of the program that handle these specific conditions.

Determining the Execution Mode of a Program

You may have a program that must do different processing depending upon what its execution mode is. It may be impractical to have two separate programs cataloged in the core image library, one program for real mode and another program for virtual mode. The RUNMODE macro can be issued during the execution of the program to inquire which mode of execution is being used. A return code is issued to the program in register 1.

Extracting Partition-Related Information

You may have programs for which you may want to do your own storage management within the partition in which they are running. This may be the case if a program does a lot of I/O and therefore would like to have as many buffers available as the size of the partition permits. In order to do such storage management you therefore need partition related information, such as partition boundaries. Use the EXTRACT ID=BDY macro to retrieve such information. You can interpret the information retrieved with the help of a mapping DSECT generated by the MAPBDY macro.

Influencing the Paging Mechanism

The macro support discussed here is intended primarily for programmers who wish to control the paging activity of the system in order to optimize program execution beyond the level of optimization provided by the VSE paging mechanism.

Releasing Pages: With the RELPAG macro, you inform the page management routines that the contents of one or more pages is no longer required and need not be saved on the page data set. Thus, page frames occupied by these released pages can be claimed for use by other pages, and page I/O activity is reduced.

All program pages paged out as a result of a FCEPGOUT are saved by writing them onto the page data set, unless a RELPAG macro is being executed for one (or more or all) of those pages. A page for which a RELPAG macro is being executed is not saved.

Forcing Page-Out: The FCEPGOUT macro is used to inform the page management routines that one or more pages will not be needed until a later stage of

```
          •
          •
FIXER     PFIX   ARTN,ARTNEND+2 FIX ARTN IN STORAGE
          B      *+4(15) BRANCH ACCORDING TO RETURN CODE
          B      HERE    CONTINUE IF OK
          B      NOPAGES GO TO CANCEL IF PART TOO SMALL
          B      WAIT    GO TO WAIT UNTIL PAGES FREED
          B      CANCL   GO TO CANCEL IF PFIX ADDRESSES INVALID
HERE      BAL    14,ARTN GO TO ARTN
          PFREE  ARTN,ARTNEND+2 FREE ROUTINE AFTER EXECUTION
ARTN        (time dependent processing which cannot be
          paged out during execution)
ARTNEND   BR     R14     RETURN
NOPAGES   LA     R1,OPCCB
          EXCP   (1)     WRITE MESSAGE TO OPERATOR
          WAIT   (1)     WAIT FOR COMPLETION
CANCL     CANCEL ALL
WAIT        (routine to free other pages)
END       EOJ
OPCCB     CCB    SYSLOG,OPCCW
OPCCW     CCW    X'09',MSG,X'20',61
MSG       DC     CL32'AM CANCELING PLEASE ENLARGE REAL'
          DC     CL29'PARTITION AND RESTART THE JOB'
          •
          •
```

Figure 10-1. PFIX and PFREE example.

processing. The pages are given the highest page-out priority, with the result that other pages which are still needed for immediate reference are kept in storage. Except when the RELPAG macro is in operation, the contents of any pages written out are saved.

Page-in in Advance: The PAGEIN macro allows you to request that one or more pages be paged-in in advance, in order to avoid page faults when the specified pages are needed in real storage. Unlike the PFIX macro, the PAGEIN macro does not fix pages and, therefore, does not guarantee that the paged-in pages are still in real storage when they are needed. If the specified pages are already in real storage when the macro is issued, they are given the lowest priority for page-out.

Balancing Teleprocessing: The TP (teleprocessing) balancing function is intended for those TP users who experience problems with response time when they have concurrent batch processing in a paging environment. TP response time can then be improved at the expense of less throughput in the batch partition(s).

Supervisor support for this function is generated automatically when you specify your teleprocessing access method in the TP parameter of the SUPVR macro. This function must, however, be invoked by the operator via the TPBAL command (see *VSE/Advanced Functions Operating Procedures*); you can then use the TPIN and TPOUT macros.

The TPIN macro deactivates one or more partitions and is issued by the teleprocessing subsystem to request preferred access to system resources. The request is ignored in each of the following cases:

- The operator has not made TP balancing active by means of the TPBAL command.
- None of the partitions specified in the TPBAL command contains a program running in virtual mode.
- The only partition that could be affected by TP balancing is the partition that issued the TPIN request.
- There is no paging in the system.

The TPIN macro must always be used in conjunction with the TPOUT macro.

The TPOUT macro is issued by the teleprocessing subsystem, to inform the supervisor that the teleprocessing subsystem has no further processing to do for the time being. The purpose of this macro is to release system resources that were exclusively used by the teleprocessing subsystem and to make them available for general use.

Failure to issue the TPOUT macro can cause considerable and unnecessary performance degradation in the batch partition(s). The operand field is ignored.

It is not recommended that you use the TPIN and TPOUT macros in your teleprocessing application programs. Use them instead in the telecommunications access methods and data base/data communication interface programs such as the IBM program product CICS/DOS/VS. The latter, when running under VSE, supports the TPIN/TPOUT interface with the supervisor.

Dynamic Allocation of Virtual Storage

With the GETVIS and FREEVIS macros, a program can dynamically acquire and release blocks of storage in the GETVIS area of the partition in which the program is running.

A minimum GETVIS area is always reserved in a partition as long as a job in that partition runs in virtual mode. The minimum can be enlarged by the SIZE command. For a discussion of the SIZE command, refer to the *VSE/Advanced Functions System Management Guide.*

For any partition, the SIZE *parameter* may be specified in the EXEC job control statement: it overrides a permanent value (minimum or set by the SIZE *command*) and sets aside GETVIS storage for the duration of the job step.

The SVA (Shared Virtual Area) contains a GETVIS area, too. However, that GETVIS area is reserved for system use.

Program Communication

For each partition, the supervisor contains a storage area called the communication region. Through the available macro support, your program can read information that is stored in that area and modify one specific field, the user area, of the communication region.

Figure 10-2 shows the portion of the communication region containing information of interest. This information is also described below.

Field Length	Information
8 bytes	Calendar date. Supplied from the system date whenever the JOB statement is encountered. Depending on the system default or the specification in the STDOPT command, the format of the date is either mm/dd/yy of dd/mm/yy where mm is month, dd is day, yy is year. This date can be temporarily overridden by a DATE statement.
4 bytes	Reserved
11 bytes	User area for communication within a job step or between job steps. All 11 bytes are set to zero

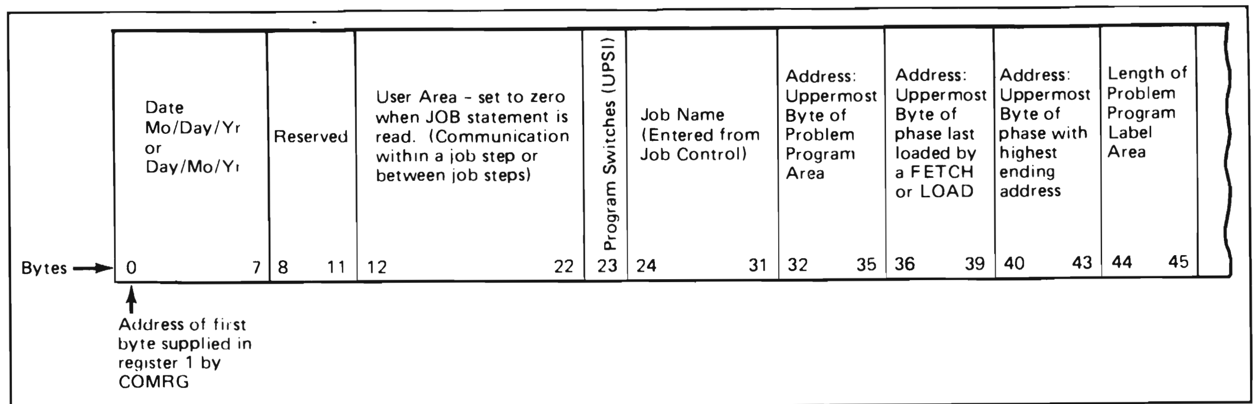


Figure 10-2. Partition Communication Region.

	whenever the JOB statement for a job is encountered.
1 byte	UPSI (user program switch indicators). Set to binary zero when the JOB statement for the job is encountered. Initialized by the UPSI job control statement.
8 bytes	Job name as found in the JOB statement for the job.
4 bytes	Address of the partition's uppermost byte available to the problem program.
4 bytes	Address of the uppermost byte of the phase placed in the program area by the last FETCH or LOAD macro in the job.
4 bytes	Highest ending main storage address of the longest phase, starting with the same 4 characters as the root phase (operand on the EXEC statement) and residing in the same core image library as the root phase. If the root phase is in the SVA, the partition start address plus 2K will be used.
2 bytes	Length of program label area

The COMRG and MVCOM macros allow your program to check the communication region and to modify the user area, respectively.

The COMRG macro places the address of the partition communication region in register 1. Your program can read any portion of its own partition's communication region by using that register as a base register; however, IBM guarantees the format and use of only those fields of a partition communication region which are shown in Figure 10-2.

The MVCOM macro modifies the contents of bytes 12 through 23 of the communication region. The following example shows how to move three bytes from the symbolic location DATA into bytes 16 through 18 of the communication region:

```
MVCOM 16,3,DATA
```

The JOBCOM macro makes communication possible between jobs or job steps of a partition. Information being communicated is stored in a 256-byte area. The system provides such an area for each partition. Through the JOBCOM macro, a program either moves information into that area or retrieves information that had previously been stored there by another program. The area remains unaltered from one job (or job step) to the next. Unless it is modified through execution of the JOBCOM macro, the contents of the area remain unchanged over any number of jobs.

The program that issues the JOBCOM macro must provide a register save area 18 fullwords long. Prior to execution of the macro, register 13 has to point to that save area.

The following example shows how 8 bytes of information are stored into the first 8 bytes of the system-supplied area. The other 248 bytes of that area remain unchanged.

```
          •
          •
          LA      13,JCOMSAVE
          JOBCOM  FUNCT=PUTCOM,                X
                  AREA=JCOMINFO,LENGTH=8
          •
          •
JCOMSAVE  DS      18F
JCOMINFO  DC      C'ABCDEFGH'
          •
          •
```

Assigning and Releasing I/O Units

Programmer logical units can be released from within a program by the RELEASE macro. RELEASE may be used only for units that are assigned to the partition in which the macro is issued.

Execution of the macro unassigns the specified programmer logical unit or units, unless they are assigned permanently. For more information about logical unit assignment, see *VSE/Advanced Functions System Management Guide.*

Be sure your program informs the system operator via a message that the assignment was released.

Assigning and Releasing Tape Drives

A magnetic tape drive that is not tied up to one of the system's partitions by a previous assignment of a logical unit can be made available to the program dynamically using the ASSIGN macro. This macro is also used for dynamic release of the drive when the program has no further use for it. The dynamic assigning and releasing of a tape drive may be particularly useful in long running, complex applications that require a magnetic tape volume only for a short period of time (for instance, for storing intermediate processing results).

The assignment is temporary. If the assignment is not released with the ASSIGN macro, it is reset along with all other temporary assignments when the next EOJ statement (/&) is encountered. If the ASSIGN macro is used to release a tape drive, it must be ensured that the logical unit number of the unit to be released is still present in the parameter list used by the macro. The layout of the parameter list can be interpreted with the help of the mapping DSECT generated by the ASPL macro. The following skeleton example shows how a tape drive can be assigned and released dynamically.

```
ASSIGN   ASPL     DSECT=YES                 Generate mapping DSECT for
*                                           parameter list
                    •
                    •
         XC ASPLX,ASPLX                     Clear parameter list
         LA RX,ASPLX                        Establish addressing and mapping
         USING ASSIGN,RX                    to the parameter list
         MVI  ASGFUNCT,ASGTPT               Indicate assign
*
         ASSIGN ASPL=(RX),SAVE=SAVEAREA     Temporarily assign any
*                                           available programmer logical unit
*                                           to any available tape drive
         MVC ASGLUNO,→DTF                   Put logical unit
                    •                       into tape DTF (e.g. DTFDI)
                    •
         OPEN/GET/PUT/CLOSE                 Perform desired I/O functions
                    •
                    •
         LA RX,ASPLX                        Ensure addressability
         MVI ASGFUNCT,ASGUAP                Indicate unassign
         ASSIGN ASPL=(RX),SAVE=SAVEAREA     Free tape and logical unit
                    •                       (ASGLUNO still intact)
                    •
ASPLX    DS CL(ASGLNG)                      Define parameter list
SAVEAREA DS 18F                             Define save area
```

Timer Services and Exit Control

Timer Services

VSE provides optional timing facilities which use hardware features that are standard in most VSE supported processors. Those timing facilities are:

1. The TOD (time-of-day) clock, used to determine the current time.
2. The IT (interval timer), which enables a time interval (measured in seconds or 1/300ths of a second) to be preset so that a program can be notified when the time interval has expired.
3. The TT (task timer), which allows a timer interval (measured in milliseconds) to be preset by the main task so that the task can give control to an exit routine when the specified time interval has elapsed. This discrete time interval is decremented only when the main task is executing.

Time of Day Clock

The TOD (time-of-day) clock is a standard high-resolution hardware feature. Any program executing under VSE can obtain the time of day by issuing the GETIME macro. This causes VSE to present to your program the time of day in accordance with your specification in the macro in one of the following formats:

- As a packed decimal number in the form hhmmss (where hh = hours, mm = minutes, ss = seconds).
- As a binary number in seconds.
- As a binary number in 1/300 seconds.

Interval Timer

Any program (or task) can set a real time interval, in seconds, or 1/300 of a second, by using a SETIME macro. The maximum valid interval is 55924 — equivalent to 15 hours, 32 minutes, 4 seconds, or 8388607 — equivalent to 7 hours, 46 minutes, 2 sec-

onds (approximately) — if expressed in 1/300 of a second. Expiration of the specified interval causes an external interrupt.

When the interrupt occurs and the program has established linkage to a timer exit routine via a STXIT IT macro, the program is interrupted and control is transferred to the timer exit routine.

At the end of the timer exit routine (statement EXIT IT), control is transferred to the point of interruption.

Note: This support is independent of the time-of-day clock; the use of the interval timer and of GETIME have no effect on one another.

Waiting for a Time Interval to Elapse: When processing depends on the expiration of a time interval, a WAIT macro can be issued to suspend processing until the interval set by a SETIME macro has elapsed.

The SETIME macro passes to the supervisor the name of the timer event control block (defined by a TECB macro) to be posted when the specified interval has elapsed. The WAIT macro specifies the same TECB and passes control to the supervisor, which allows another task to execute in the meantime. When the timer interrupt occurs, the event bit in the TECB is turned on and the task that has issued the SETIME and WAIT macros is made ready to proceed. Figure 10-3 illustrates a program that waits for a time interval to expire.

Getting the Unexpired Time: After a SETIME macro has been issued, the task issuing the macro can obtain the unexpired part of the interval by issuing a TTIMER macro. This macro returns the residual time (seconds) without disturbing the interval timer function.

If the TTIMER macro includes the operand CANCEL, a previously issued SETIME macro is canceled.

Task Timer

The task timer support can be generated only for the main task of a specific partition.

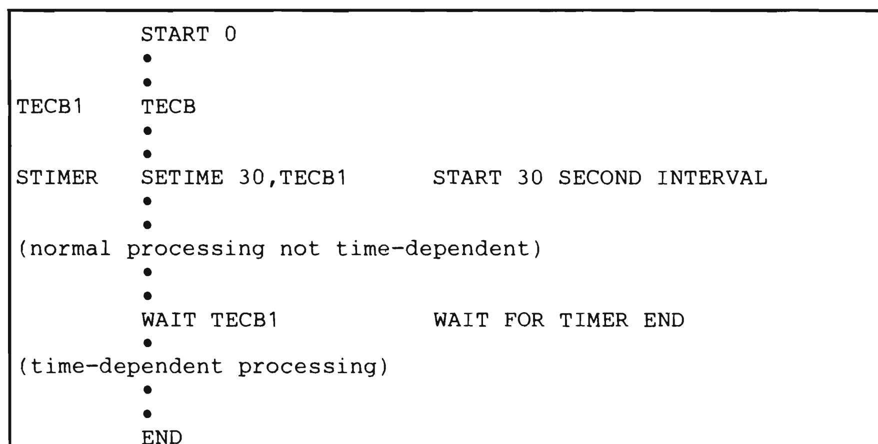
The main task sets the desired time interval by specifying it, in milliseconds, in the operand of the SETT macro; or by putting the desired interval, in milliseconds, in binary, in the register specified as the operand of the SETT macro. The maximum valid interval is 21,474,836 milliseconds. The time interval is decremented only when the main task is executing.

When the specified time interval has elapsed, the task timer routine supplied in the STXIT TT macro is entered. If a routine was not supplied to the supervisor by the time the interrupt occurs, the interrupt is ignored.

When a program is restarted from a checkpoint, the timer interval set by the SETT macro is not restarted. The task timer support is further discussed under "Task-Timer User Exit" later in this chapter.

Obtaining or Canceling the Time Remaining: The task using the task timer can issue a TESTT macro to test how much time remains in the time interval set by an associated SETT macro. The time remaining in the interval is returned, expressed in hundredths of milliseconds - in binary, in register 0.

The time remaining in the interval can be canceled by specifying CANCEL as the operand of a TESTT macro. This prevents the task timer exit routine from being entered.



```
         START 0
         •
         •
TECB1    TECB
         •
         •
STIMER   SETIME 30,TECB1      START 30 SECOND INTERVAL
         •
         •
(normal processing not time-dependent)
         •
         •
         WAIT TECB1           WAIT FOR TIMER END
         •
(time-dependent processing)
         •
         •
         END
```

Figure 10-3. Skeleton example of a program in which a 30-second interval must elapse before special processing is performed.

Linkages to User Exit Routines

Linkage to a user exit routine can be established through the STXIT macro. The STXIT macro specifies the condition under which control is to be passed to the user-written exit routine named in the macro. Figure 10-4 shows the conditions that you can request to cause control to be transferred, the requesting code you provide as the first operand, and the type of user exit routine normally associated with the exit condition.

Condition	STXIT Operand	User Routine
Interval Timer External Interrupt	IT	Interval timer exit
Task Timer Interrupt	TT	Task timer exit
Abnormal Termination of Problem Program	AB	Abnormal termination exit
Program Check Interrupt	PC	Program check exit
Operator Communications Interrupt	OC	Operator communications exit

Figure 10-4. Condition, request codes and exit routines.

Interval-Timer User Exit

If a certain processing is required when a specified time interval has elapsed, the STXIT IT macro can be used to establish and terminate linkage to the appropriate interval-timer exit routine and subsequently, when this routine completes processing, an EXIT IT macro to return control to the next sequential instruction in the main routine.

Note: If the program issuing the STXIT IT macro is an application program using ACF/VTAME, the exit will not be taken while ACF/VTAME is processing any request on behalf of the application program; the exit will be taken when ACF/VTAME has completed the program's request.

Figure 10-5 shows the application of an STXIT IT macro for entering a checkpoint routine every half hour during processing. Notice that in this example the user's interval timer exit routine need not be fixed in real storage; since there is no real-time dependency, the results cannot be influenced by the paging activity.

Multitasking Considerations: The main task or any subtask in a partition or both may issue a SETIME macro. Each may also issue a STXIT macro to establish linkage to a common user exit routine provided that this routine is reenterable and that each task has its own unique save area. Figure 10-6 illustrates this approach.

Task-Timer User Exit

Task timer support may be generated via the FOPT supervisor generation macro.

Linkage to a routine for processing a task timer interrupt is established and terminated by the STXIT TT macro. This linkage must be established before an interrupt occurs, or the interrupt will be ignored. The macro can only be issued by the main task of the partition owning the task timer.

The task timer exit routine returns control to the supervisor by issuing an EXIT TT macro. When the EXIT TT macro is processed, the interrupt status information and the contents of registers are restored from the save area. It is important, therefore, that the contents of the save area specified in the associated STXIT TT macro not be destroyed.

Abnormal-Termination User Exit

The STXIT AB macro establishes or terminates linkage to a user routine that is entered whenever the issuing program is to be terminated for any reason other than a self-requested termination. In this routine, you can do any necessary housekeeping such as closing files and writing messages before the program is terminated. You cannot use any of the macros STXIT, SETIME, and SETT within your abnormal termination exit routine. Also, do not use LOCK or ENQ in an AB exit routine for a resource that is held by the main task, since a dead lock may occur.

If the exit routine executes as a subtask, you cannot recover from an error, and it must end with a CANCEL, DETACH, DUMP, JDUMP, or EOJ macro. However, if the exit routine is part of the main task, it may be preferable to continue processing. The EXIT AB macro can be used for this purpose.

Program-Check User Exit

The linkage established by the STXIT PC macro instruction provides entry to a user exit routine for handling any program check interrupt that is not caused by a page fault. The routine can analyze the interrupt status information and the contents of the general registers stored in the user's save area.

If an error condition caused the interrupt, your exit routine can correct the error or decide to ignore it, depending on the severity of the error. Your routine can either return control to the interrupted program or request termination of the program.

```
TIMECHK   START 0
          STXIT IT,TIMINTR,TIMSA  SET UP LINK TO TIMER RTN
          MVI   STATSW,X'80'      SET SW FIRST TIME THROUGH
          SETIME 1800             TAKE CHCKPNTS EVERY 30 MIN
          •
          •
PROCESS   (perform normal processing)
          •
          •
          CLI   STATSW,X'40'      CHECK FOR TIMER INTERRUPT
          BNE   PROCESS           IF NOT CONT PROCESSING
          B     CHKPTR            IF SO TAKE CHECKPOINT

* TIMER INTERRUPT ROUTINE
TIMINTR   MVI   STATSW,X'40'      SHOW INTERRUPT
          EXIT  IT                RETURN TO INTERRUPTED PNT

* CHECKPOINT ROUTINE
CHKPTR    (do necessary processing before taking checkpoint)
          •
          •
          CHKPT  SYS001,RSTRTR,,,,DSKELE  TAKE CHECKPOINT
          LTR    R0,R0            CHECK IF CHECKPOINT OK
          BE     ERROR            GO TO ERROR RTN IF NOT
          ST     R0,CHKPTNR       PUT CHKPT NUMBER IN MSG
          LA     R1,MSG1          GET ADDRESS OF RIGHT MSG
          STCM   R1,7,OPCCW+1     PUT MSG ADDR IN CCW
          LA     R1,OPCCB         MESSAGE CCB
          EXCP   (1)              WRITE MESSAGE TO OPERATOR
          WAIT   (1)              WAIT FOR COMPLETION
          MVI    STATSW,X'80'     RESET CHECKPOINT SWITCH
          SETIME 1800             RESET TIMER
          B      PROCESS          RESUME PROCESSING

* RESTART ROUTINE
RSTRTR    STXIT IT,TIMINTR,TIMSA  RESTORE TIMER INTERR LINK
          SETIME 1800             SET TIMER
          •
          •
          (restore everything saved in checkpoint)
          •
          •
          B       PROCESS         START PROCESSING

* MESSAGE ROUTINE FOR INVALID CHECKPOINT
ERROR     LA      R1,MSG2         GET ADDRESS OF ERROR MSG
          STCM    R1,7,OPCCW+1    PUT MSG ADDR IN CCW
          LA      R1,OPCCB        LOAD MESSAGE CCB
          EXCP    (1)             WRITE MESSAGE TO OPERATOR
          WAIT    (1)             WAIT FOR COMPLETION
          CANCEL  ALL             CANCEL PROGRAM
          •
          •
END       EOJ

* CONSTANTS
TIMSA     DS      9D
OPCCB     CCB     SYSLOG,OPCCW
OPCCW     CCW     X'09',MSG1,X'20',80
MSG1      DC      CL16'CHECKPOINT NR'
CHKPTNR   DS      F
          DC      CL60'HAS BEEN TAKEN'
MSG2      DC      CL80'CHECKPOINT FAILED JOB IS CANCELED'
STATSW    DS      X
          END
```

Figure 10-5. Example of using the interval timer for taking a checkpoint every half-hour.

```
MAINTASK START 0
         •
         •           (set up addressability)
         STXIT IT,STRTER,MISKSA
         SETIME 300              MAIN TASK TIMER TO 5 MINS
         ATTACH SUBTASK1,SAVE=SAV1
         ATTACH SUBTASK2,SAVE=SAV2
         •
         •
* IT USER EXIT ROUTINE
STRTER   (reenterable routine)
         •
         •
         EXIT IT

SUBTASK1 STXIT IT,STRTER,STSK1SA USE SAME EXIT ROUTINE
         SETIME 400              SET TIME INTERVAL
         •
         •
         DETACH

SUBTASK2 STXIT IT,STRTER,STSK2SA USE SAME EXIT ROUTINE
         SETIME 500              SET TIME INTERVAL
         •
         •
         TTIMER CANCEL           CNCL INTRVL THIS TSK ONLY
         •
         •
         DETACH

MTSKSA   DS   9D
STSK1SA  DS   9D
STSK2SA  DS   9D
SAV1     DS  16D
SAV2     DS  16D
```

Figure 10-6. Skeleton example of multitask linkage to a common IT exit routine.

Having a user's program check routine can be useful when it is known that one or more programs may be checked by processing errors that are insignificant to the results or can be corrected easily. Figure 10-7 shows an exit routine for recovering from a program check caused by attempting to divide by zero. In this example, any other errors causing a program check result in the user save area being dumped before the job is terminated.

Operator-Communication User Exit

A direct communications link between the operator and a program can be established by issuing an STXIT OC macro instruction. It may be issued only by the main task in any partition.

To initiate communication the operator presses the Request key. This causes an I/O interrupt to occur. When the attention routine identifier AR appears, the operator enters MSG followed by the partition identifier (such as BG or F2), which sets the linkage to the user's operator-communication exit routine.

The operator communication exit routine may perform any processing. A typical application is the taking of a checkpoint record in a program that has to be canceled in order to start a high-priority job that has just been handed in; the checkpointed program can then be restarted later on.

Note: If the program issuing the STXIT IT macro is an application program using ACF/VTAME, the exit will not be taken while ACF/VTAME is processing any request on behalf of the application program; the exit will be taken when ACF/VTAME has completed the program's request.

Requesting Storage Dumps

Whenever a program is to be terminated by the system for a reason other than a normal end-of-job condition, and especially after a program check interrupt, a printout of all or part of the storage area occupied or used by the program at that moment is a useful aid for tracing the cause. For guidance on reading and interpreting the printout, see *VSE/Advanced Functions Serviceability Aids and Debugging Procedures.*

VSE provides several macros to request such a printout. These macros may be used, for example, at the end of a user's exit routine for handling an abnormal termination condition.

The following is a summary of the functions of macros that request storage dumps:

DUMP The macro dumps, in hexadecimal format, the contents of the supervisor area,

```
DIVTEST  CSECT
         •
         •
                   (set up addressability)
         •
         STXIT PC,PCRTN,PCSAV     SET UP PROGRAM CHECK LINK
         •
         •
         LM    R2,R3,DIVIDEND     LOAD FOR DIVIDING
         D     R2,DIVISOR         DIVIDE
         •
         •
* USER'S PROGRAM CHECK ROUTINE
PCRTN    SR    R5,R5              CLEAR REGISTER 5
         CL    R5,DIVISOR         CHECK FOR ZERO DIVISOR
         BNE   CANCELR            IF NOT CLEAR FILES & CNCL
         •
         (special recovery routine)
         •
         •
         EXIT   PC                RETURN TO NORMAL PROC
CANCELR  PDUMP  PCSAV,PCSAV+71    DUMP SAVE AREA
         •
         (close files and do other housekeeping)
         (equates and storage definitions)
         •
         CANCEL ALL
```

Figure 10-7. Skeleton example of a routine for processing a program check caused by zero division.

or the contents of some supervisor control blocks, depending on the parameters specified in the STDOPT job control command or on the // OPTION job control statement in a specific job step. (For details about the dump options you can specify in the STDOPT command or on the // OPTION statement, refer to *VSE/Advanced Functions System Control Statements.*) In addition, the DUMP macro dumps the storage contents of the partition, and all registers. The job step is terminated if the macro is issued by the main task; but if issued by a subtask, then only that subtask is detached.

JDUMP — This macro causes the same areas to be dumped as for a DUMP macro, but terminates the entire job.

PDUMP — This macro provides a hexadecimal dump of the general registers and of the storage area between the addresses specified by two operands. After execution of this macro, processing continues at the next sequential instruction.

A PDUMP macro may, therefore, be issued several times in a program to provide dumps of selected storage fields for examination at different stages of the program's execution.

Regardless of which of the above macros are used, the resulting dump is always produced on the SYSLST, which may be assigned to a printer, to disk, or to magnetic tape.

If the device assigned to SYSLST is a 3211 printer and indexing was used prior to taking a dump, a certain number of characters on every line of the printed dump may be lost.

To avoid losing characters, ensure that an FCB (forms control buffer) image without an indexing byte is loaded into the printer's FCB before you issue the dump generating macro in your problem program. You can use the LFCB macro for this purpose.

Ending a Task or a Job

Normal End of the Main Task

The normal way of ending the main task (which might be the only program executing in the partition) is to issue the EOJ macro.

Through the EOJ macro, a program informs the supervisor that the job step is to be terminated. At this time, subtasks should no longer be attached. If nevertheless they are, issuance of the EOJ macro by the main task is considered an abnormal termination condition for the subtask. In case the subtask provided an STXIT AB routine, this routine is entered.

Normal End of a Subtask

The normal way of ending a subtask is by issuing the DETACH macro. Either the subtask terminates itself, or the main task does it by making use of an operand in the DETACH; the operand points to the subtask's save area. For more information about using the DETACH macro, see the section "Multitasking" later in this chapter.

To detach itself, the subtask may also issue the EOJ macro. All other tasks in the same partition continue processing; the job step is not terminated as was the case with an EOJ issued in the main task.

Program-Requested Abnormal Ends

To terminate a task under abnormal conditions, you may use either the DUMP or JDUMP macro or the CANCEL macro.

The macros DUMP and JDUMP were discussed previously in section "Requesting Storage Dumps". When issued by the main task, the DUMP or JDUMP macro causes the job step (job) to be terminated. If the macro is issued by a subtask, only this subtask gets detached.

The CANCEL macro provides another way of terminating abnormally. As with DUMP or JDUMP, a CANCEL issued in the main task terminates processing of all tasks within the partition. A CANCEL issued in a subtask detaches only the subtask, unless the ALL was specified in the CANCEL macro; a CANCEL ALL in a subtask causes all processing in the partition to terminate.

Using the EXIT Macro

EXIT is another macro used to end a portion of your program. However, it should not be confused with the task-terminating macros EOJ, DETACH, DUMP, JDUMP, or CANCEL. Via the EXIT macro, a user exit routine (discussed previously in section "Time Services and Exit Control") causes control to return to the point of interruption within the main-line routine; thus the task continues processing.

Checkpointing a Program

The progress of a program that performs considerable processing in one job step should be protected against destruction in case the program is canceled later. The system provides support for taking up to 9999 checkpoint records in a job. Through this facility, information can be preserved at regular intervals and in sufficient quantity to allow restarting a program at an intermediate point.

The CHKPT macro stores the checkpoint record on a magnetic tape or disk.

The RSTRT job control statement restarts the program from the last or any specified checkpoint taken before cancelation. For full details on using this statement, see *VSE/Advanced Functions System Control Statements.*

Choosing a Checkpoint

The most important criterion for a checkpoint decision is a minimum of necessary housekeeping before the checkpoint record can be taken. The possibility of an error occurring either in the checkpoint routine or at restart is then also minimal. Checkpoints cannot be taken by a subtask or by a main task with subtasks attached. Therefore, when multitasking, checkpoints should be avoided where a number of subtasks must first be detached.

A successful checkpoint record taken immediately after opening all files used by the program indicates that processing can safely proceed. If such a checkpoint record is invalid, however, then the program should be canceled.

Other checkpoint records may be taken at logical breaks in data, such as at the end of a reel of magnetic tape.

After a CHKPT macro is successfully executed, register 0 contains the checkpoint number in unpacked decimal format; if CHKPT macro execution is unsuccessful, register 0 contains zero, and the reason for the failure is printed on SYSLOG.

Timing the Entry to the Checkpoint Routine

Having decided where your program can conveniently be checkpointed, you may consider entering the checkpoint routine only if a certain time interval has elapsed since the previous checkpoint record was taken.

By issuing a SETIME macro after an STXIT IT macro has established linkage to a user exit routine that sets a switch and returns, the main program can test this switch and then branch to the checkpoint routine or continue processing according to whether the switch is set or not. An example of this technique can be found in Figure 10-5.

By issuing an STXIT OC macro instruction, it is also possible to have checkpoint records taken at convenient points on command from the operator. This method is illustrated by Figure 10-8.

```
CHKPTRTN CSECT
         (set up addressability)
         STXIT  OC,OCMSG,OCSAV SET UP LINKAGE FOR OC MSG
         •
         •
         MVI    SW1,X'40'      SET CHECKPOINT SWITCH
         OPEN   (RDISKOUT),(RCHKPTF)  OPEN FILES
* DTF ADDRESSES FROM MAIN ROUTINE ARE USED
         BAL    RLINK,CHECKPT  TAKE TEST CHECKPOINT
         •
         •
START    (normal processing)
         •
         •
         CLI    SW1,X'40'      SEE IF OPERATOR SENT MSG
         BE     START          CONTINUE IF NOT
* THE FOLLOWING IS THE CHECKPOINT ROUTINE ENTERED ON
* A SIGNAL FROM THE OPERATOR
         STD    F0,REG0        SAVE FLOATING POINT REGS
         STD    F2,REG2
         STD    F4,REG4
         STD    F6,REG6
         CHKPT  SYS011,(RSTRTR),,,,(RCKPTF)  TAKE CHKPTS
         LTR    R0,R0          TEST IF SUCCESSFUL
         BZ     CANCEL         CANCEL IF NOT
         MVI    SW1,X'40'      RESET CHECKPOINT SWITCH
         B      START          RETURN TO NORMAL PROCESSING
         •
         •
         (equates)
OCMSG    MVI    SW1,X'80'      SET CHECKPOINT SWITCH
         EXIT   OC             RETURN TO POINT OF INTERR
CHECKPT  CHKPT  SYS011,(RSTRTR),,,,(RCHKPTF)
         LTR    R0,R0          SEE IF CHECKPNT SUCCESSFUL
         BNZ    0(RLINK)       RETURN IF TAKEN
CANCEL   CANCEL ALL            CANCEL IF CHECKPOINT FAILED
STRTR    STXIT  OC,OCMSG,OCSAV RESTORE LINKAGE
         LD     F0,REG0        RESTORE FLOATING POINT REGS
         LD     F2,REG2
         LD     F4,REG4
         LD     F6,REG6
         B      START          RESTART PROGRAM
END      EOJ
REG0     DS     D
REG2     DS     D
REG4     DS     D
REG6     DS     D
OCSAV    DS     9D
SW1      DS     X
         •
         •
         (equates)
         •
         •
         end
```

Figure 10-8. Skeleton example of a routine for checkpointing a program on operator command.

```
// JOB   CHECKPOINT    (the JOBNAME must be the same as before)
// ASSGN ...           (all ASSGNs must be renewed)
// ASSGN ...           (new assignments may be made)
// ASSGN ...
// RSTRT SYS001,1111,CHKPTF
```

Figure 10-9. Example of job control statements for restarting a checkpointed job from checkpoint 1111.

Saving Data for Restart

Besides the information stored by the CHKPT macro, certain data must usually be saved by the user's checkpoint routine in order to facilitate a successful restart. This may include the contents of floating point registers, any linkage that was established by a STXIT or a SETPFA macro, the interval value for a SETIME macro, and the program mask in the problem program PSW.

Restarting a Checkpointed Program

When a checkpointed program is to be restarted, the partition must start at the same location as when the program was checkpointed, and its end address must not be lower than the end address specified at checkpoint time.

If any of the pages of a virtual mode program were PFIXed when the checkpoint was taken, for S/370 mode, the real partition must also start at the same location or lower and its end address must be at least as high as at checkpoint time; for ECPS: VSE mode, the amount of PFIXable storage set by the ALLOCR command must be as high as at checkpoint time. The pages that were fixed when the checkpoint was taken are refixed by the system when the program is restarted.

Unless you reset all linkages to SVA phases yourself, the contents and location of the modules in the SVA, when restarting, must also be the same as when the program was checkpointed. The SDL must be identical if the restarted program uses a local directory list (for example, one that was generated by the GENL macro).

To restart a checkpointed program, use the job control statement RSTRT. In addition to the job control statements originally used to execute the program, label information for the checkpoint file must be submitted. An example of appropriate job control statements for restarting a checkpointed program on disk is illustrated in Figure 10-9.

Information That is Saved

When the CHKPT macro is issued, the following information is saved:

- The general registers.
- Problem-program-related information from the partition communication region.
- PFIX-information.
- All DASD file protection extents attached to logical units that belong to the checkpointed program.
- Information related to the IBM 3800 Printing Subsystem.
- The problem program area.

Information That is Not Saved

- The floating-point registers. (If needed, these registers should be stored in the problem program area before issuing CHKPT, and restored in your restart routine. If needed, any XECBs defined by your program must be also defined by your restart routine.)
- Any linkages to your routines set by the STXIT or SETPFA macros. (If needed, STXIT or SETPFA must be used in your restart routine.)
- Any timer values set by the SETIME macro. (If needed, SETIME must be used in your restart routine.)
- The program mask in your program's PSW. (If anything other than all zeros is desired, the mask should be reset in your restart routine.)

Considerations for DASD, Diskette, MICR, and 3886 Files

DASD or diskette system input or output files (SYSIPT, SYSLST, etc.) must be reopened at restart time. In your restart routine, you must be able to identify the last record processed before the checkpoint.

For MICR files, your program must disengage the device and process all follow-up documents in the document buffer before taking a checkpoint. MICR files require the DTFMR supervisor linkages to be initiated at restart time. Do this by reopening the MICR file in your restart routine that clears the document buffer.

For 3886 files, the SETDEV macro must be issued at restart time. This ensures that the proper format record will be loaded into the 3886 if the job must be restarted. If the job processing the 3886 file uses line marking with reflective ink, the job cannot be restarted.

Checkpoint File

The checkpoint information must be written on disk or on an EBCDIC magnetic tape (7- or 9-track). The 7-track tape can be in either data conversion or translation mode. However, the magnetic tape unit must have the data conversion feature. On 7-track tapes, the header and trailer labels are written in the mode of the tape and the records are written in data convert mode, with odd parity.

Checkpoint on Tape

You can either establish a separate file for checkpoints or embed the checkpoint records in an output file. When the file is read at a later time using LIOCS, the checkpoint records are automatically bypassed. If physical IOCS is used, you must program to bypass the checkpoint record sets.

If a separate magnetic tape checkpoint file with standard labels is maintained, the labels should be either checked by an OPEN or bypassed by an MTC command before the first checkpoint is taken. Alternate tape drives must not be assigned for a separate checkpoint file.

Checkpoints on Disk

If checkpoints are written on disk, the following must be observed:

- One continuous area on a single disk volume must be defined at execution time by the job control card necessary to define a DASD file.
- For checkpoints to be written to a CKD device, the number of required tracks is computed as follows:

  $t = n(1 + (w/25 + x/20 + y)/v + c/z)$

  where

  c= number of bytes to be checkpointed as determined by the end address operand (size of partition by default).

  n= the number of checkpoints to be retained (when the checkpoint file is full, the checkpoint file is overlaid, starting at the beginning).

  v= the number of 256-byte records per track:

for 2311	11
for 2314/2319	17
for 3330/3333	33
for 3340	20
for 3350	44

  w= maximum number of page frames which are fixed by PFIX at the time the checkpoint is taken.

  x= the number of disk extents including nonoverlapping split-cylinder extents. If split-cylinder extents overlap on the same cylinder, the number of extents counted is one. (This number is zero if DASD file protect is not used.)

  y= the number of IBM 3800 printers attached to the partition being checkpointed.

  z= the number of checkpoint records per track:

for 2311	3
for 2314/2319	6
for 3330/3333	10
for 3340	7
for 3350	16

- For checkpoints to be written to an FBA device, the number of required blocks is computed as follows:

  $b = n(1 + (w/25 + x/20 + y)v + c \times z)$

  where

  c, n, w, x, y are defined as for a CKD device, but

  v= 256 divided by the blocksize of the FBA device.

  z= the number of blocks required for one checkpoint record, that is, 1024 divided by the blocksize of the FBA device.

  For each division, the quotient is rounded to the next highest integer before multiplying by n.

- Each program can use a common checkpoint file or define a separate one. If a common file is used, only the last program using the file can be restarted.
- The checkpoint file must be opened before the CHKPT macro can be used.
- A DTFPH macro specifying MOUNTED=SINGLE and TYPEFLE=OUTPUT must be included for use by OPEN and the checkpoint routine. A CISIZE specified with the DTFPH macro is ignored by the checkpoint routine.

Repositioning I/O Files

The I/O files used by the checkpointed program must be repositioned on restart to the record you want to read or write next. The recorded checkpoint information provides no aids for repositioning unit-record files. You must establish your own repositioning aids and communicate these to the operator when necessary. Some suggested ways are:

- Taking checkpoints at a logical break point in the data, such as paper tape end of reel.
- Switching card stackers after each checkpoint.
- Printing information at the checkpoint to identify the record in process.
- Issuing checkpoints on operator demand.

Sequential DASD input, output, and work files require no repositioning.

When updating DASD records in an existing file, you must be able to identify the last record updated before the checkpoint was taken. This can be done in various ways, such as:

- Creating a history file to record all updates.
- Creating a field in updated records to identify the last transaction record that updated it. This field can be compared against each transaction at restart time.

Repositioning Magnetic Tape

Checkpoint provides some aid in repositioning magnetic tape files at restart. Files can be repositioned to the record following the last record processed at checkpoint. This section and Figure 10-10 describe the procedure.

The *dpointer* operand of the CHKPT macro points to two V-type address constants which you define in you program. The order of these constants is important.

The first constant points to a table containing the filenames of all logical IOCS magnetic tape files to be repositioned. The second constant points to a table containing repositioning information for physical IOCS magnetic tape files to be repositioned.

If the first, second, or both constants are zero, no tapes processed by logical, physical, or both types of IOCS, repectively, are repositioned.

If the tables are contained in the same CSECT as the CHKPT macro, the constants may be defined as A-type constants. You are responsible for building these tables.

Each filename in the logical IOCS table points to the corresponding DTF table where IOCS maintains repositioning information.

- Magnetic tapes with nonstandard labels should be repositioned past the labels at restart time (presumably the labels are followed by a tapemark so that forward-space file may be used).
- If a nonstandard label or unlabeled magnetic tape file is to be repositioned for reading backward, you must position the tape immediately past the tapemark following the last data record.
- Restart does not rewind magnetic tape when repositioning them.
- A multi-file reel should be repositioned to the beginning of the desired file.
- The correct volume of a multi-volume file must be mounted for restart.
- For tapes with a standard VOL label, restart writes the file serial number and volume sequence number on SYSLOG, and gives the operator the opportunity to verify that the correct reel is mounted.
- The logical IOCS can completely reposition files on system logical units (SYSIPT, SYSLST, etc.), if the tape is not shared with any other program and if you keep a physical IOCS repositioning table. However, if a system logical unit file is shared with other programs, a problem exists. Output, produced after the checkpoint, is duplicated at restart. Input records must be reconstructed from the checkpoint, or your restart routine must find the last record processed before the checkpoint.

The entries in the physical IOCS table are:

- First halfword - hexadecimal representation of the symbolic unit address of the tape (copy from CCB).
- Second halfword - number of files within the tape in binary notation. That is, the number of tapemarks between the beginning of tape and the position at checkpoint.
- Third halfword - number (in binary notation) of physical records between the preceding tapemark and the position at checkpoint.

DASD Operator Verification Table

If the *dpointer* operand of the CHKPT macro is used, you can build a table (in your own area of virtual storage) to provide the symbolic unit number of each DASD file used by your program. At restart, the volume serial number of each of these files is printed on SYSLOG for operator verification.

The entries in the DASD operator verification table must consist of the following two halfwords, in the order stated below:

1. The symbolic unit in hexadecimal notation copied from the CCB bytes 6 and 7.
2. Reserved.

There must be one entry for each DASD unit to be verified by the operator.

Program Linkage

A program may consist of several phases or routines produced by language translators and combined by the linkage editor. The CALL, SAVE, and RETURN macros are used for linkage between routines in storage and within the same or different phases. These macros, with conventional register and save area usage, allow branching from one routine to

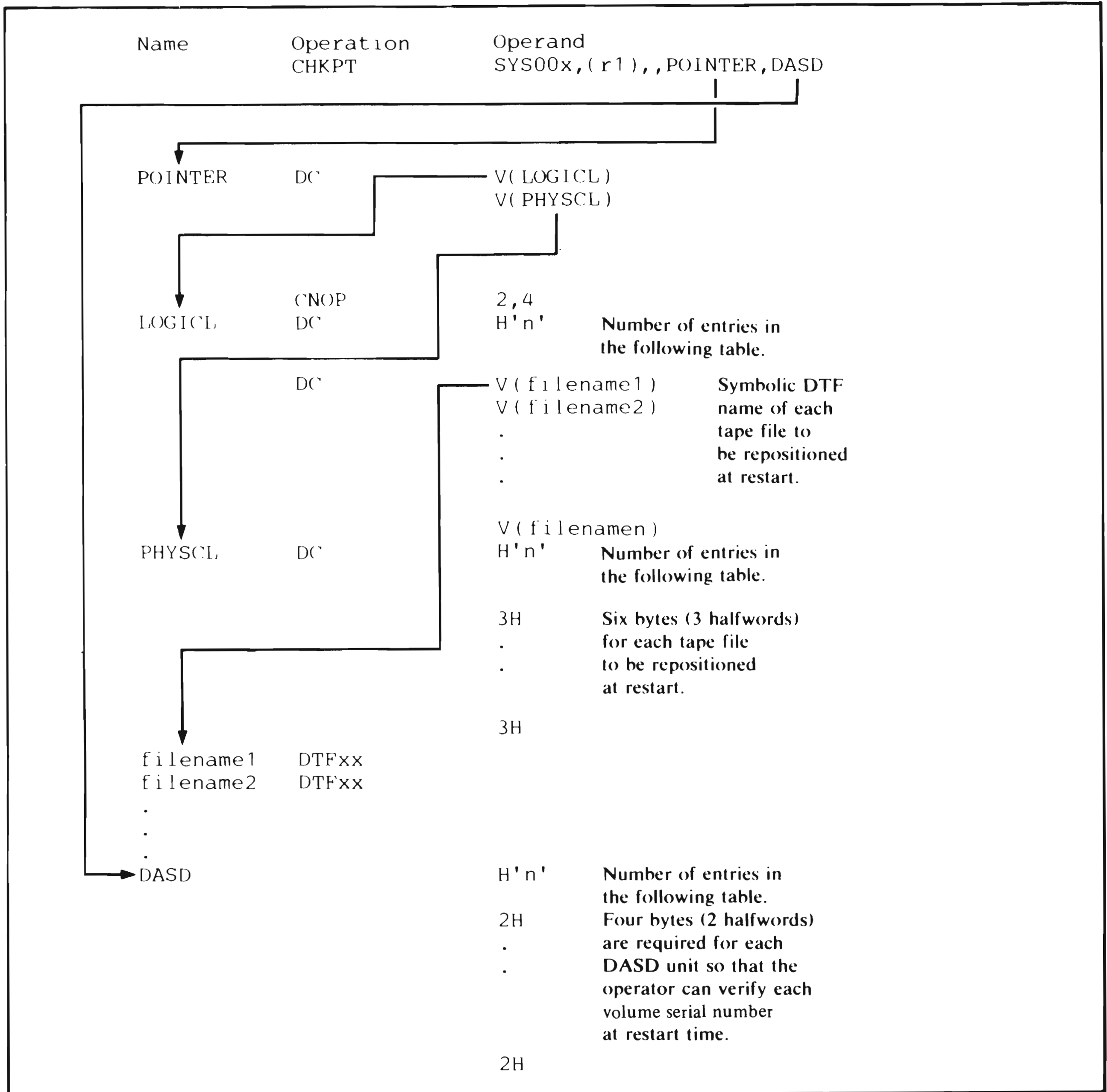


Figure 10-10. Repositioning magnetic tape and verification of DASD volume serial numbers.

another or from one phase to another and also allow passing parameters.

Passing control from one routine to another within the same phase is referred to as direct linkage. Figure 10-11 shows linkage between a main program and two subroutines. Linkage can proceed through as many levels as necessary, and each routine may be called from any level. The routine given control during the job step is initially a *called* program.

During execution of a program, the services of another routine may be required, at which time the current program becomes a *calling* program. Using Figure 10-11 as an example, when the main program passes control to B, B is a called program. When control is passed from B to C, B is the calling program and C is the called program.

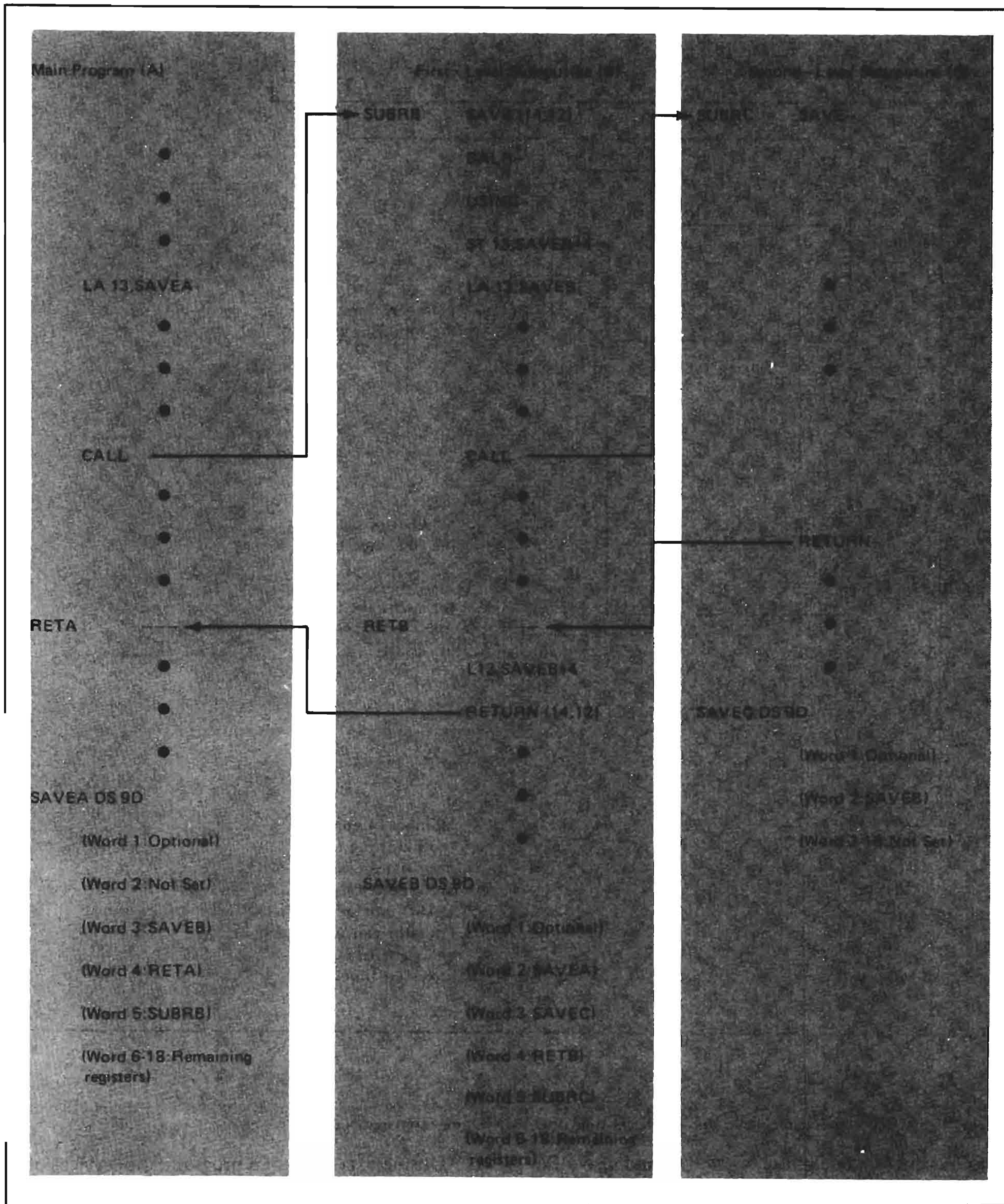


Figure 10-11. Direct Linkage.

Linkage Registers

To standardize branching and linking, registers are assigned specific roles (see Figure 10-12). Registers 0, 1, 13, 14, and 15 are known as the linkage registers. Before a branch to another routine, the calling program is responsible for the following calling sequence:

1. Loading register 13 with the address of a register save area which the called program is to use.
2. Loading register 14 with the address to which the called program will return control.
3. Loading register 15 with the address of the called program's entry point.
4. Loading registers 0 and 1 with parameters, or loading register 1 with the address of a parameter list.

Register Number	Register Name	Contents
0	Parameter register	Parameter to be passed to the called program.
1	Parameter register or Parameter list register	Parameter to be passed to the called program. Address of a parameter list to be passed to your subprogram
13	Save area register	Address of the register save area to be used by the called program.
14	Return register	Address of the location in the calling program to which control should be returned after execution of the called program.
15	Entry point register	Address of the entry point in the called program.

Figure 10-12. Linkage Registers.

Save Areas

A called program should save and restore the contents of the linkage registers, as well as the contents of any register that it uses. The registers are stored in a save area that the higher (calling) level program provided. This procedure conserves storage because the instruction to save and restore registers need not be repeated in each calling sequence.

Any calling program must provide a save area and place its address in register 13 before it executes a direct linkage. This address is then passed to the called routine. A save area occupies nine doublewords and is aligned on a doubleword boundary plus one additional word at the end if your program uses double buffering for a 2501. For programs to save registers in a uniform manner, the save area has a standard format shown in Figure 10-13 and described below.

Word	Displacement	Contents
1	0	Indicator byte and storage length; used by PL/I language program.
2	4	The address of the previous save area; that is, the save area of the subprogram that called this one (used for tracing purposes).
3	8	The address of the next save area; that is, the save area of the subprogram to which this subprogram refers.
4	12	The contents of register 14 containing the address to which return is made.
5	16	The contents of register 15 containing the address to which entry into this subprogram is made.
6	20	(The contents of) register 0.
7	24	(The contents of) register 1.
8	28	(The contents of) register 2.
9	32	(The contents of) register 3.
10	36	(The contents of) register 4.
11	40	(The contents of) register 5.
12	44	(The contents of) register 6.
13	48	(The contents of) register 7.
14	52	(The contents of) register 8.
15	56	(The contents of) register 9.
16	60	(The contents of) register 10.
17	64	(The contents of) register 11.
18	68	(The contents of) register 12.
19	72	CCB switch for double CCB support.

Figure 10-13. Save area words and contents in calling programs.

- Word 1: An indicator byte followed by three bytes that contain the length of allocated storage. Use of these fields is optional, except in programs written in the PL/I language.
- Word 2: A pointer to word 1 of the save area of the calling program. The address is passed to the called routine in register 13. The contents of register 13 must be stored by a calling program before it loads register 13 with the address of the current save area that is passed to a lower level routine (see instruction ST 13,SAVEB+4 in Figure 10-11).
- Word 3: A pointer to word 1 of the save area of the next lower level program, unless this called program is at the lowest level and does not have a save area. (The called program required a save area only if it is also a calling program.)

Thus, the called program, if it contains a save area, stores the save area address in this word.

- Word 4: The return address, which is register 14, when control is given to the called program. The called program may save the return address in this word.
- Word 5: The address of the entry point of the called program. This address is in register 15 when control is given to the called program. The called program stores the entry-point address in this word.
- Words 6 through 18: The contents of registers 0 through 12, in that order. The called program stores the register contents in these words if it is programmed to modify these registers.
- Word 19: The second CCB in the case of double buffering support for the 2501 Card Reader.

In any routine, the contents of register 13 must be saved so that the registers may be restored upon return. For purposes of tracing from save area to save area, the address of the new save area is stored. Only the registers to be modified in the routine need be saved. However, the safest procedure is to save all registers to ensure that later changes to the program do not result in the modification of the contents of a register that was not saved.

The called program would not save and restore a register if it passes a processing result in that register (see example in Figure 10-14).

CALL, SAVE, RETURN Macros

Using the CALL, SAVE, and RETURN macros greatly facilitates coding for direct linkage. Only one other instruction has to be coded: prior to the CALL statement, load the address of the calling program's save area into register 13.

The CALL macro loads registers 14 and 15 (and, if parameters were passed, register 1) appropriately and then passes control to a specified entry point in the called program.

Examples: In the following examples EX1 gives control to an entry point named ENT. EX2 gives control to an entry point whose address is contained in register 15. The examples are:

```
EX1 CALL ENT
EX2 CALL (15),(ABC,DEF)
```

Two parameters , ABC and DEF, can be accessed by the called program: after execution of the macro, register 1 points to a list of fullwords that contain the addresses of ABC and DEF.

Code in calling routine:

```
          LA      13,SAVEAREA1               A
          •
          •
          •
          CALL    SUBROUT,(PAR1,PAR2)        B
          C       12,ZERO                    J
          •
          •
          •
SAVEAREA1 DS      9D
          •
          •
PAR1      DC      C'ABCDEF'
PAR2      DS      F
ZERO      DC      F'0'
```

Code in called routine:

```
          SAVE    (14,11)                    C
          BALR    . . .                      D
          USING   . . .                      D
          ST      13,SAVEAREA2+4             E
          LA      13,SAVEAREA2               F
          •
          •
          •
      processing
          •
          •
          •
          L       13,SAVEAREA2+4             G
          RETURN  (14,11)                    H
SAVEAREA2 DS      9D                         I
          •
          •
```

```
A   Points to save area in calling
    program.
B   Passes parameters PAR1, PAR2.
C   Saves registers of calling pro-
    gram.
D   Establishes addressability.
E   Provides a backpointer to the
    calling program's save area.
F   Points to new save area (for trac-
    ing purposes).
G   Restores calling program's save
    area register.
H   Restores the specified registers
    and returns control to instruction
    at J.
I   May be smaller if no other program
    is called.
J   The called program passed the
    processing result to the calling
    program in register 12.
```

Figure 10-14. Use of CALL, SAVE, and RETURN macros.

The called program must be in virtual storage when the CALL macro is executed. The called program is brought into virtual storage in one of two ways:

1. As part of the program issuing the CALL. In this case, the CALL macro must specify an entry point by symbolic name, the linkage editor includes the phase containing that entry point in the phase containing the CALL macro, and the called program resides in storage throughout

execution of the calling program. This may waste storage if the called program is not needed throughout execution of the calling program.

2. As the phase specified by a LOAD macro. In this case, the CALL macro specifies register 15 (the entry-point register) into which the entry-point address of the program to be called was loaded. The LOAD macro must precede the first CALL for that program. Specifying register 15 preceded by a LOAD macro is most useful when the same program is called several times during execution of the calling program, but is not needed in storage throughout execution of the calling program.

The SAVE macro stores the contents of specified registers in the save area provided by the calling program. It should be written before any registers can be modified by the new program, preferably at the entry point.

The RETURN macro restores the registers whose contents were saved and returns control to the calling program. Prior to execution of the RETURN macro, register 13 must contain the address of the save area of the program to which control will be returned.

Figure 10-14 illustrates the usage of the CALL, SAVE, and RETURN macros.

Multitasking Functions

This section discusses subtask initiation and termination, resource protection, intertask communication, and DASD track protection. In addition, this section discusses the use of shared modules and files.

Subtask Initiation

The maximum possible number of subtasks that can be initiated at any one time in the system is determined by the NTASKS parameter of the SUPVR macro. The maximum number is 208. Up to 31 subtasks can run concurrently within a partition, provided the overall limitation of NTASKS is not exceeded.

The part of the subtask containing the entry point must be in storage before the subtask can be successfully attached. The block of program instructions that makes up the subtask can be part of one large CSECT program section which, possibly, includes also the main task; or the subtask can be a separate phase, in which case the phase must first be read into storage with the LOAD or CDLOAD macro before the ATTACH is issued.

Required Save Areas

The system provides a save area for the main task. The attaching task must provide a save area for the subtask it attaches. When, later on during its execution, the subtask receives an interrupt, the supervisor saves in that area the subtask's interrupt status information, the contents of the general registers, and the floating-point registers.

The first 8 bytes are reserved for the subtask name. The attaching task should fill in the subtask's name before attaching it. The name is used to identify the subtask in an abnormal termination message.

A second save area is needed if the attached task is using the attaching task's abnormal termination routine. The save area of the attaching task is then reserved for the abnormal termination of only the attaching task.

Specifying an Event Control Block

In the ATTACH macro, the name of an event control block (ECB) can be specified. The ECB is a fullword defined in the main task; for the format see Figure 10-15. Specifying an ECB is required if other tasks can be affected by this subtask's termination, or if resources are controlled by ENQ and DEQ macros within the subtask. A task may be waiting for the subtask's termination by having issued the WAIT or WAITM macro with the ECB as an operand. When the subtask terminates, the ECB gets "posted" (see Figure 10-15) by setting to one:

bit 0 of byte 2	on program-requested termination, that is, as a result of a CANCEL, DETACH, DUMP, JDUMP or EOJ macro,
bit 1 of byte 2	on abnormal termination

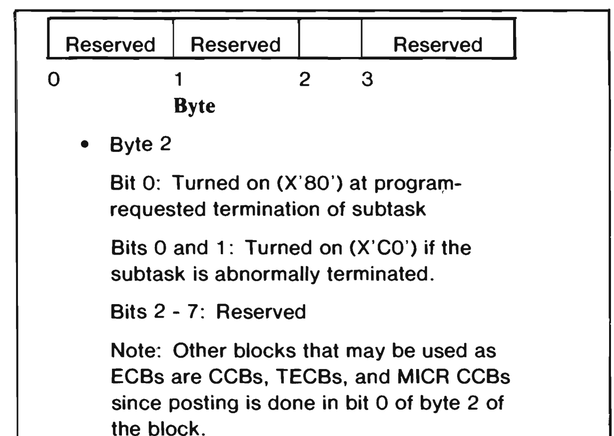


Figure 10-15. Event Control Block (ECB).

Testing for Successful Subtask Attachment

The attempt to attach a subtask may not be successful. This happens when the maximum possible number of subtasks is already attached. In this case, the main task will keep control and register 1 (main task) will contain the address of an ECB within the supervisor that will be posted when the system can initiate another subtask. Register 1 will also have the high order bit 0 on to aid the main task in testing for an unsuccessful ATTACH.

Changing the Processing Priority

If the ATTACH macro successfully initiates a subtask, the subtask is given higher priority than the main task. Among all the subtasks in a partition, a subtask can give itself the lowest processing priority of all attached subtasks within a partition by issuing the CHAP macro.

```
    MAINTASK      BALR      2,0
                  USING     *,2
                  •
                  •
                  STXIT     AB,MTABEND,MTSAVE
A                 MVC       ST1SAV(8),SUB1NAME     Initialized subtask 1 save area
B                 ATTACH    SUBTASK1,SAVE=ST1SAV,ECB=ST1ECB,ABSAVE=ST1ABSV
C                 LTR       1,1                    Test if ATTACH is successful
                  BNM       ATST1OK                BR if successful
                  WAIT      (1)                    WAIT to retry ATTACH
                  B         ATST1                  BR to retry
    ATST1OK       BCTR      0,0                    Get end of subtask 1 save area
                  ST        0,ST1SVEND             and store it
                  •
                  •
D   SUBTASK1      BALR      3,0
                  USING     *,3
E                 ST        1,MTSVAR               Store address of main task save area
                  •
                  •
F   MTABEND       STC       0,ABSVCODE             Save ABTERM code
G                 C         1,=A(ST1ABSV)              Test if subtask 1 ABTERM
                  BE        ST1ABEND               BR if YES
                  •
                  •
    ST1ABEND      EQU       *
                  •
                  •
    ST1SAV        DC        16D'0'                 Subtask 1 save area with FP regs
    ST1ABSV       DC        9D'0'                  Subtask 1 AB save area
H   ST1ECB        DC        F'0'                   Subtask 1 ECB
    MTSVAR        DC        F'0'                   Address of main task save area
    ST1SVEND      DC        F'0'                   End address of subtask 1 save area
    SUB1NAME      DC        C'SUBTASK1'            Subtask 1 name
    ABSVCODE      DC        X'0'
    MTSAVE        DS        9D                     Main task save area used by STXIT
```

A Initializes the subtask save area with the subtask's name which is used for subtask identification in messages written on SYSLOG.

B Attaches the subtask. SUBTASK1 is the entry point of the subtask. ST1SAV is the subtask's save area, ST1ECB is its ECB, and ST1ABSV is its abnormal termination save area. In this example, the subtask uses the main task's abnormal termination routine.

C The statements test for a successful ATTACH. If the ATTACH was successful, the main task stores the ending address of the subtask's save area for later reference, if necessary. The main task can then continue to do other processing.

D This is the subtask's entry point. In this example, the main task and the subtask use different base registers. This may not be necessary, depending on program design. The subtask could have omitted the BALR and USING statements because addressability is warranted through the main task's register 2.

E The statement saves the address of the main task's save area for reference by the subtask -- if it is necessary for the subtask to name the main task in the POST macro.

F The statement stores the ABTERM code when the abnormal termination routine is entered. This routine is shared by the main task and the subtask.

G The statements determine which task was abnormally terminated (the ABTERM save area of the task in error is stored in register 1).

H Defines the user-coded ECB for the subtask.

Figure 10-16. Attaching a subtask.

Subtask Termination

A subtask is normally terminated via the DETACH macro issued by the main task or by the subtask itself.

A subtask can further detach itself by issuing the CANCEL, EOJ or DUMP macro. When a subtask is detached, all pending I/O operations are completed and any tracks held by this subtask are freed.

If a subtask being detached has an ECB, that ECB is posted and any tasks waiting on the ECB are removed from the wait state. The task with the highest priority then gains control. The supervisor ECB for subtask attachment is also posted so that any main task in another partition waiting to attach another subtask is removed from the wait state.

Figure 10-17 shows an example of detaching subtasks. The main task attaches two subtasks. When subtask 1 completes processing, it notifies the main task. The main task then detaches subtask 1 by issuing a DETACH macro and specifying the save area of subtask 1. When subtask 2 completes its processing, it detaches itself.

```
MAINTASK  BALR   2,0
          USING  *,2
          •
          •
ATST1     ATTACH ST1,SAVE=ST1SAV,        X
                 ECB=ST1ECB
          •
          •
ATST2     ATTACH ST2,SAVE=ST2SAV,        X
                 ECB=ST2ECB
          •
          •
          DETACH SAVE=ST1SAV
                 Detach subtask 1
          •
          •
ST1       ST     1,MTSVAR1
          •
          •
          B      ST1+4
ST2       ST     1,MTSVAR2
          •
          •
*Subtask 2 DETACHes itself
          DETACH
```

Figure 10-17. Detaching a subtask.

Resource Protection

When two or more tasks in the same partition manipulate a resource (data in the same area, an I/O device, a set of instructions, etc.), protection should be provided to prevent the resource from being used concurrently by these tasks. If every task within the partition uses the RCB, ENQ, and DEQ macros, such protection is possible. Note, however, that resource protection is not possible for system units such as the SYSLST device.

A task protects a resource by issuing an ENQ (enqueue) macro. The ENQ macro refers to the resource through the name of a resource control block (RCB). The RCB is an 8-byte field generated by issuing the RCB macro. For the format of the RCB, refer to Figure 10-18.

X	Reserved			Flag Byte	ECB address of current resource owner		
0	1	2	3	4	5	6	7

Byte

- Byte 0: Availability byte —
  All ones when resource is in use.
  All zeros when not in use.
- Byte 4: Flag byte: Bit 0 =
  1, another task waiting for the resource.
  0, no other task waiting for the resource.

Figure 10-18. Resource Control Block (RCB).

Any subtask that enqueues a resource must have an ECB specified in its ATTACH macro, and that ECB should not be used for any purpose other than resource protection as long as the resource is enqueued. The address of the ECB is stored in the RCB (see Figure 10-18).

A task requesting the use of a resource is either enqueued and executed or put into the wait state if the resource has already been enqueued by another task. If an ENQ macro is issued for an already enqueued resource, the system indicates this in the RCB and stores the address of the current resource owner's ECB in register 1 of the task that is placed into the wait state.

A task releases a resource by issuing the DEQ macro. If other tasks are enqueued on the same RCB, the DEQ macro frees from their wait condition all other tasks that were waiting for that resource. Once a resource has been enqueued, only the current owner of the resource can dequeue that resource. The task with the highest priority obtains control. If no other tasks are waiting for the RCB, control returns to the dequeueing task.

Figures 10-19 through 10-21 show examples of the use of the ENQ, DEQ, and RCB macros and the resource control block.

Figure 10-19 shows a main task which has two subtasks sharing the same resource and protecting that resource from simultaneous access. The subtasks use the same file in a common subroutine. The subroutine is not reentrant, and the file cannot use track hold. Each subtask must, therefore, enqueue the RCB associated with the resource and dequeue it when the resource can be released.

In Figure 10-20, two subtasks share a common processing routine that is defined in the first subtask. The common routine, called TOTAL, is protected in subtask 1 by the RCB named RCBA. The protection is effective only if every segment of code within the partition that refers to TOTAL issues the ENQ macro before executing TOTAL and subsequently dequeues that resource with the DEQ macro. This is effectively accomplished by branching to the same code in subtask 1.

The common code need not be reentrant. You should, however, ensure that the values for constants associated with the subroutine do not have to be retained from one reference to the next, whenever the resource is used. If the values must be retained, you should save them in the appropriate subtask and restore them when required.

In Figure 10-21, the subtasks again share the use of the same resource, but they use different subroutines for processing that resource. The resource, called RESRCA, may be a data area or a file defined by a DTF macro. In either case, RESRCA is protected from being used by subtask 2 while subtask 1 is operating on it. Thus, if all tasks enqueue and dequeue each reference to RESRCA, RESRCA is protected during the time it takes to process instructions from that task's ENQ to its DEQ macro. This is readily apparent if RESRCA is in storage. However, if it is a file, the record being operated upon is protected while in storage, but it is not necessarily protected on the external storage device. If the file is on DASD, the track hold facility should, if possible, be used.

In your program design, be careful to avoid situations that might lead execution of the program into a deadlock.

Consider the following two segments of code being executed concurrently by two tasks:

```
task1 executes:         task2 executes:

A  ENQ  RCBA            B  ENQ  RCBB
    •                       •
    •                       •
    •                       •
C  ENQ  RCBB            D  ENQ  RCBA
    •                       •
    •                       •
    •                       •
   DEQ  RCBA               DEQ  RCBB
```

If the macros were executed in the sequence **A**, **B**, **C**, processing of both tasks would end at statements **C** and **D** without a chance of ever being resumed.

```
MAINTASK   BALR    2,0
           USING   *,2
           •
           •
SUBTASK1   EQU     *
           •
           •
SBTASK1A   EQU     RCB1        Protect resource
           BAL     4,WRITEDTA  Write a record
           DEQ     RCB1        Release resource
           •
           •
           B       SBTASK1A
           •
           •
SUBTASK2   EQU     *
           •
           •
SBTASK2A   ENQ     RCB1        Protect resource
           BAL     4,WRITEDTA  Write a record
           DEQ     RCB1        Release resource
           •
           •
           B       SBTASK2A
           •
           •
RCB1       RCB                 Resource control block for WRITEDTA
```

Figure 10-19. Sharing a resource in a common subroutine.

```
MTASK       START  0
            •
            •
            ATTACH STASK1,SAVE=SAVE1,ECB=ECB1
            •
            •
            ATTACH STASK2,SAVE=SAVE2,ECB=ECB2
            •
            •
STASK1      ENQ    RCBA        Protect resource TOTAL
            •
            •
* Process TOTAL                Used by STASK1 and STASK2
            •
            •
            DEQ    RCBA        Release resource TOTAL
            •
            •
STASK2      EQU    *
            •
            •
            B      STASK1      Process TOTAL
            •
            •
RCBA        RCB                RCB for resource TOTAL
```

Figure 10-20. Sharing a resource defined in one task.

```
MTASK     START 0
          •
          •
          ATTACH ST1,SAVE=SAVE1,ECB=ECB1
          •
          •
          ATTACH ST2,SAVE=SAVE2,ECB=ECB2
          •
          •
STASK1    EQU     *
          •
          •
          ENQ     RCBA           Protect resource RESRCA
          •
          •
* Update RESRCA
          •
          •
          DEQ     RCBA           Release resource RESRCA
          •
          •
STASK2    EQU     *
          •
          •
          ENQ     RCBA           Protect resource RESRCA
          •
          •
* Update RESRCA                  Process using RESRCA
          •
          •
          DEQ     RCBA           Release resource RESRCA
          •
          •
RCBA      RCB                    RCB for resource RESRCA
RESRCA    DS or DTF              Shared resource
```

Figure 10-21. Sharing a resource in different subroutines.

Resource-Share Control

Another set of macros protect a resource against concurrent use by different tasks (in the same or in different partitions) while permitting controlled sharing of the resource. The macros:

- define a protected resource: DTL, GENDTL, MODDTL
- control resource sharing: LOCK, UNLOCK.

As with the ENQ/DEQ macros, protection is possible only if all users of a particular resource use the protection service in a consistent manner; that is, make use of the available macros and adhere to installation defined naming conventions.

Unlike the ENQ/DEQ macros, the LOCK/UNLOCK macros protect a resource across partitions and allow sharing among several tasks.

Defining a Shareable Resource

A resource to be protected by this share control service must be defined in a lock control block called DTL (Define The Lock). The DTL macro is used to assemble a DTL in your program; the GENDTL macro is used to dynamically build a DTL when your program is executed. The lock control block indicates the installation-defined name of the protected resource and how access to the resource may be shared with other programs.

In the macro that defines the DTL, you can specify the resource as either type E (exclusive) or type S (shareable). In addition, you specify a lock option that determines the extent of shareability; this may be one of the following:

- LOCKOPT=1 means that the resource is either accessed by one user exclusively *or* is shared by several users.
- LOCKOPT=2 means that the resource may be accessed by one exclusive user *and* by several shared users.

The DTL definition macros also allow you to specify the scope of share control; whether a locked resource is automatically released at the end of the particular job step, or whether it remains locked for the next job step (the resource is always released at end of job). You may also specify whether a lock issued by one task can be unlocked only by the same task or also by another task.

Controlling a Shareable Resource

Once the DTL has been built, a task can request control over the protected resource with the LOCK macro and give up control over the resource with the UNLOCK macro.

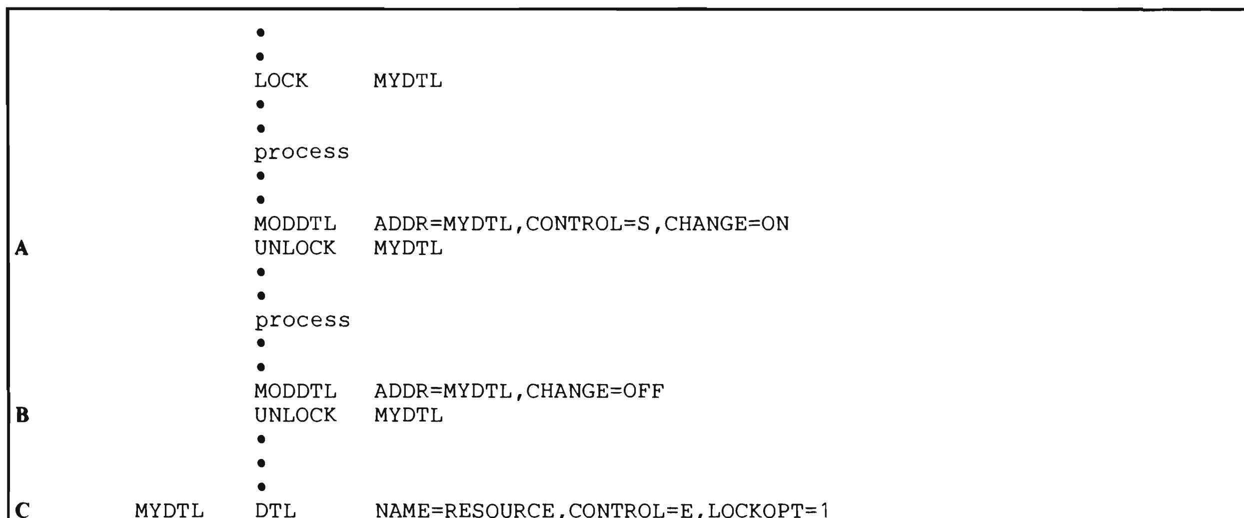
When a LOCK request is issued and the resource is already locked by another task, further system action depends on your specification in the FAIL operand of the LOCK macro. For example, you can request that control is to return to the requesting task regardless of whether the resource can be obtained or not — in which case your program has to test the return code set by the system, or that the requesting task be set into the wait state until a locked resource can be accessed.

A task or partition may lock a resource more than once. The system maintains a lock request count which reflects the number of lock requests issued for the resource. When a resource is locked more than once by a task or partition, the task or partition has to issue as many unlock requests as it issued lock requests to yield control of the resource completely.

The MODDTL macro modifies a lock control block at the time of program execution. This is its normal function. In addition, it is also used to lower the lock control level of a locked resource. When its CHANGE operand is specified as ON, the MODDTL macro causes a subsequent issuance of the UNLOCK macro to keep the resource locked, but with a lower locking level, rather than release the resource. The resource continues to be held; however, another task waiting for this resource can be dispatched again. This method of reducing the lock level can be employed only when the lock level is defined with the most stringent values possible; that is, CONTROL=E (exclusive) and LOCKOPT=1.

Figure 10-22 illustrates the two occurrences of the UNLOCK macro.

In addition to coding the macros, implementation of resource-share control requires that your installation's supervisor includes a sufficiently large lock table space (which you define in the NRES operand of the IOTAB generation macro). A maximum size table accomodates up to 512 concurrent locks.



```
                      •
                      •
                      LOCK      MYDTL
                      •
                      •
                      process
                      •
                      •
                      MODDTL    ADDR=MYDTL,CONTROL=S,CHANGE=ON
A                     UNLOCK    MYDTL
                      •
                      •
                      process
                      •
                      •
                      MODDTL    ADDR=MYDTL,CHANGE=OFF
B                     UNLOCK    MYDTL
                      •
                      •
                      •
C          MYDTL      DTL       NAME=RESOURCE,CONTROL=E,LOCKOPT=1
```

A The resource is not unlocked. It remains locked, but it may now be shared with other tasks.

B Due to CHANGE=OFF in the preceding MODDTL macro, the resource is actually unlocked.

C CONTROL=E with LOCKOPT=1 indicates that no other task is to gain access to resource RESOURCE as long as access is controlled via this DTL.

Figure 10-22. Example of the UNLOCK Macro when CHANGE is set ON in the MODDTL Macro.

Intertask Communication

Tasks communicate with each other through event control blocks (ECBs). For the format, refer to Figure 10-15. One task sets itself into the wait state by issuing the WAIT or WAITM macro, another task issues a POST macro. These macros use ECBs as operands.

A WAIT macro would be used if your task waits for a single event to occur; a WAITM macro is appropriate if your task can continue processing when one or several out of a number of events occurred. The task that issues the WAIT or WAITM remains in the wait state until one of the designated ECBs gets posted by the POST macro, that is: bit 0 of byte 2 set to 1. Blocks that can be used as ECBs are CCBs and TECBs. However, a task never regains control if it is waiting for a CCB to be posted by another task's I/O completion.

A MICR CCB gets posted only when the device stops, not when reading a record is complete. Furthermore, telecommunication ECBs and all RCBs must not be waited for because bit 0 of byte 2 of these blocks would never be posted.

When control returns to a task that was waiting due to a WAITM, register 1 points to the posted ECB. This allows the task to determine which event removed it from the wait state.

A task that issues the WAITM macro should ensure that the waiting task allows an eventual outlet if an event might not occur. (Such a condition could occur if, for example, a task that is to post an event is terminated abnormally.)

In Figure 10-23, the WAITM macro specifies a preferred event (ECBPREF) as the first operand and a secondary event (ECBSEC) as the second operand. The preferred event is the posting of ECBPREF after subtask 1 completes its processing. If, however, the subtask is terminated before it can finish its processing, the supervisor posts the ECB defined via the ATTACH macro (ECBSEC). This would be the secondary event and satisfy the WAITM macro. With either event, the address of the posted ECB is in register 1 after the WAITM macro has been satisfied. This address can select a problem program routine.

In this particular example, a branch instruction points to a table that contains a list of ECBs with corresponding branch instructions to the routine that is to receive control when the pertinent ECB was posted. The table can easily be expanded to include a maximum of 16 ECBs.

Whenever a task posts an ECB, any task waiting on this ECB to be posted is removed from the wait state.

You can code your application to have just one task or all tasks waiting on a particular event removed from the wait state. To have all tasks removed, simply issue a POST macro with the ECB name specified as the only operand. Example:

```
POST     TSK1ECB
```

```
MAINTASK  BALR     2,0
          USING    *,2
          •
          •
          ATTACH ST1,SAVE=SAVE1,ECB=ECBSEC
          •
          •
          WAITM ECBPREF,ECBSEC          Wait for preferred or secondary event
          B        4(1)                 BR to branch in vector table
          •
          •
PREVENT   EQU      *                    Continue after preferred event
          •
          •
SEVENT    EQU      *                    Continue after secondary event
          •
          •
          EOJ                           Main task end of job
          •
          •
ST1       EQU      *
          •
          POST     ECBPREF              POST completion of preferred event
          •
          •
ECBSEC    DC       F'0'                 ECB for secondary event
          B        SEVENT               Vector BR from secondary event
ECBPREF   DC       F'0'                 ECB for preferred event
          B        PREVENT              Vector BR from preferred event
```

Figure 10-23. Waiting for preferred and secondary events.

To have only one task removed, specify the name of the task's save area in the POST macro. Example:

```
POST   TSK1ECB,SAVE=SAVTSK1
```

Time is saved by specifying the SAVE operand. This operand can be used to control which task is to receive control. That is to say, the SAVE operand can be used to prevent the task with the highest priority from gaining control.

Be careful with this technique when the ECB to be posted is the ECB specified in the ATTACH macro and the ENQ/DEQ macros are used, because the DEQ macro also removes from the wait state all tasks waiting for the protected resource. To avoid such problems, you are advised to use two different ECBs, you are responsible for resetting the traffic bit (bit 0 of byte 2) in the second ECB using the instruction MVI ECB+2, X'00' so that tasks testing that ECB can be put into the wait state.

Figure 10-24 illustrates the use of the POST macro. The example shows three subtasks: ST1, ST2, and ST3. ST1 depends on input which can be supplied by ST2 or ST3 and, therefore, issues a WAITM macro on the ECBs for those subtasks.

Initially, ST1 is placed into the wait state by the WAITM macro. Control then passes to ST2 and then to ST3. When either of the two subtasks has the necessary data for ST1, it posts its ECB that removes ST1 from the wait state. When ST1 finishes processing, it posts its ECB, thus causing the main task to be taken from the wait state. The main task can then detach ST1.

DASD Record Protection (Track Hold)

When a record is being modified by one task, the data transmission unit containing that record must be prevented from being accessed by another. For a CKD device, the data transmission unit is one track; for an FBA device, this unit is an intergral number of blocks. For ease of reading, this unit is frequently referred to as "track". VSE includes the DASD record protection support (frequently called "track hold function") to ensure the required data integrity as indicated above. This support is available for use with both CKD and FBA devices.

Within a partition, record protection can be accomplished for a particular DASD by the resource protection macros or the intertask communication macros. With the resource protection macros, an RCB can be enqueued before each reference to the DASD. With the intertask communication macros, a subtask can wait for an ECB to be posted before each reference to the DASD.

The *hold* function obtains DASD record protection for programs that define files by means of the DTFSD, DTFIS, or DTFDA macros. In these cases, DASD record protection can be obtained within the entire system if the TRKHLD operand of the FOPT

```
MAINTASK    BALR     2,0
            USING    *,2
            •
            ATTACH   ST1,SAVE=AREA1,ECB=ECB1
            •
            •
            ATTACH   ST2,SAVE=AREA2,ECB=ECB2
            •
            •
            ATTACH   ST3,SAVE=AREA3,ECB=ECB3
            •
            •
            WAIT     ECB1                 Wait for completion of subtask 1
            DETACH   SAVE=AREA1           Detach subtask 1
            •
            •
            EOJ
ST1         ST       1,MTSVAR             Store address of main task save area
            •
            •
            WAITM    ECB2,ECB3            Wait for subtask 2 or subtask 3
            •
            •
            •
ST1EOJ      L        0,MTSVAR             Get address of main task save area
            POST     ECB1,SAVE=(0)        POST ECB for main task
            WAIT     ECB1A                WAIT to be detached
            •
            •
ST2         EQU      *
            •
            •
ST2A        EQU      *
            •
            •
            POST     ECB2                 POST ECB for subtask 1
            •
            •
            B        ST2A
            •
            •
ST3         EQU      *
            •
            •
ST3A        EQU      *
            •
            •
            POST     ECB3
            •
            •
            B        ST3A
            •
            •
MTSVAR      DC       F'0'                 Save area address for main task
ECB1A       DC       F'0'                 Dummy ECB for subtask 1
ECB1        DC       F'0'                 ECBs for subtasks
ECB2        DC       F'0'
ECB3        DC       F'0'
```

Figure 10-24. Use of the POST Macro.

macro is specified at system generation time, and if every task specifies the HOLD=YES operand of the DTFxx macro.

The hold function can be used in four specific situations:

1. Updating DTFSD data files.
2. Updating DTFSD work files.
3. Processing DTFDA files.
4. Processing DTFIS files.

In the first and second situation, the track or block range being held is freed automatically by the system. More specifically, the next GET issued to a new track for the file frees the previous hold, and your program should not issue the FREE macro. If a FREE macro is issued by your program, it is ignored

because the logic modules implicitly handle the holding and freeing of tracks (blocks).

For situation 3, the program must issue the FREE macro for each hold that is placed on the track. A hold is placed on a track each time the track is accessed with a READ, and each hold is released by issuing FREE, or a CLOSE macro for that file, or a DETACH macro for the associated task.

For DTFDA files using WRITE or WRITE AFTER, DAMOD initially places a hold on the track. However, a WRITE AFTER issued to a track that has the maximum number of holds already in effect cancels the task (or partition).

When a READ ID or READ KEY macro is issued, DAMOD holds the track but does not free it automatically. This must be done in the user program by the FREE macro.

For situation 4, the method of implementation depends on the function being performed:

- Sequential Retrieval - The track index is held at the beginning of retrieval from each cylinder. A search and hold is issued for the data track, the index track is released, and a wait is issued for the data track. When the system is finished with the data track (prime or overflow), it is released, and the next track is held. Your program must release the track hold function by issuing either a PUT (if the file is updated) or a GET (no update) for the next record, or an ESETL.
- Random Retrieval - The track index is held while ISAM reads in the required entries. The data track is held, and the desired record is searched for. When the record is found, the track index is released. Your program must release the data track by issuing a WRITE (if the file is updated) or a FREE (no update).
- Add - The track index and the data track are held. If the record is not going onto the prime data track, the track index is released. All tracks being changed are held during modification. The track index is again held while it is updated to reflect the added records. After alteration, the tracks are released automatically.

  A file for which HOLD=YES is specified must be closed by a CLOSE macro in your program in order to be opened again for processing sometime in the future. If, for any reason, a file cannot be closed properly during execution of a job, run a dummy job that does no more than issue a CLOSE for that file.
- SETL macro - SETL issues a hold on the track index on which processing will begin. This hold is released automatically at the appropriate time.
- ESETL macro - ESETL frees any tracks that are held by sequential retrieval when the ESETL is issued. Since the ESETL macro issues a FREE whether or not any tracks are held, you should not issue ESETL if SETL has not been successful.

The maximum number of DASD track protection holds that can be in effect within a system is specified at the time of system generation. This can be any number up to 255, with a system default option of 10. If a task attempts to exceed the limit, the task is placed in the wait state until a previous hold is lifted.

The same track can be held more than once without an intervening FREE if the hold requests are from the same task. The same number of FREEs must be issued before the track is completely freed. However a task is terminated if more than 16 hold requests from it are recorded without an intervening FREE, or if the task issues FREE for a file that does not have a hold request for that track.

If a task requests a track that is being held by another task, that task is placed into the wait state at the GET (or WAITF) macro associated with the I/O request. The request is fulfilled after the track is freed and when control returns to the requesting task.

If more than one track is being held, it is possible for your program to inadvertently put the entire system in the wait state. This occurs if each task is waiting for a track that is already held by another task. A way to prevent this is to FREE each track held by a task before this task places (or attempts to place) a hold on another track.

Track Hold - An Example

Figure 10-25 shows an example of the use of the track hold facility in a multitasking program.

Although track hold applies across partitions, the example in Figure 10-25 only shows two subtasks sharing the same DA file and the same DA modules. A similar set of routines could be executing in another partition and share the file with this partition, but the second partition would then have to have its own DA module.

Because the subtasks in Figure 10-25 share the same file, HOLD=YES and RDONLY=YES must have been specified in both DTFDAs and in the DAMOD macro. In addition, before any READ, WRITE, or WAITF macro is issued, register 13 must contain the

address of a unique save area to store the registers used by the module. Register 13 is not altered between I/O operations performed by a given task and, therefore, needs to be initialized only once. If other reentrant access methods were used by the subtask, register 13 would have to be initialized for each LIOCS function.

Shared Modules and Files

The DTF and logic module macros for the various file types contain the operand RDONLY=YES indicating that a shareable read-only module is to be generated.

Each time a read-only module is entered, register 13 must contain the address of a 72-byte, doubleword-aligned save area. A separate save area is also required if an exit to a user routine (AB, IT, OC, PC, TT) is established and the exit routine issues I/O request(s) requiring the same logic module as the main routine.

Each task using a read-only module requires its own unique save area in addition to the save areas that might be needed for multitasking and program linkage. The fact that the module save areas are unique for each task makes the module reentrant (that is, capable of being used concurrently by several tasks).

If an ERROPT or WLRERR routine issued I/O macros that use the same read-only module which passed control to either error routine, your program must provide another save area: one save area for the initial I/O and the other for I/O operations in the ERROPT or WLRERR routine. Before control returns to the module that entered the ERROPT routine, reg-

```
MAINTASK    START     0
            •
            •
            ATTACH    ST1,SAVE=AREA1,ECB=ECB1
            •
            •
            ATTACH    ST2,SAVE=AREA2,ECB=ECB2
            •
            •
ST1         OPEN      DAFILE1            OPEN DA master file
            •
            •
            LA        13,DASAVE1         Initialize register 13 with DA save area
            READ      DAFILE1,KEY        Read and hold record
            •
            •
            WAITF     DAFILE1
            •
            •
            WRITE     DAFILE1,KEY        Write updated record
            WAITF     DAFILE1
            FREE      DAFILE1            Release track
            •
            •
DAFILE1     DTFDA     HOLD=YES,RDONLY=YES,...
            •
            •
ST2         OPEN      DAFILE2            OPEN DA master file
            •
            •
            LA        13,DASAVE2         Initialize register 13 with DA save area
            READ      DAFILE2,KEY        Read and hold record
            WAITF     DAFILE2            From DA master file
            •
            •
            WRITE     DAFILE2,KEY        Write updated record
            WAITF     DAFILE2
            •
            •
            FREE      DAFILE2            Release track
            •
            •
DAFILE2     DTFDA     HOLD=YES,RDONLY=YES,...
            •
            •
DASAVE1     DC        8D'0'              Save areas used for
DASAVE2     DC        8D'0'              Shared and reentrant modules
```

Figure 10-25. Using the track hold facility.

ister 13 must contain the address of the save area originally specified for the task.

Programs using devices such as an optical reader can make use of the multitasking function to increase I/O overlap without reentrant modules. However if the program ignores module considerations, two tasks may attempt to use a single nonentrant module. When this occurs, the results are unpredictable because values for the first task using the module are modified by the second task. To prevent this undesirable situation, several methods can be used.

One method is to assemble a module with a different module name for each task that could attempt to use the module simultaneously. This method requires that you specify the appropriate module name in the DTF macro operand MODNAME.

Another method is to link-edit DTF and module separately for each task that could simultaneously attempt to use the same module. Then, before a task attempts to reference a device through that module, the DTF and module can be fetched or loaded into storage.

Either of these methods prevents the linkage editor from resolving linkage to one module. Thus, separate modules can be provided to perform each function. For more information on the linkage between the DTF and logic module, see "Interrelationship of the Macros" in the section "Macro Types and Their Use".

If several tasks are to share processing or to reference data on the same file, not only should reentrant modules be employed but each task must contain its own DTF table for that file (unless you use the ENQ and DEQ macros). Each task can either open its own DTF, or the main task in the partition can open all files for the subtasks.

There are two methods that can be used for a shared file. You can either supply a separate set of label statements (DLBL and EXTENT, TLBL etc.) for each corresponding DTF filename, or you can assemble each DTF and program (subtask) separately with the same filename and one set of label statements. In the latter case, each separately assembled program must open its DTF.

Loading a Forms Control Buffer

An application may require a change of forms one or more times during its execution. VSE provides the LFCB macro for that purpose.

The LFCB macro loads a phase that is cataloged in the core image library into the printer's forms control buffer (FCB). The phase contains the forms spacing layout that you wish to load into the printer's FCB. For information on the contents and format of an FCB phase, see the section "System Control Buffer Load (SYSBUFLD)" in *VSE/Advanced Functions System Control Statements.*

An FCB whose contents have been changed by means of this macro retains new contents until one of the following occurs:

- another LFCB macro is issued for the printer;
- an LFCB command is issued for the printer;
- the SYSBUFLD program is executed to reload the printer's FCB;
- IPL is performed for the system.

The LFCB macro can be particularly useful in an abnormal termination routine that you specify in an STXIT macro. If the routine causes a dump to be produced on a 3211 printer, and if indexing was used before your abnormal termination routine received control, a certain number of characters on every line of the printed dump may be lost. To avoid losing characters, your abnormal termination routine must first issue an LFCB macro that specifies an image without an indexing byte, followed by the macro that requests the dump.

The LFCB macro, when executed, generates messages to request operator action (such as changing forms) if any manual action is required, and to inform the operator that the FCB of the specified printer has been reloaded.

If the load of the FCB fails, the program is not canceled. A return code is posted in register 15. After each use of the LFCB macro, it is advisable to examine the return code and take appropriate action in your program. A list of return codes is shown in *VSE/Advanced Functions Macro Reference.* Following is an example of how testing register 15 will permit you to continue execution when the LFCB macro fails.

Controlling the number of lines per inch (LPI) of printed output differs for PRT1 printers and non-PRT1 printers. For non-PRT1 printers the LPI is controlled by a hardware switch. For PRT1 printers the LPI is controlled by a code in the FCB image. With the LFCB macro you have the option of specifying LPI=6 or 8. When this macro is issued for the non-PRT1 printer, a message is generated so that the operator can adjust the hardware switch to 6 or 8 lines per inch. If the operand is specified for a PRT1 printer, the system compares the value in the buffer image with the value in the LPI operand of the LFCB macro. If they do not match, the load of the FCB

fails and a return code of X'04' is placed in register 15.

There may be times when it is not possible to determine which of several printers at an installation will be used at execution time. The following routine will allow the appropriate FCB phase to be loaded whether or not the printer is a PRT1 or non-PRT1 type:

```
         LFCB    SYSLST,FCB5203,LPI=8
         LTR     15,15
         BZ      CONTINUE
         CH      15,=H'4'
         BE      TRYAGAIN
         PDUMP   INAREA,OUTAREA
         B       CONTINUE (or B CANCL)
TRYAGAIN LFCB    SYSLST, FCBPRT1
         LRT     15,15
         BZ      CONTINUE
         B       CANCL
CONTINUE •
         •
         •
         EOJ
CANCL    CANCEL ALL
```

In the above example the first LFCB macro will attempt to load the phase FCB5203. If the printer is a non-PRT1 printer (such as the 5203) a message will be issued to the operator to set the hardware switch for 8 lines per inch. If the printer is of the PRT1 type, the value 8 (from the LFCB macro) is compared with the value in the buffer image in the phase FCB5203. In phases that are to be loaded in non-PRT1 printers, the buffer image for lines per inch should be a 0 in bit 3. In phases that are meant to be loaded in PRT1 printers, a 0 in bit 3 means 6 lines per inch. If the printer that you are attempting to load is a PRT1, the LFCB would fail and give a return code of X'04'. This would cause a branch to the LFCB at the label TRYAGAIN. The second LFCB loads the phase FCBPRT1, which has been coded appropriately for the PRT1 printer.

Requesting System Information

You can make inquiries about the current supervisor using the SUBSID INQUIRY macro. The macro retrieves a byte string, which can be interpreted using the mapping DSECT generated by the MAPSSID macro. Thus, you can for instance check whether your current supervisor has been generated for ECPS:VSE mode or for 370 mode, or whether it contains DASD sharing support.

Appendix A: Control Character Codes

CTLCHR=ASA

If the ASA option is chosen, a control character must appear in each record. If the control character for the printer is not valid, a message is given and the job is canceled. If the control character for card devices other than the 2560, 5424, and 5425 is not V or W, the card is selected into stacker 1. The codes are:

Code	Interpretation
blank	Space one line before printing*
0	Space two lines before printing
-	Space three lines before printing
+	Suppress space before printing
1	Skip to channel 1 before printing*
2	Skip to channel 2 before printing
3	Skip to channel 3 before printing
4	Skip to channel 4 before printing
5	Skip to channel 5 before printing
6	Skip to channel 6 before printing
7	Skip to channel 7 before printing
8	Skip to channel 8 before printing
9	Skip to channel 9 before printing
A	Skip to channel 10 before printing
B	Skip to channel 11 before printing
C	Skip to channel 12 before printing
V	Select stacker 1
W	Select stacker 2
X	Select stacker 3 (2560 and 5424/5425 DTFCD files only)
Y	Select stacker 4 (2560 and 5424/5425 DTFCD files only)
X	Select stacker 5 (2560 DTFCD files only)
For DTFDI files on 2560 and 5424/5425	
V	Primary hopper: select stacker 1
W	Primary hopper: select stacker 2
V	Secondary hopper: select stacker 5 (on 2560)
V	Secondary hopper: select stacker 4 (on 5424/5425)
W	Secondary hopper: select stacker 3
* For 3525 print (not associated) files, either space one or skip to channel 1 must be used to print on the first line of a card. For 3525 print associated files, only space one must be used to print on the first line of a card.	

CTLCHR=YES

The control character for this option is the command-code portion of the CCW used in printing a line or spacing the forms. If the character is not one of the following characters, unpredictable events will occur:

Stacker Selection Codes

Hexadecimal Code	Punch Combination	Function
Stacker Selection on 1442 and 2596		
81	12,0,1	Select into stacker 1
C1	12,1	Select into stacker 2
Stacker Selection on 2520		
01	12,9,1	Select into stacker 1
41	12,0,9,1	Select into stacker 2
Stacker Selection on 2540		
01	12,9,1	Select into stacker 1
41	12,0,9,1	Select into stacker 2
81	12,0,1	Select into stacker 3
13	11,3,9	Primary hopper: select into stacker 1
23	0,3,9	Primary hopper: select into stacker 2
33	3,9	Primary hopper: select into stacker 3
43	12,0,3,9	Primary hopper: select into stacker 4
53	12,11,3,9	Primary hopper: select into stacker 5 (2560 only)
93	12,11,3	Secondary hopper: select into stacker 1
A3	11,0,3	Secondary hopper: select into stacker 2
B3	12,11,0,3	Secondary hopper: select into stacker 3
C3	12,3	Secondary hopper: select into stacker 4
D3	11,3	Secondary hopper: select into stacker 5 (2560 only)
Stacker Selection on 3504, 3505, and 3525		
01	12,9,1	Select into stacker 1
41	12,0,9,1	Select into stacker 2

Printer Control Codes

Hexadecimal Code	Punch Combination	Function
Printer Control (except for 3525)		
01	12,9,1	Write (no automatic space)
09	12,9,8,1	Write and space 1 line after printing
11	11,9,1	Write and space 2 lines after printing
19	11,9,8,1	Write and space 3 lines after printing
89	12,0,9	Write and skip to channel 1 after printing
91	12,11,1	Write and skip to channel 2 after printing
99	12,11,9	Write and skip to channel 3 after printing
A1	11,0,1	Write and skip to channel 4 after printing
A9	11,0,9	Write and skip to channel 5 after printing
B1	12,11,0,1	Write and skip to channel 6 after printing
B9	12,11,0,9	Write and skip to channel 7 after printing
C1	12,1	Write and skip to channel 8 after printing
C9	12,9	Write and skip to channel 9 after printing
D1	11,1	Write and skip to channel 10 after printing
D9	11,9	Write and skip to channel 11 after printing
E1	11,0,9,1	Write and skip to channel 12 after printing
0B	12,9,8,3	Space 1 line immediately
13	11,9,3	Space 2 lines immediately
1B	11,9,8,3	Space 3 lines immediately
8B	12,0,8,3	Skip to channel 1 immediately
93	12,11,3	Skip to channel 2 immediately
9B	12,11,8,3	Skip to channel 3 immediately
A3	11,0,3	Skip to channel 4 immediately
AB	11,0,8,3	Skip to channel 5 immediately
B3	12,11,0,3	Skip to channel 6 immediately
BB	12,11,0,8,3	Skip to channel 7 immediately
C3	12,3	Skip to channel 8 immediately
CB	12,0,9,8,3	Skip to channel 9 immediately
D3	11,3	Skip to channel 10 immediately
DB	12,11,9,8,3	Skip to channel 11 immediately
E3	0,3	Skip to channel 12 immediately
03	12,9,3	No operation

Hexadecimal Code	Punch Combination	Function
Printer Control for 3525 with Print Feature		
0D	12,5,8,9	Print on line 1
15	11,5,9	Print on line 2
1D	11,5,8,9	Print on line 3
25	0,5,9	Print on line 4
2D	0,5,8,9	Print on line 5
35	5,9	Print on line 6
3D	5,8,9	Print on line 7
45	12,0,5,9	Print on line 8
4D	12,5,8	Print on line 9
55	12,11,5,9	Print on line 10
5D	11,5,8	Print on line 11
65	11,0,5,9	Print on line 12
6D	0,5,8	Print on line 13
75	12,11,0,5,9	Print on line 14
7D	5,8	Print on line 15
85	12,0,5	Print on line 16
8D	12,0,5,8	Print on line 17
95	12,11,5	Print on line 18
9D	12,11,5,8	Print on line 19
A5	11,0,5	Print on line 20
AD	11,0,5,8	Print on line 21
B5	12,11,0,5	Print on line 22
BD	12,11,0,5,8	Print on line 23
C5	12,5	Print on line 24
CD	12,0,5,8,9	Print on line 25

Appendix B: Assembling Your Program, DTF's, and Logic Modules

All the programs described in this (first) section of Appendix B perform the same function, namely, a card-to-disk operation with the following equipment and options:

1. Card reader: 2540 (SYS004)
2. Disk: 3330 with user labels
3. Record size: 80 bytes
4. Block size: 408 bytes including 8-byte count field (blocking factor of 5)
5. One I/O area and work area for the card reader
6. Two I/O areas for the disk.

The following methods may be used to furnish the DTFs and IOCS logic modules to the card-to-disk program.

1. DTFs, IOCS logic modules, and your program assembled together.
2. Logic modules assembled separately.
3. DTFs and logic modules assembled separately, label exit, EOF exit, and I/O areas assembled with DTFs.
4. Same as in 3 except that I/O areas are moved back into main program.
5. Same as in 4 except that label exit and EOF exit are also moved back into main program.

An example of each of these five methods of assembling the main program, modules, DTFs, and related functions follows. In the figures that accompany the examples, each dashed arrow

– – – – – →

represents a symbolic linkage; with an external reference at the base of the arrow, and a label or section definition designating the same symbol at the head of the arrow.

At the points where an arrow is marked with a circle,

– o – – – – →

it is your responsibility to define an ENTRY or EXTRN symbol, as applicable.

Each dotted arrow

• • • • • →

represents a direct linkage. Components are represented by the small rectangles. Assemblies are represented by the larger bordered areas.

Some of the coding examples have numbers in parentheses in the left hand margin opposite specific instructions. These are provided as reference points in the discussion of subsequent examples.

The examples are followed by a comparison of the five methods.

This section of the appendix finally provides an RPS example to show how the DTFs of an existing program are linked to the RPS logic module in the SVA.

Example 1: Assembling Your Programs, DTFs, and Logic Modules Together

Figure B-1 shows the assembly of the DTFs, logic modules, and your program. Note that the disk logic module is pre-assembled and loaded into the Shared Virtual Area at IPL time. The assembly source deck is:

```
                                              column 72
                                                   |
                                                   v
CDTODISK  START   0                                     Initialize base register.
          BALR    12,0                                  Establish addressability.
          USING   *,12                                  Use reg 13 as ptr to save area
          LA      13,SAVEAREA                           Open both files.
          OPEN    CARDS,DISK
NEXT      GET     CARDS,(2)                             Read one card and move it
                                                        to the disk output buffer.
                                                        Return for next card.
          PUT     DISK
          B       NEXT
SAVEAREA  DS      9D                                    Save area is 72-byte, doubleword
                                                        aligned.
EOFCD     CLOSE   CARDS,DISK                            At card-reader EOF, close both
          EOJ                                           files and exit to job control.

MYLABELS  •                                             Your label-processing routine.
          •
          LBRET   2                                     Return to main program.

CARDS     DTFCD   DEVADDR=SYS004,                  X
                  EOFADDR=EOFCD,                   X
                  IOAREA1=A1,                      X
                  WORKA=YES
DISK      DTFSD   BLKSIZE=408,                     X
                  IOAREA1=A2,                      X
                  IOAREA2=A3,                      X
                  IOREG=(2),                       X
                  LABADDR=MYLABELS,                X
                  RECFORM=FIXBLK,                  X
                  RECSIZE=80,                      X
                  TYPEFLE=OUTPUT
A1        DS      80C                                   Card-input buffer
A2        DS      408C                                  First disk buffer
A3        DS      408C                                  Second disk buffer
          CDMOD   DEVICE=2540,                     X    Card logic module
                  TYPEFLE=INPUT,                   X
                  WORKA=YES
          END     CDTODISK                              Program-start address
```

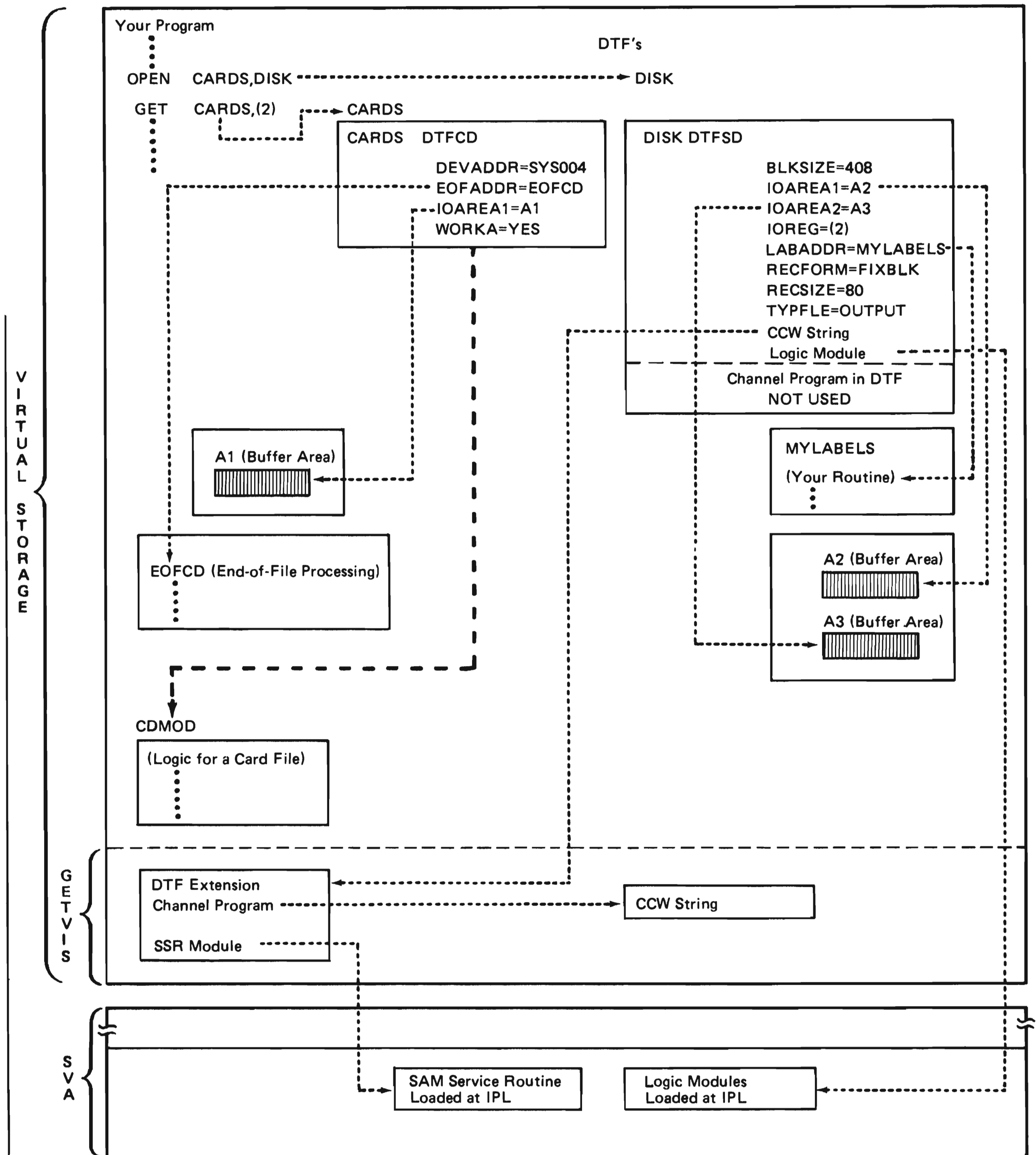


Figure B-1. Assembling your programs, DTFs, and logic modules together (Example 1).

Example 2: Assembling the Logic Modules Separately

The main-program source deck is identical to that in Example 1 up to (1). Figure B-2 shows the separation of the I/O logic modules. The disk logic module is preassembled. The source deck for assembly of the card logic module is as follows:

```
                                    column 72
                                    ______
                                     ↓

CDMOD     DEVICE=2540,               X    Card
          TYPEFLE=INPUT,             X     logic
          WORKA=YES,                 X       module
          SEPASMB=YES
END
```

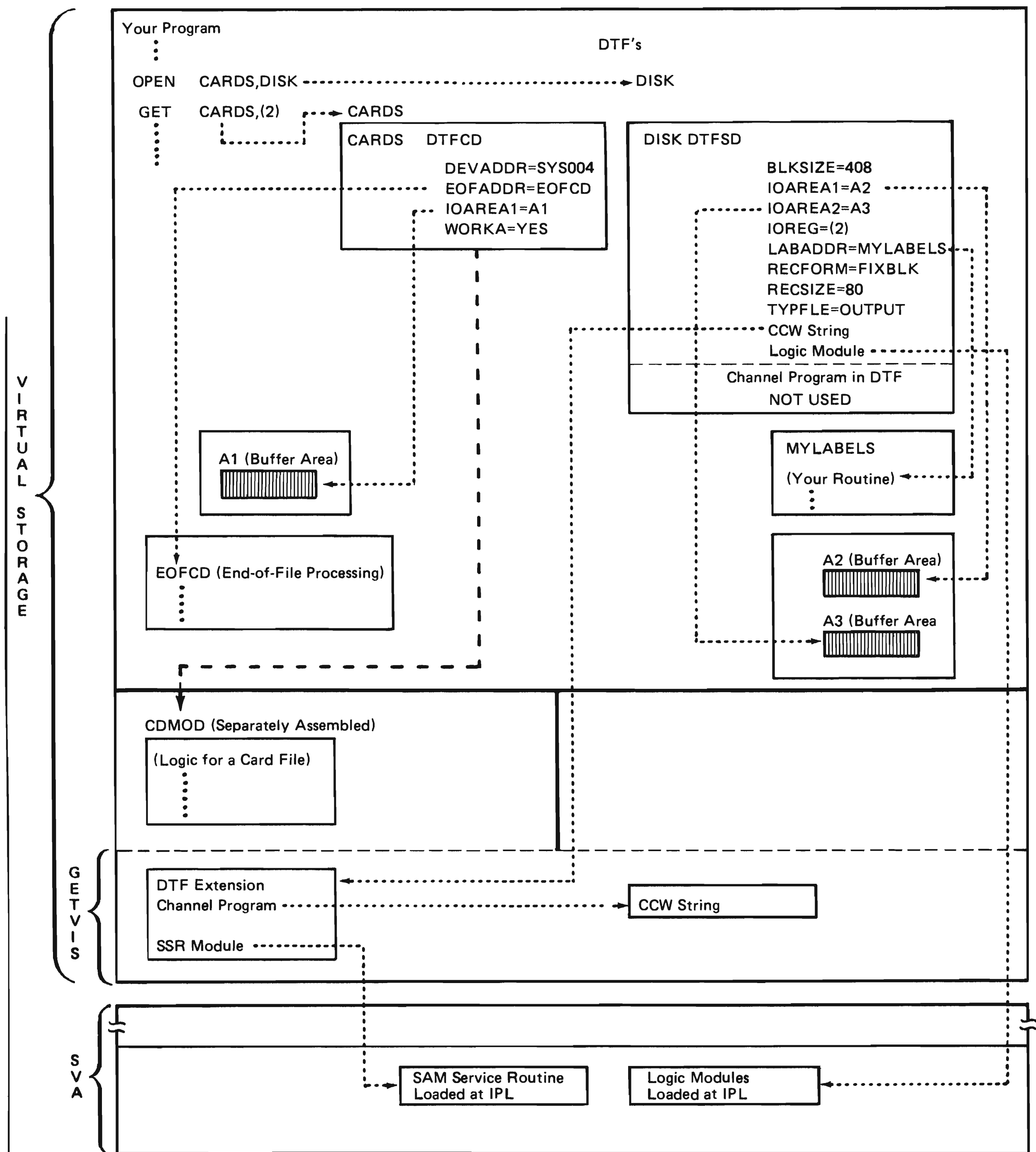


Figure B-2. Logic modules assembled separately (Example 2).

After assembly, each logic module is preceded by the appropriate CATALR card. The modules may be added to the system relocatable library during a maintenance run. Thereafter, logic modules are automatically included in your program by the linkage editor while it prepares the preceding main program for execution.

Example 3: Assembling the DTFs and Logic Modules Separately

The source deck for the main program is as follows:

```
    CDTODISK     START    0
                 BALR     12,0
                 USING    *,12
                 LA       13,SAVEAREA
                 OPEN     CARDS,DISK
    NEXT         GET      CARDS,(2)
                 PUT      DISK
                 B        NEXT
    SAVEAREA     DS       9D
2                EXTRN    CARDS,DISK
                 END      CDTODISK
```

The logic modules are assembled as in Example 2. Figure B-3 shows the separation of the DTFs and logic modules. The DTFCD and related functions are assembled; the source cards are:

```
                                            column 72
                                                ↓
    CARDS    DTFCD                              X
                       DEVADDR=SYS004,          X
                       SEPASMB=YES,             X
                       EOFADDR=EOFCD,           X
                       IOAREA1=A1,              X
                       WORKA=YES
             USING     *,14

    EOFCD    CLOSE     CARDS,DISK
             EOJ
             EXTRN     DISK
3   A1       DS        80C
             END
```

The DTFCD and related functions are assembled; the source deck contains cards as follows:

```
                                            column 72
                                                ↓
   DISK         DTFSD                           X
                         BLKSIZE=408,           X
                         SEPASMB=YES,           X
                         •
                         •
                         •
                         TYPEFLE=OUTPUT
   MYLABELS     BALR     10,0
                USING    *,10
                •
                •
                LBRET    2
4  A2           DS       408C
5  A3           DS       408C
                END
```

In the card-file and the disk-file assemblies, a USING statement was added because certain routines are segregated from the main program and moved into the DTF assembly.

When your routines, such as error, label processing, or EOF routines, are segregated from the main program, it is necessary to establish addressability for these routines. You can provide this addressability by assigning and intializing a base register. In the special case of the EOF routine, the addressability is established by logical IOCS in register 14. For error exits and label-processing routines, however, this addressability is not supplied by logical IOCS. Therefore, if you segregate your error routines, it is your responsibility to establish addressability for them.

Figure B-4 contains the printer output to show how the coding of Example 3 would look when assembled.

In Figure B-4, the standard name was generated for the logic module: V(IJCFZIWO) for DTFCD (see statement 13). The module name appears in the External Symbol Dictionary of the logic module assembly.

A DTF assembly generates a table that contains no executable code. Each of the two DTF tables is preceded by the appropriate CATALR card. The two object decks can be cataloged as follows into the relocatable library together with the logic modules:

```
// JOB CATRELOC
// EXEC MAINT
   (DTFCD Assembly)
   (DTFSD Assembly)
   (CDMOD Assembly)
/*
```

Alternatively, the object decks from these assemblies (DTF tables and logic modules) can be furnished to the linkage editor along with the main program object deck. The sequence follows:

```
// JOB CATALOG
// OPTION CATAL
   INCLUDE
   PHASE name,*
   (object deck, main program)
   (object deck, DTFCD assembly)
   (object deck, DTFSD assembly)
   (object deck, CDMOD assembly)
/*
// EXEC LNKEDT
/&
```

Note: It is not necessary to remove the CATALR card because the linkage editor bypasses it.

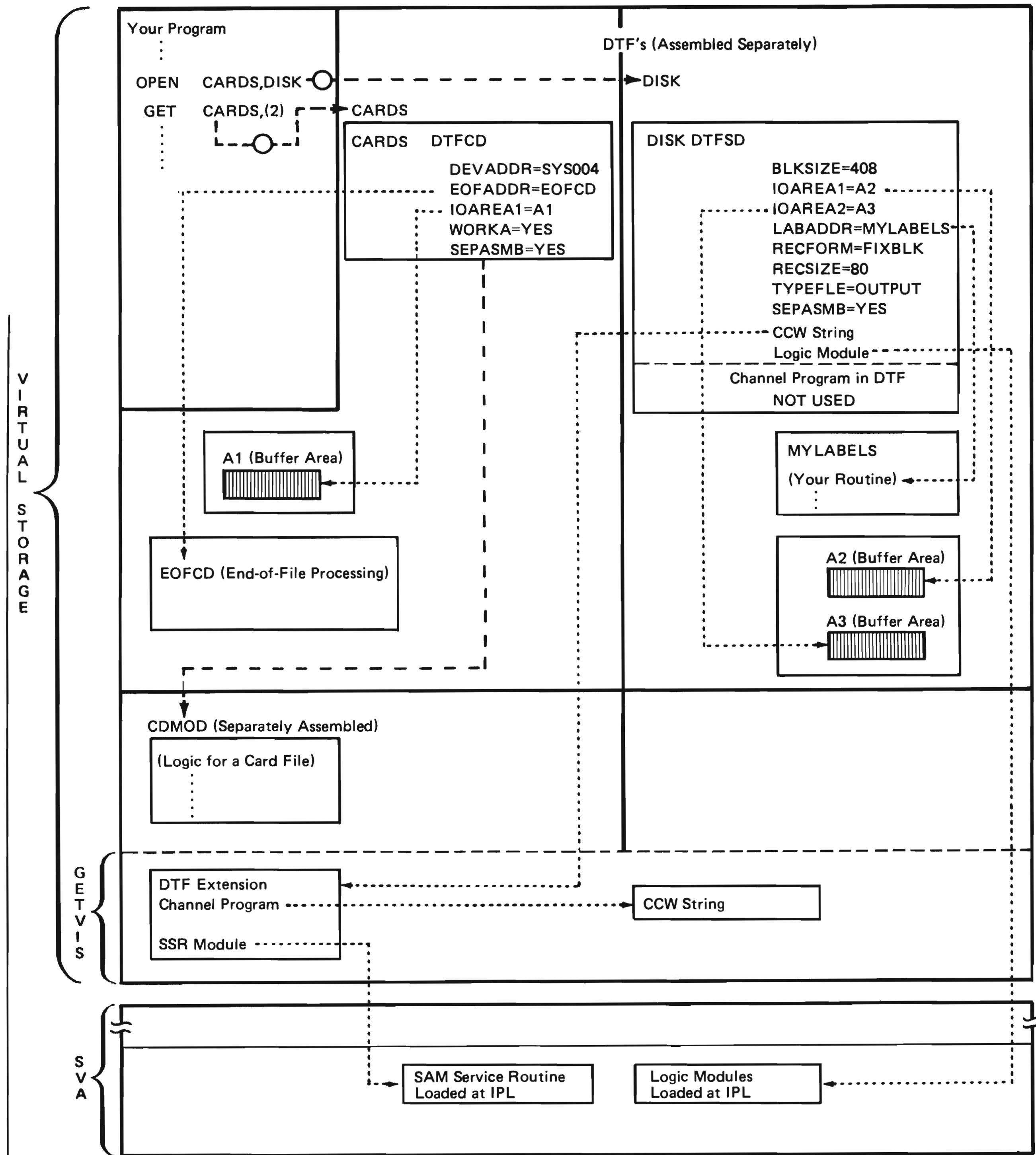


Figure B-3. Logic modules and DTFs assembled separately (Example 3).

```
MAIN PROGRAM
                                        EXTERNAL SYMBOL DICTIONARY                          PAGE    1

SYMBOL    TYPE ID  ADDR   LENGTH LD ID

CDTODISK  SD   01  000000 000090    Section definition.    Control section defined by START statement.
CARDS     ER   02                   External reference.  } Defined by EXTRN statement.
DISK      ER   03                   External reference.
```

```
                                                    EXAMPLE 3                                PAGE    1

  LOC  OBJECT CODE    ADDR1 ADDR2   STMT   SOURCE STATEMENT

000000                                 1 CDTODISK    START   0
000000 05C0                            2             BALR    12,0            INITIALIZE BASE REGISTER
000002                                 3             USING   *,12            ESTABLISH ADDRESSABILITY
000002 41D0 C03E            00040      4             LA      13,SAVEAREA     USE REGISTER 13 AS POINTER TO SAVE
                                       5 * OPEN THE FILE
                                       6             OPEN    CARDS,DISK      OPEN BOTH FILES
                                       7+* IOCS - OPEN -
000006 0700                            8+          CNOP  0,4
000008                                 9+          DC    OF'0'
000008 4110 C086            00088     10+          LA    1,=C'$$BOPEN '
00000C 1BFF                           11+          SR    15,15 ZERO R15 FOR ERROR RETURN     5-0
00000E 0700                           12+          NOPR  0 WORD ALIGNMENT                  5-0
000010 4500 C01A            0001C     13+IJJ00001 BAL   0,*+4+4*(3-1)
000014 00000000                       14+          DC    A(CARDS)
000018 00000000                       15+          DC    A(DISK)
00001C 0A02                           16+          SVC   2
                                      17 NEXT        GET     CARDS,(2)       READ ONE CARD, MOVE TO WORK AREA
                                      18+* IOCS AND DEVICE INDEPENDENT I/O - GET -
00001E 5810 C08E            00090     19+NEXT      L     1,=A(CARDS) GET DTF TABLE ADDRESS
000022 1802                           20+          LR    0,2 GET WORK AREA ADDRESS
000024 58F1 0010            00010     21+          L     15,16(1) GET LOGIC MODULE ADDRESS
000028 45EF 0008            00008     22+          BAL   14,8(15) BRANCH TO GET ROUTINE
                                      23             PUT     DISK            WRITE ON DISK
                                      24+* IOCS AND DEVICE INDEPENDENT I/O - PUT -
00002C 5810 C092            00094     25+          L     1,=A(DISK) GET DTF TABLE ADDRESS
000030 58F1 0010            00010     26+          L     15,16(1) GET LOGIC MODULE ADDRESS      3-5
000034 45EF 000C            0000C     27+          BAL   14,12(15) BRANCH TO PUT ROUTINE          3-5
000038 47F0 C01C            0001E     28             B       NEXT            GO FOR NEXT CARD
000040                                29 SAVEAREA    DS      9D              72-BYTE SAVE AREA
                                      30             EXTRN   CARDS,DISK
000000                                31             END     CDTODISK
000088 5B5BC2D6D7C5D540               32                =C'$$BOPEN '
000090 00000000                       33                =A(CARDS)
000094 00000000                       34                =A(DISK)
```

Figure B-4. Separate assemblies, Example 3 (Part 1 of 4).

```
DTFCD ASSEMBLY
                                         EXTERNAL SYMBOL DICTIONARY                                   PAGE    1

SYMBOL    TYPE ID  ADDR  LENGTH LD ID

CARDSC     SD  01 000000 0000A0        Section definition.           { Generated by specifying SEPASMB=YES in DTFCD macro.
CARDS      LD     000000         01    Label definition (entry point). {
IJCFZIWO   ER  02                      External reference.   Corresponds to V-type address constant generated in DTFCD.
DISK       ER  03                      External reference.   Defined by EXTRN statement.
```

```
                                                                                                      PAGE    1

                                                  EXAMPLE 3

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                     1 CARDS       DTFCD   DEVADDR=SYS004,                              X
                                                           SEPASMB=YES,                                 X
                                                           EOFADDR=EOFCD,                               X
                                                           IOAREA1=A1,                                  X
                                                           WORKA=YES
                                     2+* IOCS AND DEVICE INDEPENDENT I/O - DTFCD -
                                     3+          PUNCH '   CATALR CARDS,4.0' 4-0
000000                               4+CARDSC    CSECT
                                     5+          ENTRY CARDS
000000                               6+          DC    0D'0'
000000 000080000000                  7+CARDS     DC    X'000080000000' RES. COUNT,COM. BYTES,STATUS BTS
000006 01                            8+          DC    AL1(1) LOGICAL UNIT CLASS
000007 04                            9+          DC    AL1(4) LOGICAL UNIT
000008 00000020                     10+          DC    A(IJCX0001) CCW ADDRESS
00000C 00000000                     11+          DC    4X'00' CCB-ST BYTE,CSW CCW ADDR.
000010 00                           12+          DC    AL1(0) SWITCH 3                         4-0
000011 000000                       13+          DC    VL3(IJCFZIWO) ADDRESS OF LOGIC MODULE    3-3
000014 02                           14+          DC    X'02' DTF TYPE (READER)
000015 01                           15+          DC    AL1(1) SWITCHES
000016 02                           16+          DC    AL1(2) NORMAL COMM.CODE
000017 02                           17+          DC    AL1(2) CNTROL COMM.CODE
000018 0000004C                     18+          DC    A(A1) ADDR. OF IOAREA1
00001C 00                           19+          DC    AL1(0) JJ
00001D 00000034                     20+          DC    AL3(EOFCD) EOF ADDRESS                          JJ
000020 0200004C20000050             21+IJCX0001 CCW    2,A1,X'20',80
000028 4700 0000          00000     22+          NOP   0 LOAD USER POINTER REG.
00002C D24F D000 E000 00000 00000   23+          MVC   0(80,13),0(14) MOVE IOAREA TO WORKA
000032                              24+IJJZ0001 EQU    *
000032                              25             USING   *,14            ESTABLISH ADDRESSABILITY
                                    26 * CLOSE THE FILE
                                    27 EOFCD       CLOSE   CARDS,DISK      END OF FILE ADDRESS FOR CARD READER
                                    28+* IOCS - CLOSE -
000032 0700                         29+          CNOP  0,4
000034                              30+EOFCD     DC    OF'0'
000034 4110 E06E          000A0     31+          LA    1,=C'$$BCLOSE'
000038 1BFF                         32+          SR    15,15 ZERO R 15 FOR ERROR RETURN   5-0
00003A 0700                         33+          NOPR  0 WORD ALIGNMENT               5-0
00003C 4500 E016          00048     34+IJJC0002 BAL    0,*+4+4*(3-1)
000040 00000000                     35+          DC    A(CARDS)
000044 00000000                     36+          DC    A(DISK)
000048 0A02                         37+          SVC   2
                                    38             EOJ
                                    39+* SUPVR COMMN MACROS - EOJ -
00004A 0A0E                         40+          SVC   14
                                    41             EXTRN   DISK
00004C                              42 A1          DS      80C            CARD I/O AREA
                                    43             END
0000A0 5B5BC2C3D3D3D6E2C5           44                 =C'$$BCLOSE'
```

Figure B-4. Separate assemblies, Example 3 (Part 2 of 4).

```
DTFSD ASSEMBLY
                                EXTERNAL SYMBOL DICTIONARY                                    PAGE    1
SYMBOL    TYPE ID  ADDR   LENGTH LD ID

DISKC      SD  01 000000 0003D4           Section definition.          } Generated by specifying SEPASMB=YES in DTFSD macro.
DISK       LD     000000          01      Label definition (entry point).
IJGFOZZZ   ER  02                         External reference.  Corresponds to V-type address constant generated in DTFSD.
```

```
                                   EXAMPLE 3                                                  PAGE    1

  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                      1 DISK       DTFSD   BLKSIZE=408,                                   X
                                                          SEPASMB=YES,                                    X
                                                          IOAREA1=A2,                                     X
                                                          IOAREA2=A3,                                     X
                                                          IOREG=(2),                                      X
                                                          LABADDR=MYLABELS,                               X
                                                          RECFORM=FIXBLK,                                 X
                                                          RECSIZE=80,                                     X
                                                          TYPEFLE=OUTPUT,                                 X
                                                          DEVICE=3330
                                      2+* SEQUENTIAL DISK IOCS - DTFSD -
                                      3+           PUNCH '    CATALR DISK,4.0' 4-0
000000                                4+DISKC      CSECT
                                      5+           ENTRY DISK
000000                                6+           DC     OD'0'
000000 000080040000                   7+DISK       DC     X'000080040000' CCB
000006 FF                             8+           DC     AL1(255) LOGICAL UNIT CLASS
000007 FF                             9+           DC     AL1(255) LOGICAL UNIT NUMBER
000008 00000068                      10+           DC     A(IJGC0001) CCB-CCW ADDRESS
00000C 00000000                      11+           DC     4X'00' CCB-ST BYTE,CSW CCW ADDRESS
000010 00                            12+           DC     AL1(0) 3-3
000011 000000                        13+           DC     XL3'0'
000014 20                            14+           DC     X'20' DTF TYPE
000015 49                            15+           DC     AL1(73) OPEN/CLOSE INDICATORS
000016 C4C9E2D2404040                16+           DC     CL7'DISK' FILENAME
00001D 04                            17+           DC     X'04' INDICATE 3330             4-0
00001E 000000000000                  18+           DC     6X'00' BCCHHR ADDR OF F1 LABEL IN VTOC
000024 0000                          19+           DC     2X'00' VOL SEQ NUMBER
000026 08                            20+           DC     X'80' OPEN COMMUNICATIONS BYTE
000027 00                            21+           DC     X'00' XTENT SEQ NO OF CURRENT EXTENT
000028 00                            22+           DC     X'00' XTENT SEQ NO LAST XTENT OPENED
000029 0000A0                        23+           DC     AL3(MYLABELS) USER'S LABEL ADDRESS
00002C 000000A4                      24+           DC     A(A2) ADDRESS OF IOAREA                 4-0
000030 80000000                      25+           DC     X'80000000' CCHH ADDR OF USER LABEL TRACK
000034 0000                          26+           DC     2X'00' LOWER HEAD LIMIT
000036 00000000                      27+           DC     4X'00' XTENT UPPER LIMIT
00003A 0000                          28+DISKS      DC     2X'00' SEEK ADDRESS-BB
00003C 0000FF00                      29+           DC     X'0000FF00' SEARCH ADDRESS-CCHH
000040 00                            30+           DC     X'00' RECORD NUMBER
000041 00                            31+           DC     X'00' KEY LENGTH
000042 0190                          32+           DC     H'400' DATA LENGTH
000044 00000000                      33+           DC     4X'00' CCHH CONTROL FIELD
000048 17                            34+           DC     AL1(23) R    CONTROL FIELD
000049 04                            35+           DC     B'00000100' 3-2
00004A 018F                          36+           DC     H'399' SIZE OF BLOCK-1
00004C FFFFFFFFFF                    37+           DC     5X'FF' CCHHR BUCKET                   3-7
000051 00                            38+           DC     X'00'
000052 32E6                          39+           DC     H'13030' TRACK CAPACITY CONSTANT
000054 5821 0058           00058     40+           L      2,88(1) LOAD USER'S IOREG
000058 000000AC                      41+           DC     A(A2+8) DEBLOCKER-INITIAL POINTER     4-0
00005C 00000050                      42+           DC     F'80' DEBLOCKER-RECORD SIZE
000060 0000023B                      43+           DC     A(A2+8+400-1) DEBLOCKER LIMIT               3-9
000064 0A                            44+           DC     AL1(10) LOGICAL INDICATORS
000065 000000                        45+  .        DC     AL3(0) USER'S ERROR ROUTINE
000068 0700003A40000006              46+IJGC0001 CCW      7,*-46,64,6 SEEK
000070 3100003C40000005              47+           CCW    X'31',*-52,64,5 SEARCH ID EQUAL
000078 0800007000000000              48+           CCW    8,*-8,0,0 TIC
```

Figure B-4. Separate assemblies, Example 3 (Part 3 of 4).

DTFSD (Continued)

```
                                               EXAMPLE 3                                      PAGE    2

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
00
000080 1D00023C00000198              49+          CCW   X'1D',A3,0,400+8 WRITE COUNT KEY AND DATA
000088 3100003C40000005              50+          CCW   X'31',DISKS+2,64,5 SEARCH ID EQUAL
000090 0800008800000000              51+          CCW   8,*-8,0,0 TIC
000098 1E00009830000001              52+          CCW   30,*,48,1 VERIFY
0000A0                               53+IJJZ0001 EQU   *
0000A0 05A0                          54 MYLABELS  BALR    10,0                  INITIALIZE BASE REGISTER
0000A2                               55           USING   *,10                  ESTABLISH ADDRESSABILITY
                                     56 *                                       USER'S LABEL                *
                                     57 *                                       PROCESSING ROUTINE          *
                                     58                LBRET 2                  RETURN TO LIOCS
                                     59+* IOCS - LBRET -
0000A2 0A09                          60+          SVC   9 BRANCH BACK TO IOCS
0000A4                               61 A2         DS      408C                 FIRST DISK I/O AREA
00023C                               62 A3         DS      408C                 SECOND DISK I/O AREA
                                     63            END
```

CDMOD ASSEMBLY

```
                                         EXTERNAL SYMBOL DICTIONARY
  SYMBOL   TYPE ID  ADDR   LENGTH LD ID

IJCFZIWO   SD  01 000000 000060   Section definition. CSECT name generated by CDMOD macro
```

```
                                               EXAMPLE 3

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                      2           PRINT NOGEN
                                      3           CDMOD                                               X
                                                        DEVICE=2540,                                  X
                                                        SEPASMB=YES,                                  X
                                                        TYPEFLE=INPUT,                                X
                                                        WORKA=YES
                                     73           END
```

Figure B-4. Separate assemblies, Example 3 (Part 4 of 4).

Example 4: DTFs and Logic Modules Assembled Separately, I/O Areas with Main Program
The main program is identical to Example 3 except the following four cards, which are inserted after the card marked (2):

```
A1      DS      80C
A2      DS      408C
A3      DS      408C
        ENTRY   A1,A2,A3
```

The separate assembly of logic modules is identical to Example 3 except as indicated below:

In the card-file assembly of Example 3, replace the card marked (3) with the following card:

```
EXTRN A1
```

Similarly, in the disk-file assembly of the previous example, replace the cards marked (4) and (5) with the following card:

```
EXTRN A2,A3
```

Figure B-5 shows the separation of the logic modules, DTFs, and I/O areas.

Example 5: Assembling DTFs and Logic Modules Separately; I/O Areas, Label Exit, and End-of-File Exit with Main Program
In addition to the changes in Example 4, the label exit and the end-of-file exit may be assembled separately. Figure B-6 shows these separate assemblies. The main program is assembled:

```
                                            column 72
                                            ---------
                                                    ↓
CDTODISK  START     0
          BALR      12,0
          USING     *,12
          LA        13,SAVEAREA
          OPEN      CARDS,DISK
NEXT      GET       CARDS,(2)
          PUT       DISK
          B         NEXT
SAVEAREA  DS        9D
EOFCD     CLOSE     CARDS,DISK
          EOJ
MYLABELS
          •
          •
          •
          LBRET     2
          EXTRN     CARDS,DISK
A1        DS        80C
A2        DS        408C
A3        DS        408C
          ENTRY     A1,A2,A3,                               X
                    EOFCD,MYLABELS
          END       CDTODISK
```

The file definitions are separately assembled:

```
                                             column 72
                                             ---------
                                                     ↓
CARDS     DTFCD     DEVADDR=SYS004,                  X
                    WORKA=YES,                       X
                    EOFADDR=EOFCD,                   X
                    SEPASMB=YES,                     X
                    IOAREA1=A1
          EXTRN     EOFCD,A1
          END
DISK      DTFSD     BLKSIZE=408,                     X
                    TYPEFLE=OUTPUT,                  X
                    SEPASMB=YES,                     X
                    •
                    •
                    •
                    IOAREA1=A2,                      X
                    IOAREA2=A3
          EXTRN     A2,A3,MYLABELS
          END
```

The separate assembly of logic modules is identical to Example 3 and Example 4.

Comparison of the Five Methods

Example 1: Requires the most assembly time and the least link-edit time. Because the linkage editor is substantially faster than the assembler, frequent reassembly of the program requires more total time for program preparation than examples 2 through 5.

Example 2: Segregates the IOCS logic modules from the remainder of the program. Because these modules are generalized, they can serve several different applications. Thus, they are normally retained in the system relocatable library for ease of access and maintenance.

When a system pack is generated or when it requires maintenance, the IOCS logic modules that are required for all applications should be identified and generated onto it. Each such module requires a separate assembly and a separate catalog operation, as shown in examples 2 through 5. Many assemblies, however, can be batched together as can many catalog operations.

Object programs produced by COBOL, PL/I, and RPG require one or more IOCS logic modules in each executable program. These modules are usually assembled (as in Example 2) during generation of a system pack and are permanently cataloged into the system relocatable library.

Example 3: Shows how a standardized IOCS package can be separated almost totally from a main program. Only the imperative IOCS macros, OPEN, CLOSE, GET, and PUT remain. All file parameters, label processing, other IOCS exits, and buffer areas are preassembled. If there are few IOCS changes in an application, compared to other changes, this me-

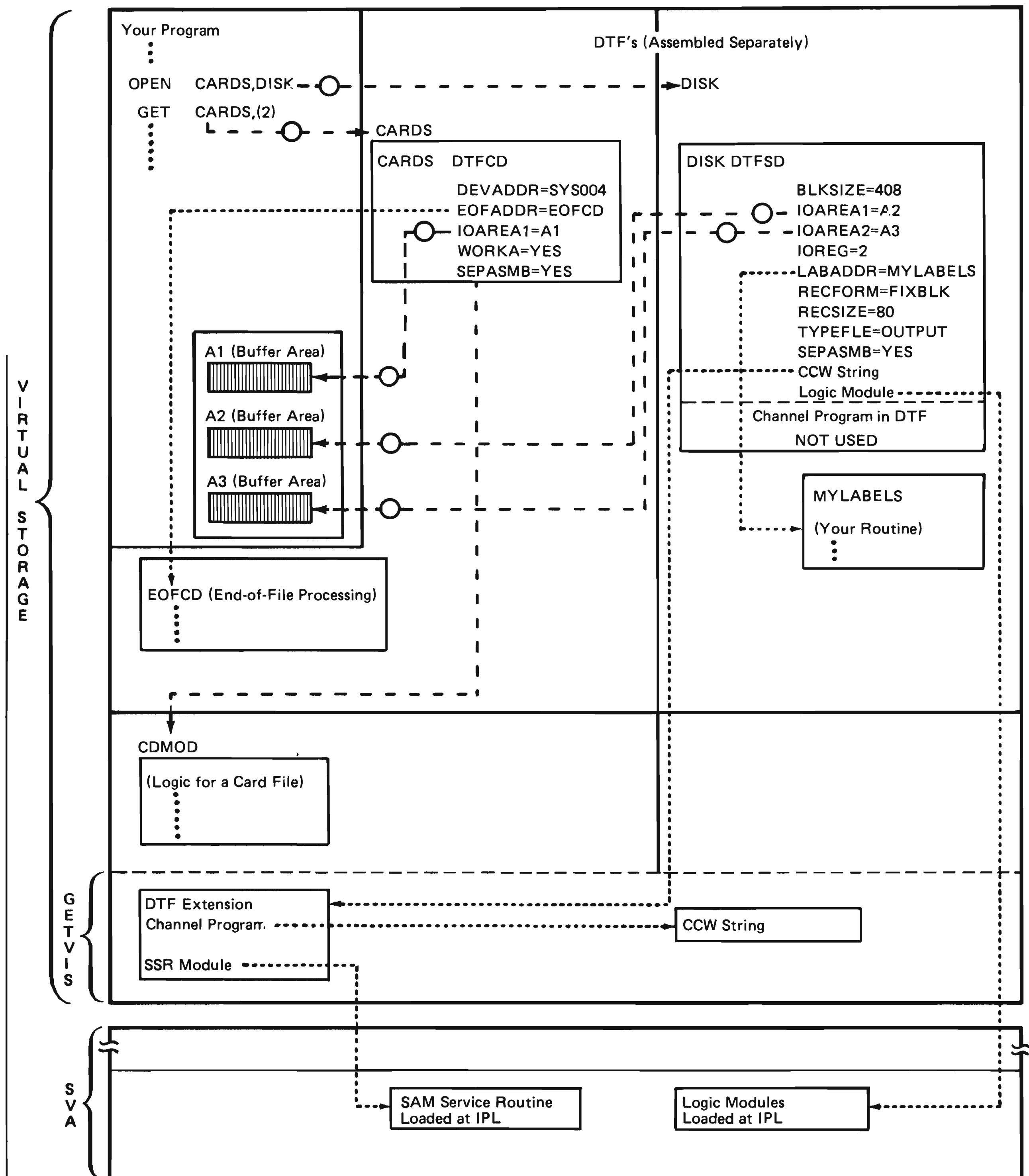


Figure B-5. Logic modules and DTFs assembled separately, I/O areas with main program (Example 4).

thod reduces to a minimum the total development and maintenance time. This approach also serves to standardize file descriptions so that they can be shared among several different applications. This reduces the chance of one program creating a file that is improperly accessed by subsequent programs. In example 3, you need only be concerned with the record format and the general register pointing to the record. You can virtually ignore the operands BLKSIZE, LABADDR, etc. in your program, although you must ultimately consider their effect on virtual storage, job control cards, etc.

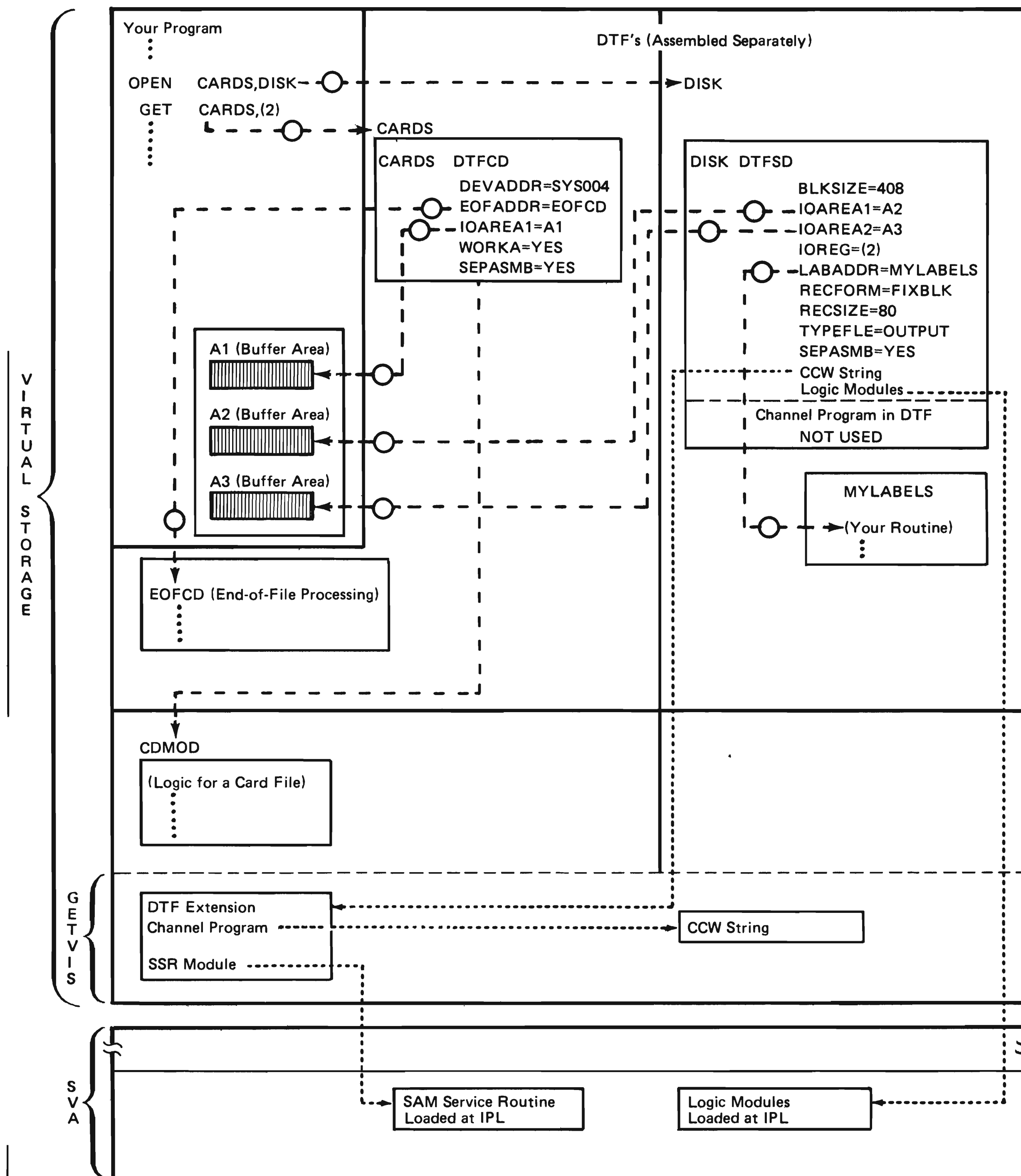


Figure B-6. DTFs and logic modules assembled separately; I/O areas, label exit, EOF exit with main program (Example 5).

In Example 4, a slight variant of example 3, the I/O buffer areas are moved into the main program rather than being assembled with the DTFs.

In Example 5, the label processing and exit functions are also moved into the main program.

Examples 4 and 5: Show how buffers and IOCS facilities can be moved between main program and separately assembled modules. If user label processing is standard throughout an installation, label exits should be assembled together with the DTFs. If each application requires special label processing, label exits should be assembled into the main program.

RPS Example

Example 6: Shows DTFs and logic modules assembled jointly with the program and with Rotational Position Sensing support included.

This example, which applies to DTFIS disk storage access method, shows how the DTF is linked to the RPS DTF extension and the logic module before, during, and after processing with imperative macros. Figure B-7 shows the contents and linkages between these elements before opening the DTF, after a standard non-RPS open, and after closing the file. Figure B-8 shows the linkage between the assembled DTF, the RPS DTF extension, and the RPS logic module in the shared virtual area; the program flow and the related DTFCD block have been omitted for clarity's sake. Note that the original, non-RPS logic module is not used to process the file after a successful RPS open.

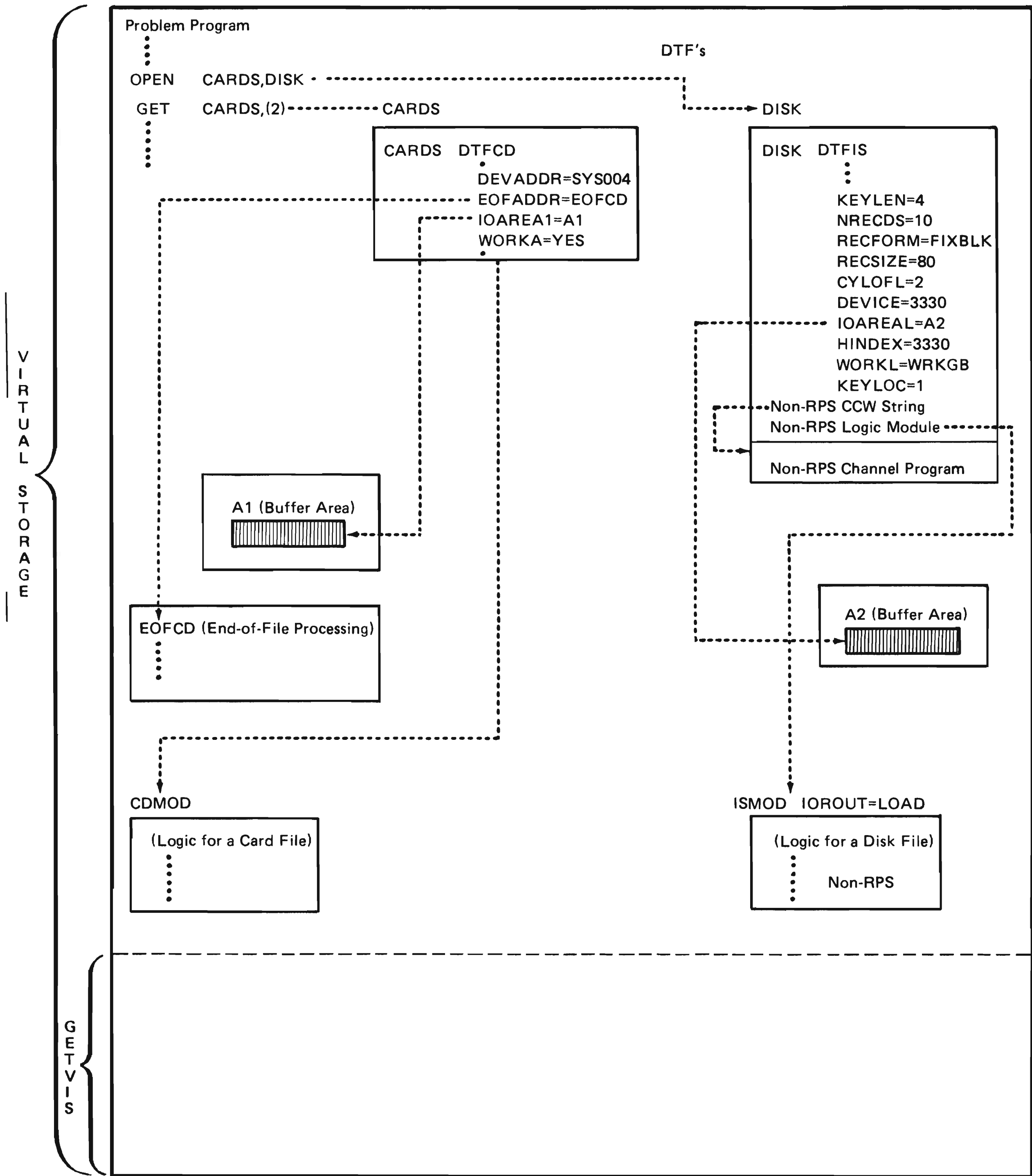


Figure B-7. DTFs and logic modules assembled jointly (Example 6).

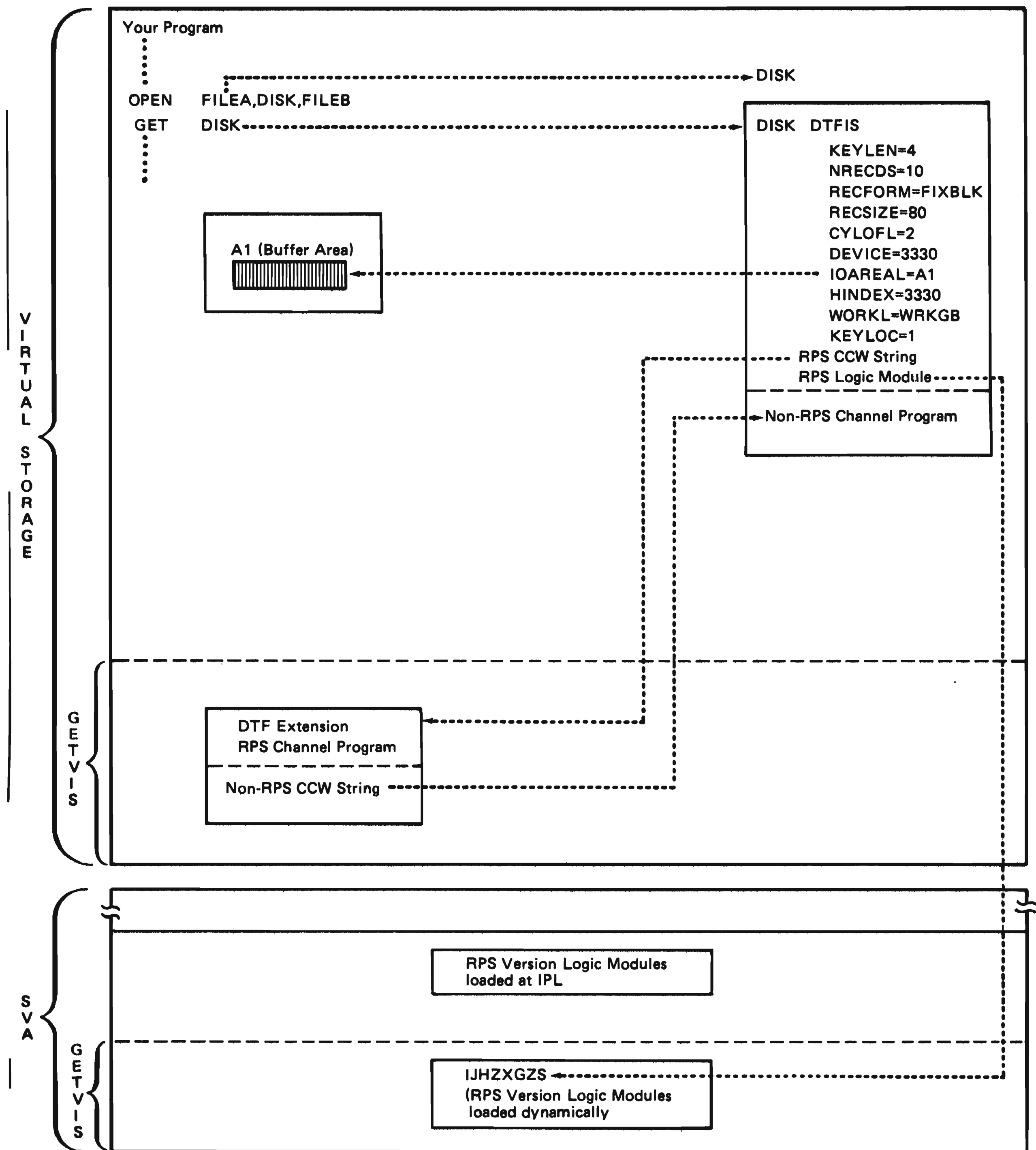


Figure B-8. DTFs and logic modules assembled jointly, with Rotational Position Support included (Example 6).

FBA DASD Example

Example 7: Shows a DTF assembled with a program and a pre-assembled FBA logic module that was loaded into the Shared Virtual Area at IPL time.

This example, which applies to DTFDI and DTFSD access methods, shows how the DTF is linked to the FBA DTF extension and the logic module after a file ASSGNed to an FBA device is opened. OPEN processing establishes the linkages, which are illustrated in Figure B-9.

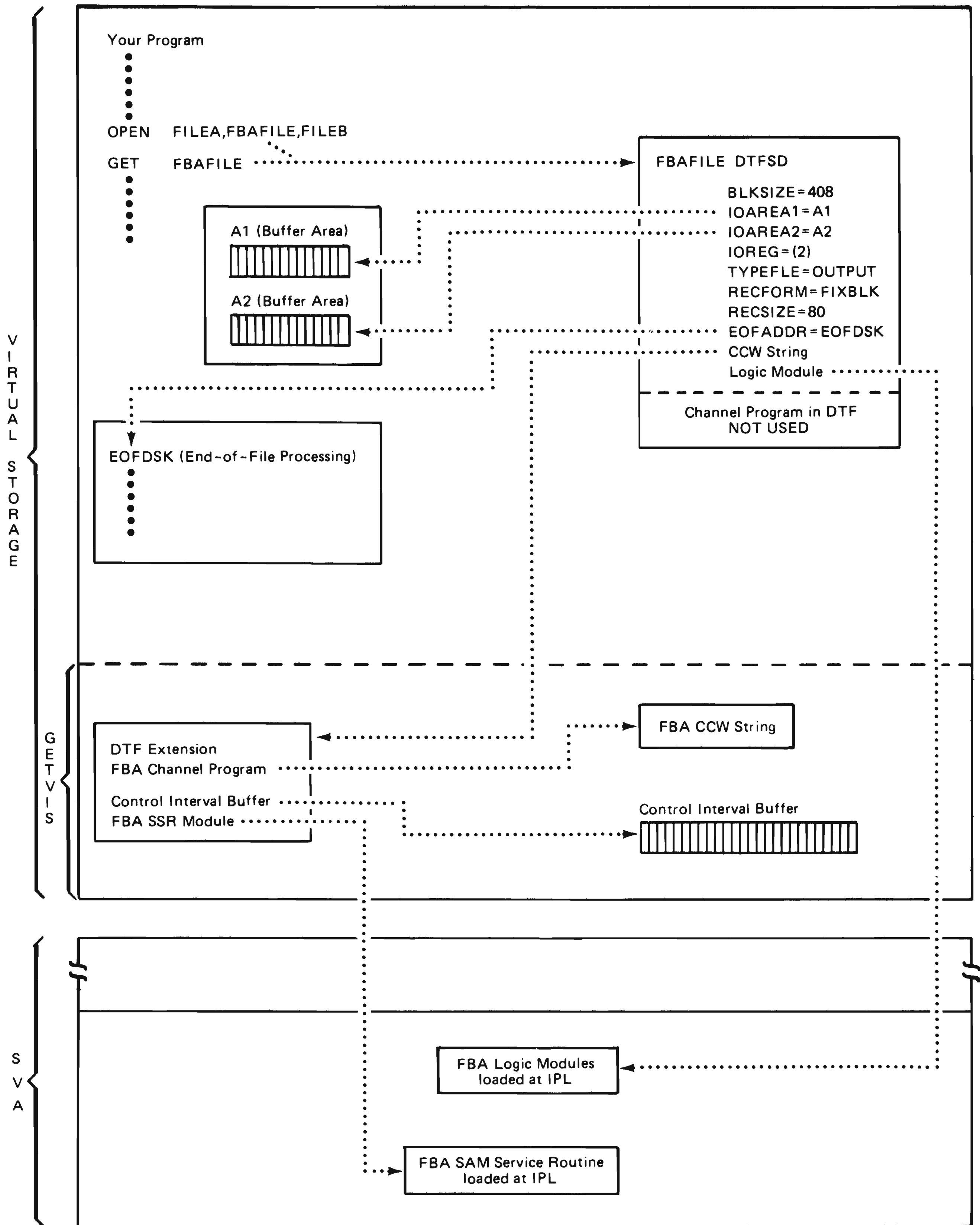


Figure B-9. DTFs and logic modules assembled jointly, with FBA DASD support included (Example 7).

Appendix C: Label Processing

The information contained in this section applies to the non-VSAM access methods available under VSE. For VSE/VSAM, the VSE/VSAM catalog management routines rather than the VSE label processing routines receive control, and it is those VSE/VSAM routines which perform functions such as making sure that the correct volume has been mounted and that no unexpired files are overwritten. Detailed information on label processing for VSE/VSAM files is found in the publications associated with the VSE/VSAM Program Product.

This section provides the information you need in order to process labels with the non-VSAM IOCS macros. More information about labeling conventions and label processing considerations will be found in *VSE System Data Management Concepts, VSE/Advanced Functions DASD Labels,* and *VSE/Advanced Functions Tape Labels,* as listed in the Preface.

DASD Standard Labels

Labels are required when processing files on direct access devices. Accordingly you must supply both a DASD label (DLBL) job control statement for each logical file to be processed, and one or more EXTENT job control statements to allocate one or more areas on a direct access device. More information will be found in *VSE/Advanced Functions System Control Statements,* as listed in the Preface.

OPEN Macro Processing

The OPEN macro uses the information supplied in the DLBL and EXTENT job control statements as well as information from the DTF for the file.

For input, the extent(s) for a file must either coincide with, or be within, the existing extent(s) as defined in the VTOC (Volume Table of Contents). This is necessary input because IOCS opens only an existing file or a subset of an existing file.

Note: If the extent(s) is within the existing extent(s), it is valid only for DTFSD or DTFPH (MOUNTED=SINGLE) processing.

For output, the file to be written cannot overlap existing unexpired files. IOCS does not destroy an unexpired file without your explicit request, except when an internal system file (IJSYS) overlays an identical system file. However, if OPEN determines that the output file will overlay an existing file that has expired, the OPEN deletes the expired label(s) from the VTOC. This in effect removes the file from the volume. In a multi-volume file, the file may be removed from all the volumes that it occupies or from only some of the volumes.

If OPEN determines that an existing file to be overlaid by the output file has not expired, the old file cannot be destroyed automatically. In this case, one of the following actions are possible:

1. Delete the unexpired file or
2. Terminate the job.

Reopening a File

If further processing of a file which your program has closed is required at some later time in the program, the file must be reopened. When a file is processed in sequential order, IOCS checks the label(s) on the first volume and makes the first extent available, the same as at the original OPEN. When a file is processed by physical IOCS with the MOUNTED=SINGLE operand of the DTFPH macro, IOCS opens the next extent specified by your EXTENT job control statement. When a file is processed by DAM (defined by the DTFDA macro), by ISAM (defined by the DTFIS macro), or by physical IOCS with the DTFPH operand MOUNTED=ALL specified, all label processing is repeated and all extents are again made available.

For more information on label processing see the discussion of the OPEN macro under the appropriate access method.

End-of-Volume Processing

During processing, IOCS recognizes an end-of-volume condition when the extents on one volume have been processed and an extent for another volume is encountered. When this condition occurs, IOCS branches to your LABADDR routine (if provided) to write or pass individually each user standard trailer label to be processed. After all user standard trailer labels are processed, IOCS processes the standard labels on the next volume and branches to your LABADDR routine to process user standard header labels. After the header labels are processed, IOCS continues to process the data.

End-of-File Processing

Output Files

When all records for a logical output file have been written, the CLOSE macro must be issued to perform normal end-of-file processing. IOCS then branches to your LABADDR routine (if provided) to write user trailer labels, and the file is closed. If the end of the

last extent specified for the file is reached before the CLOSE macro is issued, IOCS assumes an error condition.

Input Files

IOCS determines an end-of-file condition for a logical input file either by the ending address of the last extent specified for the file in the EXTENT job control statement, or by an end-of-file record read from the file. For SAM processing with DTFSD, IOCS branches to the EOFADDR routine upon an end-of-file condition. For sequential processing with DTFIS, IOCS posts the end-of-file condition in the field referred to as filenameC. You can then test this byte and take action necessary to close your file. However, when processing in random order you must determine the end-of-file by checking filenameC (DTFIS) or ERRBYTE (DTFDA).

User Standard Labels

If you want user standard labels, you must supply a LABADDR routine, SAM and DAM process both user header and trailer standard labels. ISAM does not process user standard labels. User labels cannot be created for a file whose first extent is a split cylinder extent. DAM writes a user trailer label only on the first volume of a multi-volume file.

When the LABADDR routine is entered, IOCS loads an alphabetic O, V, or F into the low-order byte of register 0. O indicates header labels, V indicates end-of-volume trailer labels, and F indicates end-of-file labels. Your LABADDR routine can test this character to determine the labels to be processed. IOCS also loads the address of an 80-byte IOCS label area in register 1; this is the address you use if you are checking labels, or from which you move the label to your program's label area if you are modifying labels.

Within the LABADDR routine, you cannot issue a macro that calls a transient routine (such as OPEN, CLOSE, DUMP, CANCEL, or CHKPT). For multi-volume files, the LABADDR routine should save registers 14 and 15 upon entry, and restore them before issuing the LBRET macro to return to IOCS.

Writing User Standard Labels on Disk

When you specify LABADDR, OPEN reserves the first track of the first data extent as a user label area. At least one user header and trailer label must be written if the access method is to process it. For DAM, when TRLBL=YES is specified with LABADDR, trailer labels are processed.

IOCS uses bytes 1 through 4 of the 80-byte label for the label identification (for example: UxLn, where x = H or T and n = 1,2, ..., 8). You can use the other 76 bytes as you wish. The maximum number of user standard header or trailer labels is eight for files on all DASDs. IOCS stores the label information (UHLn or UTLn) that it generates in bytes 1 through 4 of the IOCS label area. You can test this information, in addition to registers 0 and 1, to determine the type and number of the label. (The label formats will be found in *VSE/Advanced Functions DASD Labels,* as listed in the Preface.)

In your area of virtual storage, build either an 80-byte label, leaving the first four bytes free, or simply a 76-byte label. For the 80-byte label, load the address of the label area into register 0; for the 76-byte label, load the label area address minus four into register 0. Then issue the LBRET macro. When the label is moved into the IOCS area, IOCS adds four to the address in register 0, thus only moving the 76 bytes of user information into the IOCS label area.

When the label is ready to be written, the LBRET macro returns control to IOCS. If LBRET 2 is used, OPEN writes the label and returns control to your label routine unless the maximum number of labels has been written. If LBRET 1 is used, the label set is considered complete and no more labels can be created.

When IOCS receives control, the IOCS routine moves the label from the address you loaded into register 0 into the IOCS label area. If the maximum number of labels has not been written, IOCS increases the identification number by 1 and returns to your label routine unless LBRET 1 was used. If the maximum number of labels has been created, IOCS automatically terminates building of the label set.

Checking User Standard Labels on Disk

When a file on a DASD contains user standard labels, IOCS makes these labels available one at a time if LABADDR is specified in the DTF (see "DASD Standard Labels," above). If the labels are to be checked against information obtained from another input file, that file must be opened ahead of the file on a DASD.

When your program has finished checking a label, it can update it or leave it unmodified. If it is to be updated, your program must move the label to an area within the program before modifying it. After the label is modified, the program must initialize register 0 with the address of the modified label before issuing the LBRET 3 macro. The program then updates the appropriate label fields by issuing the LBRET 3 macro. This causes the OPEN routine to rewrite that label and read the next label. Register 1 points to the label in the IOCS label area. If the label

is to remain unmodified, you can issue a LBRET 2 macro so OPEN will read the next label. In either situation, if the end-of-file record is encountered at the end of the labels, OPEN automatically terminates label checking.

If you wish to end label checking before all the labels have been read, the LBRET 1 macro may be issued.

Diskette Labels

Labels are required when processing files on diskette I/O units. Accordingly, you must supply a DASD label (DLBL) job control statement for each logical file to be processed, and one or more EXTENT job control statements (more information will be found in *VSE/Advanced Functions System Control Statements,* as listed in the Preface).

OPEN Macro Processing

The OPEN macro uses information that is supplied in the DLBL and EXTENT job control statements, the DTF for the file, and the file label on the diskette.

For input, extent limits are taken directly from the file label in the VTOC on the diskette; extent limits provided in the extent statement(s) are ignored.

For output files, the extent limits for the file are determined by OPEN from available space on the diskette; extent limits provided by the user are ignored. If the name of the output file to be created is the same as that of an unexpired or write-protected file already present on the volume, OPEN will cause the job to be canceled. You will not be allowed to request that a duplicate file (unexpired or write-protected) be deleted. If the duplicate file has expired and is not write-protected or if the file to be created is not a duplicate one, OPEN will allocate space for the file, starting at the cylinder following the end of the last unexpired or write-protected file on the diskette. If expired and non-write-protected files are overlapped by this allocation, their labels are deleted from the VTOC.

End-of-Volume Processing

During processing, IOCS recognizes an end-of-volume condition when end-of-extent is reached on a volume and more extents are available. When this occurs, IOCS processes the standard labels on the next volume and continues to process the data.

End-of-File Processing

Output Files

When all records for an output logical file have been written, the CLOSE macro must be issued to perform normal end-of-file procedures. If the end of the last extent specified for the file is reached before the CLOSE macro is issued, IOCS assumes an error condition.

Input Files

IOCS determines an end-of-file for an input logical file by the end-of-data address. This address is specified in the file label in the VTOC of the last or only diskette of the file. IOCS branches to the EOFADDR routine upon an end-of-file condition.

Tape Labels

Tape Output Files

For output on magnetic tape, OPEN, CLOSE, or an end-of-volume condition rewinds the tape as specified in the DTFMT REWIND operand. No rewind can be defined in the DTFPH macro, and tape positioning depends on the labels to be processed and is your responsibility.

If you write any user standard labels, a LABADDR routine must be supplied. (For ASCII tape files, the LABADDR routine may be used only to process user standard labels.) Your LABADDR routine, specified in the DTF, cannot issue a macro that calls a transient routine. For example, OPEN, CLOSE, DUMP, CANCEL, and CHKPT cannot be issued. Also when processing multi-volume files, your label routine must save and restore register 15 if any logical IOCS macros other than LBRET are used. When user standard labels are written, they always follow the standard labels on the tape.

When all records of a file are processed, CLOSE can be issued to execute the EOF (end-of-file) routines. These routines write any record or blocks of records that are not already written. A partially filled record block is truncated; that is, a short block is written on the tape. Following the last record, IOCS writes a tapemark, the trailer labels, and two tapemarks, then executes the rewind option. If no trailer labels are written, two tapemarks are written and the rewind option is executed. In either case, if no rewind is specified and you have not specified any positioning, the tape is positioned between the two tapemarks at the end of the file.

If an EOV (end-of-volume) reflective marker is sensed on an output tape before a CLOSE is issued, logical IOCS prepares for closing the file by ensuring

that all records are written on the tape. If you issue another PUT, indicating that more records are to be written on this output file, EOV procedures are initiated. If you issue a CLOSE, the EOF procedures are initated.

Under certain conditions, an unfilled block of records may be written at an EOV or EOF condition, even though the file is defined as having fixed-length blocked records. When this file is used for input, logical IOCS recognizes and processes this short block. You need not be concerned or aware of the condition.

Label processing for the EOV condition resembles that for the EOF condition, except that a standard label is coded EOV instead of EOF. Also, only one tapemark is written after the label set or after the data for unlabeled files. In an ASCII file, two tapemarks follow the EOV labels.

When IOCS detects the EOV condition, it switches to an alternate unit as designated in an ASSGN job control statement. If an alternate drive is not specified, the operator is requested to mount a new volume (on the same drive) or cancel the job. When the operator mounts the volume, IOCS checks the standard header labels and processing continues.

In some cases, you may need to force an end-of-volume condition at a point other than the reflective marker. You may want to discontinue writing the records on the present volume and continue on another volume. This may be necessary because of some major change in category of records or in processing requirements. The FEOV (forced EOV) macro is available for this function (see "Forcing End-of-Volume" in "Chapter 6. Processing Magnetic Tape Files").

Writing Standard Labels

When standard labels are written (DTFMT FILABL=STD or DTFPH TYPEFLE=OUTPUT), you must supply the TLBL job control statement for standard label information. More information will be found in *VSE/Advanced Functions System Control Statements,* as listed in the Preface.

When an OPEN macro is issued and the tape is positioned at load point, the volume (VOL1) label is checked. Whether at load point or not, the old file header, if present, is read and checked to make sure that the file on the tape is no longer active and may be over-written. If the file is inactive or if a tapemark was read, the tape is backspaced and the new file header (HDR1) label is written with the information you supply in the tape label statement. The volume label is not rewritten, altered, or updated.

A comparison is made between the specified density and the VOL1 density of the expired tape. If a discrepancy is found and the tape is at load point, the volume label(s) is (are) rewritten according to the specified density.

If an output file begins in the middle of a reel, it is your responsibility to properly position the tape immediately past the tapemark for the preceding file before issuing the OPEN macro. The MTC command can be used to do this. If the tape is improperly positioned, IOCS issues an appropriate message to the operator.

If user standard labels are written, the LABADDR operand must be specified in the DTF. After writing the standard label (header or trailer), IOCS loads register 0 (low-order byte) as follows:

O indicates header labels.

V indicates end-of-volume labels.

F indicates end-of-file labels.

Your LABADDR routine can test this character to determine what labels should be written. IOCS also loads the address of an 80-byte IOCS label area in register 1; this is the address you use if checking labels, or from which you move the label to your program's label area if you are modifying labels.

Note: For ASCII files, you process your standard labels in EBCDIC.

A maximum of eight user standard header (UHL), or trailer (UTL) labels can be written following the standard header (HDR1), or trailer (EOV1 or EOF1) labels. The user standard labels are 80 bytes long and are built entirely by you. Bytes 1 through 4 must contain the label identification (UxLn, where x=H or T and n=1, 2, ..., 8); the other 76 bytes can be used as desired.

For ASCII tape files, you can have any number of user standard header or trailer labels. To comply with the standards for an ASCII file, these labels are identified by UHL*a* and UTL*a,* where *a* represents an ASCII character in the range 2/0 through 5/14, excluding 2/7 (apostrophe). The remaining 76 bytes can be used as desired. It is your responsibility to ensure that labels contain UHLa and UTLa in the first four bytes.

Note: When creating user header and trailer labels for 7-track tapes, only unpacked data is valid in the 76-byte data portion of the label.

You should build your labels in your area of virtual storage, and load the address of the label into register 0 before issuing the LBRET macro.

When the label is ready to be written, you issue the LBRET macro, which returns control to IOCS. If

LBRET 2 is used, IOCS writes the label and returns control to your label routine. If LBRET 1 is used, the label set is terminated and no more labels can be created. When IOCS receives control, IOCS writes the label on the magnetic tape and either returns control (LBRET 2) or writes a tapemark (LBRET 1).

When a standard trailer label is written, IOCS accumulates the block count for the label when logical IOCS is used. However, if physical IOCS (DTFPH) is used, your program must accumulate the block count, if desired, and supply it to IOCS for inclusion in the standard trailer label. For this, the count (in binary form) must be moved to the 4-byte field within the DTF table named filenameB. For example, if the filename specified in the DTFPH header name is DELTOUT, the block count field is addressed by DELTOUTB.

If checkpoint records are interspersed among data records on an output tape, the block count accumulated by logical IOCS does not include a count of the checkpoint records. Only data records are counted. Similarly, if physical IOCS is used, your program must omit checkpoint records and count data records only.

After all trailer labels (including user labels, if any) are written at end-of-volume or end-of-file, IOCS initiates the EOF or EOV routines.

Writing Nonstandard Labels

To write nonstandard labels, you must specify FILABL=NSTD and LABADDR=name. When the file is opened, the tape must be positioned to the first label that you wish to process. The MTC job control statement can be used to skip the necessary number of tapemarks or records to position the file. You must also write your own channel program and use physical IOCS macros to transfer the labels from virtual storage onto tape.

When a file is opened or closed, or when a volume is finished, IOCS supplies the hexadecimal representation (in the two low-order bytes of register 1) of the symbolic unit currently in use. See bytes 6 and 7 of the CCB for these values. IOCS also loads into the low-order byte of register 0 one of the following:

O to indicate header labels.

V to indicate end-of-volume labels.

F to indicate end-of-file labels.

Your LABADDR routine can then test this character to determine the type of labels to be written.

In your LABADDR routine, physical IOCS macros must be used to transfer labels from virtual storage onto tape. For each label record, a CCB and CCW must be established, and the EXCP macro must be issued. Other logical IOCS macros can be used for any processing other than the transfer of labels from virtual storage to tape. Additional LABADDR routine restrictions have been discussed above.

After all labels are written, you return control to IOCS by use of the LBRET 2 macro. IOCS processing after LBRET is executed has been discussed above.

Note: Nonstandard labels are not permitted with ASCII.

Writing Unlabeled Files

If you use unlabeled files, you should specify FILABL=NO and omit TPMARK=NO in the DTF to improve the efficiency of your program. Your file must be positioned properly with the MTC job control statement, if necessary, and writing begins immediately. Other processing information can be found under "Tape Input Files," below.

For unlabeled ASCII files, TPMARK=NO is the only valid entry. If the operand is omitted entirely, TPMARK=NO is the default. Leading tapemarks are not supported on unlabeled ASCII files. Special error recovery procedures facilitate reading backwards.

Tape Input Files

For a magnetic tape input file, the macros OPEN, CLOSE, or an end-of-volume condition cause the tape to be rewound as specified by the DTFMT REWIND parameter. No rewind can be defined in the DTFPH macro. Tape positioning depends on the labels to be processed and is your responsibility.

If any labels other than standard labels are to be checked, a LABADDR routine must be supplied. Your LABADDR routine, specified in the DTF, cannot issue a macro that calls a transient routine. This is the same as for the tape output files.

When an end-of-file condition occurs, IOCS branches to your EOFADDR routine specified in the DTF. Generally, you issue a CLOSE in this routine to initiate a rewind operation for the tape (as specified by the DTF REWIND operand), and deactivate the file. If CLOSE is issued before the end of data is reached, the rewind option is executed and the file is deactivated without any subsequent label checking.

When logical IOCS reads a tapemark on a tape input file, either an end-of-file or end-of-volume condition exists. This condition is determined by IOCS or by yourself, depending on the type of labels (if any) used for the file. IOCS performs the appropriate functions.

IOCS can determine an end-of-volume condition only when trailer labels have been checked (see

"Checking Standard Labels" or "Checking Nonstandard Labels," below). If labels are not processed, your EOFADDR routine must process the condition (see "Forcing End-of-Volume" in "Chapter 6. Processing Magnetic Tape Files"). When IOCS does detect the EOV condition, it switches to an alternate unit as designated in an ASSGN job control statement. If an alternate drive is not specified, a message to mount a new volume is issued. At this time, the operator may also cancel the job. When the operator mounts the volume, processing resumes. If the input file is processed by physical IOCS (DTFPH), you must issue an OPEN macro for the new volume. Then, IOCS checks the header label(s) and processing continues.

In some cases, you may desire to force an end-of-volume condition at a point other than at the normal tapemark. You may want to discontinue reading the records on the present volume and continue reading records on the next volume. This may be necessary because of some major change in record category or in processing requirements. An FEOV (forced end-of-volume) macro is available for such cases.

Reading a Tape Backwards

When reading backwards (READ=BACK), a labeled tape must be positioned so that the first record read, when OPEN is executed, is the tapemark physically following the trailer labels. An unlabeled file must be positioned so that the first record read, when OPEN is executed, is the tapemark physically following the first logical data record to be read (the last record written when the file was created). Although ASCII unlabeled tapes contain no leading tapemark, special error recovery procedures allow these tapes to be read backwards.

Label checking of standard and nonstandard labels is similar. That is, IOCS still processes standard labels, and your routine (if specified) still processes user or nonstandard labels. The only difference is that the volume label is not read immediately for standard labels, the trailer labels are processed in reverse order (relative to writing), and header labels are processed at EOF time, also in reverse order. If physical IOCS macros are used to read records backwards, labels cannot be checked (DTFPH must not be specified).

Because backwards reading is confined to one volume, an end-of-file condition always exists when the header label is encountered. At end-of-file for standard labels, IOCS checks only the block count (which was stored from the trailer label) and then branches to your EOFADDR routine. At EOF for nonstandard labels, IOCS branches to your LABADDR routine where the header label may be checked. To check labels, you must evoke physical IOCS macros to read the label(s).

Your LABADDR routine, specified in the DTF, cannot issue a macro that calls a transient routine. For example, OPEN, CLOSE, DUMP, CANCEL, or CHKPT cannot be issued. Also, when processing multivolume files, your label routine must save and restore register 15 if any logical IOCS macros other than LBRET are used. When user standard labels are checked, the checking is the same as that for standard labels.

Checking Standard Labels

When standard labels are to be checked (DTFMT FILABL=STD or DTFPH TYPEFLE=INPUT), you must supply the TLBL job control statement for standard label information. More information is found in *VSE/Advanced Functions System Control Statements,* as listed in the Preface.

When standard labeled files positioned at load point are opened, IOCS requires that the first record be a volume (VOL1) label. The next label could be any HDR1 label preceding the file. IOCS locates the correct file header (HDR1) label by checking the file sequence number.

After checking the standard label (if user standard labels UHL1 through UHL8 or UTL1 through UTL8 are present for EBCDIC files, or UHLa or UTLa for ASCII files), IOCS enters the LABADDR routine and inserts one of the following in the low-order byte of register 0:

O to indicate header labels.

V to indicate end-of-volume labels.

F to indicate end-of-file labels.

Your routine can test this character to determine what labels should be checked. IOCS also loads the address of an 80-byte IOCS label area in register 1; this is the address you use if checking labels, or from which you move the label to your program's label area if you are modifying labels.

After each label is checked, a LBRET 2 macro can be issued for IOCS to read the next label. However, if a tapemark is read instead, label checking is terminated. If you wish to end label checking before all labels are read, you can issue a LBRET 1 macro. After all trailer labels are checked, IOCS initiates EOV or EOF procedures.

Checking Nonstandard Labels

Any tape labels not conforming to the standard label specifications are considered nonstandard. It is your responsibility to check such labels if they are present. The MTC job control statement can be issued to

skip the necessary number of tapemarks or records to position the file. On input, nonstandard labels may or may not be followed by a tapemark. The following possible conditions can thus be encountered:

1. One or more labels, followed by a tapemark, are to be checked.
2. One or more labels, not followed by a tapemark, are to be checked.
3. One or more labels, followed by a tapemark, are not to be checked.
4. One or more labels, not followed by a tapemark, are not to be checked.

For conditions 1 and 2, the DTFMT operands FILABL=NSTD and LABADDR=name must be specified. For condition 3, the operand FILABL=NSTD must be specified. If LABADDR is omitted, IOCS skips all labels, bypasses the tapemark, and positions the tape at the first data record to be read. For condition 4, the entries FILABL=NSTD and LABADDR=name must be specified. In this case, IOCS cannot distinguish labels from data records because there is no tapemark to indicate the end of the labels. Therefore, you must read all labels even though checking is not desired to position the tape at the first data record.

Each time IOCS opens a file or reads a tapemark, it supplies (in the low-order bytes of register 1) the hexadecimal representation of the symbolic unit currently used. These values are as shown in the layout description of CCB bytes 6 and 7 in Figure 9-3. IOCS also loads the character O into the low-order byte of register 0 when the file is opened.

When your routine gains control, the tape is not moved by OPEN. Physical IOCS macros must be used to transfer labels from tape to virtual storage. Therefore, you must establish a CCB and a CCW. The EXCP macro is used to initiate the transfer. After all labels are checked, you return control to OPEN by use of the LBRET 2 macro.

When IOCS reads a tapemark, it checks to determine if you have supplied a LABADDR routine. If a LABADDR routine was supplied, IOCS exits to the routine. Otherwise, IOCS skips the labels and branches to the EOFADDR routine. In the LABADDR routine, you must use physical IOCS macros to read your label(s). Furthermore, you must determine the EOF and/or EOV condition and indicate to IOCS which condition exists by loading either EF (end-of-file) or EV (end-of-volume) into the two low-order bytes of register 0. When this information is passed to IOCS, it initiates the end-of-file or end-of-volume procedures.

Unlabeled Input Files

The first record for unlabeled tapes (FILABL=NO) may or may not contain a tapemark. Unlabeled tapes with ASCII contain no leading tapemark. If a tapemark is present, the next record is considered to be the first data record. If there is no tapemark, IOCS reads the first record, determines that it is not a tapemark, and backspaces to the beginning of that record. The file can be properly positioned by use of the MTC job control statement. When the tapemark following the last data record is read, IOCS branches to the end-of-file address.

Label Processing With Access Control Option

On systems with VSE/ICCF installed, you can take advantage of the protection offered when processing standard labeled or unlabeled tape input files.

If the access control option is present in your system (SEC=xx in the FOPT macro; where xx is a value between 10 and 32K-1), the tape label processing routines protect against unauthorized use or accidental destruction of file data. This is accomplished by not allowing an "IGNORE" response to various label processing messages that would otherwise permit skipping over label errors or processing labeled input files as if they were unlabeled files.

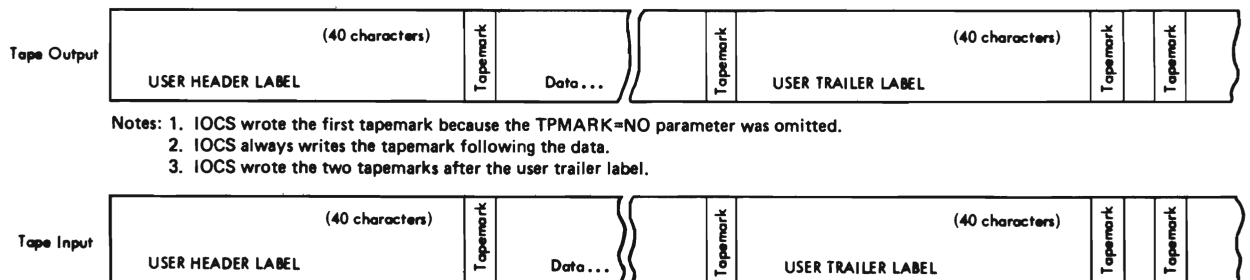
Reading, Writing, and Checking with Nonstandard Labels

```
                                                   EXTERNAL SYMBOL DICTIONARY                                        PAGE    1
  SYMBOL   TYPE ID  ADDR   LENGTH LD ID

           PC   01 003000 00048C
IJCFZIZ0   ER   02
IJFFZZZZ   ER   03
IJFFBZZZ   ER   04
IJDFZZZZ   ER   05
IJ2L0006   SD   06 00349C 000064
```

```
       TEST  CREATING AND PROCESSING NON-STANDARD LABELS                                                              PAGE    1

  LOC  OBJECT CODE     ADDR1 ADDR2  STMT    SOURCE STATEMENT

                                       2            PRINT ON,NOGEN,NODATA                                                NSTD0004
003000                                 3            START 12288                                                          NSTD0005
                                       4 *                                                                               NSTD0006
                                       5 READER     DTFCD DEVICE=2540,DEVADDR=SYSIPT,BLKSIZE=80,TYPEFLE=INPUT,          *NSTD0007
                                                          EOFADDR=ENDCARD,IOAREA1=IOAREA                                 NSTD0008
                                      26 *                                                                               NSTD0009
                                      27 TAPEOUT    DTFMT DEVADDR=SYS004,IOAREA1=IOAREA,BLKSIZE=80,TYPEFLE=OUTPUT,*NSTD0010
                                                          LABADDR=LABELOUT,READ=FORWARD,FILABL=NSTD                      NSTD0011
                                      58 *                                                                               NSTD0012
                                      59 TAPEIN     DTFMT DEVADDR=SYS004,IOAREA1=IOAREA,BLKSIZE=80,TYPEFLE=INPUT,       *NSTD0013
                                                          EOFADDR=ENDTAPE,READ=FORWARD,FILABL=NSTD,REWIND=NORWD,         *NSTD0014
                                                          LABADDR=LABELIN                                                NSTD0015
                                      93 *                                                                               NSTD0016
                                      94 TAPEIN2    DTFMT DEVADDR=SYS004,IOAREA1=IOAREA,BLKSIZE=80,TYPEFLE=INPUT,       *NSTD0017
                                                          EOFADDR=ENDTAPE2,READ=BACK,FILABL=NSTD                         NSTD0018
                                     129 *                                                                               NSTD0019
                                     130 PRINT      DTFPR DEVICE=1403,DEVADDR=SYSLST,IOAREA1=IOAREA,BLKSIZE=80          NSTD0020
                                     151 *                                                                               NSTD0021
                                     152 CONSOLE    DTFCN BLKSIZE=80,DEVADDR=SYSLOG,IOAREA1=CAREA,RECFORM=FIXUNB,       *NSTD0022
                                                          WORKA=YES                                                      NSTD0023
                                     221 *                                                                               NSTD0024
                                     222 *                                                                               NSTD0025
0031AC 0520                          223 START      BALR  2,0                         SET UP A BASE REGISTER             NSTD0026
0031AE                               224            USING *,2                                                            NSTD0027
                                     225 * ** ROUTINE TO WRITE TAPE                                                      NSTD0028
                                     226            OPEN  TAPEOUT                     TO WRITE NSTD RECORDS              NSTD0029
                                     234 GETCARD    GET   READER                      READ A CARD FROM CARD READER       NSTD0030
                                     239            PUT   TAPEOUT                     WRITE CARD IMAGE ON TAPE           NSTD0031
0031D6 47F0 2010               031BE 244            B     GETCARD                     BRANCH AND GET ANOTHER CARD        NSTD0032
                                     245 ENDCARD    CLOSE TAPEOUT                     TO WRITE NSTD TRAILER LABEL        NSTD0033
                                     253 * ** ROUTINE TO READ TAPE FORWARD                                               NSTD0034
                                     254            OPEN  PRINT,TAPEIN                TO PROCESS NSTD LABEL              NSTD0035
                                     263 GETTAPE    GET   TAPEIN                      GET A CARD IMAGE FROM TAPE         NSTD0036
                                     268            PUT   PRINT                       PRINT CARD IMAGE ON PRINTER        NSTD0037
003216 47F0 2050               031FE 273            B     GETTAPE            BRANCH AND GET ANOTHER TAPE RECORD          NSTD0038
                                     274 ENDTAPE    CLOSE TAPEIN                      PROCESS NSTD LABELS                NSTD0039
                                     282 * ** ROUTINE TO READ TAPE BACKWARDS                                             NSTD0040
                                     283            OPEN  TAPEIN2                     BYPASS NSTD LABELS                 NSTD0041
                                     291 GETTAPE2   GET   TAPEIN2                     READ A TAPE RECORD                 NSTD0042
                                     296            PUT   PRINT                       PRINT RECORD                       NSTD0043
003252 47F0 208C               0323A 301            B     GETTAPE2           BRANCH AND GET ANOTHER TAPE RECORD          NSTD0044
                                     302 ENDTAPE2   CLOSE PRINT,TAPEIN2               BYPASS NSTD RECORDS                NSTD0045
                                     311            CNTRL TAPEIN2,REW                 REWIND TAPE TO LOAD POINT          NSTD0046
                                     317            EOJ                               NORMAL END OF JOB                  NSTD0047
                                     320 * ** LABEL CREATION ROUTINE                                                     NSTD0048
00327A 4900 22A8               03454 321 LABELOUT   CH    0,ALPHAO                    OPEN OR CLOSE                      NSTD0049
00327E 4770 20F0               0329E 322            BNE   TRAILOUT                    BRANCH IF CLOSE                    NSTD0050
003282 D227 221C 21CC 033CA 0337A 323            MVC   IOAREA(40),HEADER           MOVE HEADER TO I/O AREA            NSTD0051
                                     324 RITELAB    EXCP  OUTCCB                      WRITE LABEL                        NSTD0052
                                     328            WAIT  OUTCCB                      WAIT FOR COMPLETION                NSTD0053
                                     334            LBRET 2                           RETURN CONTROL TO IOCS             NSTD0054
```

Figure C-1. Reading, writing, and checking with nonstandard labels (Part 1 of 2).

```
TEST  CREATING AND PROCESSING NON-STANDARD LABELS                                                                PAGE    2

  LOC  OBJECT CODE     ADDR1 ADDR2  STMT    SOURCE STATEMENT

00329E D227 221C 21F4 033CA 033A2   337 TRAILOUT MVC    IOAREA(40),TRAILER     MOVE TRAILER LABEL TO I/O AREA    NSTD0055
0032A4 47F0 200A            03288   338          B      RITELAB                BRANCH TO WRITE THE LABEL         NSTD0056
                                    339 * ** LABEL PROCESSING ROUTINE                                             NSTD0057
0032A8 4900 22A6            03454   340 LABELIN  CH     0,ALPHAO               OPEN OR CLOSE                     NSTD0058
0032AC 4780 212C            032CA   341          BE     HEADIN                 OPEN TIME                         NSTD0059
                                    342 TRAILIN  EXCP   INCCB                  READ A TRAILER LABEL              NSTD0060
                                    346          WAIT   INCCB                  WAIT FOR I/O COMPLETION           NSTD0061
0032C4 9101 2270      0341F         352          TM     INCCB+4,X'01'          TEST FOR A TAPE MARK              NSTD0062
0032C8 4710 2164            03312   353          BO     EXITEOF                BRANCH IF YES                     NSTD0063
0032CC D527 221C 21F4 033CA 033A2   354          CLC    IOAREA(40),TRAILER     COMPARE TRAILER LABEL             NSTD0064
0032D2 4780 2102            032B0   355          BE     TRAILIN                BRANCH TO GET ANOTHER RECORD      NSTD0065
0032D6 47F0 2152            03300   356          B      ERRLAB          BRANCH IF LABELS DO NOT COMPARE          NSTD0066
                                    357 HEADIN   EXCP   INCCB                  READ A HEADER LABEL               NSTD0067
                                    361          WAIT   INCCB                  WAIT.FOR COMPLETION               NSTD0068
0032EE 9101 2270      0341E         367          TM     INCCB+4,X'01'          TEST FOR A TAPE MARK              NSTD0069
0032F2 4710 2168            0331E   368          BO     EXIT                   BRANCH IF YES                     NSTD0070
0032F6 D527 221C 21CC 033CA 0337A   369          CLC    IOAREA(40),HEADER      DOES HEADER LABEL COMPARE         NSTD0071
0032FC 4780 212C            032CA   370          BE     HEADIN                 IF YES, BRANCH AND READ TAPE      NSTD0072
                                    371 ERRLAB   PUT    CONSOLE,LABELERR       PUT LABEL ERROR MESSAGE           NSTD0073
                                    377          EOJ                           TERMINATE JOB                     NSTD0074
003312 5800 22A2            03450   380 EXITEOF  L      0,EOFIND               INDICATE EOF TO IOCS              NSTD0075
                                    381 EXIT     LBRET 2                       RETURN CONTROL TO IOCS            NSTD0076
                                    384 * CONSTANTS                                                               NSTD0077
003318 4040404040404040             385 CAREA    DC     CL50' '                CONSOLE I/O AREA                  NSTD0078
00334A E4E2C5D940D3C1C2             386 LABELERR DC     C'USER LABELS DO NOT COMPARE. ABNORMAL END OF JOB.'      NSTD0079
00337A E4E2C5D940C8C5C1             387 HEADER   DC     CL40'USER HEADER LABEL'                                  NSTD0080
0033A2 E4E2C5D940E3D9C1             388 TRAILER  DC     CL40'USER TRAILER LABEL'                                 NSTD0081
0033CA 4040404040404040             389 IOAREA   DC     CL80' '                INPUT/OUTPUT AREA                 NSTD0082
                                    390 INCCB    CCB    SYS004,INCCW           READ TAPE CCB                     NSTD0083
                                    401 OUTCCB   CCB    SYS004,OUTCCW          WRITE TAPE CCB                    NSTD0084
00343A 000000000000
003440 020033CA00000028             412 INCCW    CCW    X'02',IOAREA,X'00',40  READ TAPE CCW                     NSTD0085
003448 010033CA00000028             413 OUTCCW   CCW    X'01',IOAREA,X'00',40  WRITE TAPE CCW                    NSTD0086
003450 0000C5C6                     414 EOFIND   DC     X'0000C5C6'                                              NSTD0088
003454 00D6                         415 ALPHAO   DC     X'00D6'                                                  NSTD0089
0031AC                              416          END    START                                                    NSTD0090
003458 5B5BC2D6D7C5D540             417                 =C'$$BOPEN '
003460 5B5BC2C3D3D6E2C5             418                 =C'$$BCLOSE'
003468 00003000                     419                 =A(READER)
00346C 00003038                     420                 =A(TAPEOUT)
003470 00003090                     421                 =A(TAPEIN)
003474 00003150                     422                 =A(PRINT)
003478 000030F0                     423                 =A(TAPEIN2)
00347C 0000342A                     424                 =A(OUTCCB)
003480 0000341A                     425                 =A(INCCB)
003484 0000318C                     426                 =A(CONSOLE)
003488 0000334A                     427                 =A(LABELERR)
```



Notes: 1. IOCS wrote the first tapemark because the TPMARK=NO parameter was omitted.
2. IOCS always writes the tapemark following the data.
3. IOCS wrote the two tapemarks after the user trailer label.

Notes: 1. IOCS reads the first tapemark or bypasses it if user labels are not checked.
2. Upon encountering the second tapemark IOCS branches to your label routine address.
3. After you read the third tapemark you should issue a LBRET 1 and IOCS will branch to the end-of-file address.

Figure C-1. Reading, writing, and checking with nonstandard labels (Part 2 of 2).

Appendix D: Writing Self-Relocating Programs

A self-relocating program is one that can be loaded for execution at any location in virtual storage. While having this distinct advantage, self-relocating programs are slightly more time consuming to write and they usually require slightly more storage. And, they may only be written in assembler language. For these reasons you may want to make use of the relocating loader instead. The relocating loader, standard in VSE (available as a system generation option in DOS/VS), accomplishes the same thing as writing self-relocating programs but without any of these disadvantages.

However, prior to the availability of a relocating loader, some users coded self-relocating programs to gain the advantage of running them in any one of the available partitions without their having to be link-edited again. When the program was link-edited, OPTION CATAL and a PHASE statement such as:

```
PHASE   phasename,+0
```

were used. This caused the linkage editor to assume that the program was loaded at storage location zero, and to compute all absolute addresses from the beginning of the phase. The job control EXEC function recognized a zero phase address to compensate for the current partition boundary save area. Control was then given to the updated entry address of the phase. Programs that were written using self-relocating techniques could be cataloged as either self-relocating or non-self-relocating phases.

If you have to perform maintenance on such a program, you must write this program in assembler language according to the same rules under which the program was originally written.

Rules for Writing Self-Relocating Programs

In general, if a problem program is written to be self-relocating, the rules below must be followed. Rules 1 through 5 apply to any program that is to be self-relocating. Rules 6 through 8 apply only to those self-relocating programs which consist of two or more control sections.

1. The PHASE card must specify an origin of +0.
2. The program must relocate all address constants used in the program. Whenever possible, use the LA instruction to load an address in a register instead of using an A-type address constant. For example, instead of writing:

```
        USING    *,12
        BALR     12,0
        LA       12,0(12)
        BCTR     12,0
        BCTR     12,0
        LA       1,EOF
        ST       1,AEOF
        •
        •
        L        10,AEOF
        •
        •
EOF     EOJ
        •
        •
AEOF    DC       A(EOF)
```

Code your program like this:

```
        USING    *,12
        BALR     12,0
        LA       12,0(12)
        BCTR     12,0
        BCTR     12,0
        •
        •
        LA       10,EOF
        •
EOF     EOJ
```

3. If logical IOCS is used, the program must use the OPENR and CLOSER macros to open and close all files, including console files.
4. If physical IOCS is used, the program must relocate the CCW address field in the CCB or IORB, and the address field in each CCW.
5. Register notation must be used with imperative I/O macros and supervisor macros. An example of coding the GET macro with a work area in self-relocating format follows:

```
RCARDIN  EQU    4
RPRTOUT  EQU    5
RWORK    EQU    6
         LA     RCARDIN,CARDIN
         LA     RPRTOUT,PRTOUT
         LA     RWORK,WORK
         OPENR  (RCARDIN),(RPRTOUT)
         •
         •
         •
         GET    (RCARDIN),(RWORK)
```

Note: Since the DTF name can be a maximum of seven characters, an R can be prefixed to this name to identify the file. Thus, RCARDIN in this example can immediately be associated with the corresponding DTF name CARDIN.

6. The relocation factor should be calculated and stored in a register for future use. For register economy, the base register can hold the relocation factor. For example:

```
USING    *,12
BALR     12,0
LA       12,0(12)
BCTR     12,0
BCTR     12,0
```

   Register 12 now contains the relocation factor and the program base.

7. When branching to an external address, use one of the following techniques:

```
L        15,=V(EXTERNAL)
BAL      14,0(12,15)
         or
L        15,=V(EXTERNAL)
AR       15,12
BALR     14,15
```

   where register 12 is the base register.

8. The calling program is responsible for relocating all address constants in the calling list(s). See Figure D-1 for an example of the calling program relocating the address constants in a calling list.

Programming Techniques

A self-relocating program is capable of proper execution regardless of where it is loaded. DTFDI should be used to resolve the problem of device differences between partitions. A self-relocating program must also adjust all its own absolute addresses to point to the proper address. This must be done after the program is loaded, and before the absolute addresses are used.

Within these self-relocating programs, some macros generate self-relocating code. For example the OPENR and CLOSER macros, which can be used in place of OPEN and CLOSE, adjust all of the address constants in the DTFs opened and closed. OPENR and CLOSER can be used in any program because the OPENR macro computes the amount of relocation. If relocation is 0, the standard open is executed. In addition, all of the module generation (xxMOD) macros generate self-relocating code.

```
// JOB A
// OPTION LINK
// EXEC ASSEMBLY
CSECT1      START  0
            USING  *,12                   Use load point value as the base to
            BALR   12,0                   find the load point value.
            LA     12,0(12)
            BCTR   12,0
            BCTR   12,0
            •
            LA     1,A
            LA     2,B
            LA     3,C                    Modify the CALL address constant list.
            LA     4,D
            STM    1,4,LIST
            OI     LIST+12,X'80'          Restore end of list bit in last adcon.
            LA     13,SAVEAREA
            L      15,=V(EXTERNAL)
            AR     15,12                  Adjust CALL address by relocating Factor.
            CALL   (15),(A,B,C,D)
LIST        EQU    *-16                   For`address constants (4 bytes each).
            EOJ
SAVEAREA    DC     9D'0'
            END
/* *
// EXEC ASSEMBLY
CSECT       START  0
            ENTRY  EXTERNAL
EXTERNAL    SAVE   (14,12)
            USING  *,12
            BALR   12,0                   Establish new base
            •
            •
            RETURN (14,12)
            END
/* *
// EXEC LNKEDT
/&
```

Figure D-1. Relocating address constants in a calling list.

The addresses of all address constants containing relocatable values are listed in the relocation dictionary in the assembly listing. This dictionary includes both those address constants that are modified by self-relocating macros, and those that are not. The address constants not modified by self-relocating macros must be modified by some other technique. After the program has been link-edited with a phase origin of +0, the contents of each address constant is the displacement from the beginning of the phase to the address pointed to by that address constant.

The following techniques place relocated absolute addresses in address constants. These techniques are required only when the LA instruction cannot be used.

Technique 1: Code named A-type address constants:

```
      •
      LA      4,ADCONAME
      ST      4,ADCON
      •
      •
ADCON DC      A(ADCONAME)
```

Techique 2: Place A-type address constants in the literal pool:

```
      •
      LA      3,=A(ADCONAME)
      LA      4,ADCONAME
      ST      4,0(3)
      •
      LTORG   =A(ADCONAME)
```

Technique 3: Code A-type address constants with a specified length of three bytes, and a non-zero value in the adjacent left byte (as in CCWs):

A. If the CCW list dynamically changes during program execution:

```
         •
         LA     4,IOAREA
         STCM   4,X'07',TAPECCW+1
         •
TAPECCW  CCW    1,IOAREA,X'20',100
         •
IOAREA   DS     CL100
```

B. If the CCW list is static during program execution:

```
         •
         LA     4,IOAREA
         ST     4,TAPECCW
         MVI    TAPECCW,1
         •
TAPECCW  CCW    1,IOAREA,X'20',100
         •
IOAREA   DS     CL100
```

or

```
         •
         USING  *,12
         BALR
         LA     12,0(12)
         BCTR   12,0
         BCTR   12,0 Reg 12 contains
                     relocation factor
         •
         L      11,TAPECCW
         ALR    11,12
         ST     11,TAPECCW
         •
TAPECCW  CCW    1,IOAREA,X'20',100
         •
IOAREA   DS     CL100
```

Technique 4: Use named V-type or A-type address constants:

```
         •
         •
         LA  3,ADCONAST  Determine
         S   3,ADCONAST  relocation
                         factor
         •
         L   4,ADCON
         AR  4,3         Add reloc.
                         factor
         ST  4,ADCON
         •
ADCONAST DC  A(*)
ADCON    DC  V(NAME)
```

The load point of the phase is not synonymous with the relocation factor as developed in register 3 (technique 4). If the load point of the phase is taken from register 0 (or calculated by a BALR and subtracting 2) immediately after the phase is loaded, correct results are obtained if the phase is link-edited with an origin of +0. If a phase is link-edited with an origin of * or S, incorrect results will follow because the linkage editor and the program have both added the load point to all address constants. Figure D-2 shows an example of a self-relocating program.

```
             REPRO
             PHASE  EXAMPLE,+0                +0 ORIGIN IMPLIES SELF-RELOCATION
             PRINT  NOGEN
PROGRAM      START  0
             BALR   12,0
             USING  *,12
*       ROUTINE TO RELOCATE ADDRESS CONSTANTS
             LA     1,PRINTCCW                RELOCATE CCW ADDRESS
             ST     1,PRINTCCB+8               IN CCB FOR PRINTER
             LA     1,TAPECCW                 RELOCATE CCW ADDRESS
             ST     1,TAPECCB+8                IN CCB FOR INPUT TAPE
             IC     2,PRINTCCW                SAVE PRINT CCW OP CODE
             LA     1,OUTAREA                 RELOCATE OUTPUT AREA ADDRESS
             ST     1,PRINTCCW                 IN PRINTER CCW
             STC    2,PRINTCCW                RESTORE PRINT CCW OP CODE
             LA     1,INAREA                  RELOCATE INPUT AREA ADDRESS
             ST     1,TAPECCW                  IN TAPE CCW
             MVI    TAPECCW,READ              SET TAPE CCW OP CODE TO READ
*       MAIN ROUTINE...READ TAPE AND PRINT RECORDS
READTAPE     LA     1,TAPECCB                 GET CCB ADDRESS
             EXCP   (1)                       READ ONE RECORD FROM TAPE
             WAIT   (1)                       WAIT FOR I/O COMPLETION
             LA     10,EOFTAPE                GET ADDRESS OF TAPE EOF ROUTINE
             BAL    14,CHECK                  GO TO UNIT EXCEPTION SUBROUTINE
             MVC    OUTAREA(10),INAREA        EDIT RECORD
             MVC    OUTAREA+15(70),INAREA+10  IN
             MVC    OUTAREA+90(20),INAREA+80  OUTPUT AREA
             LA     1,PRINTCCB                GET CCB ADDRESS
             EXCP   (1)                       PRINT EDITED RECORD
             WAIT   (1)                       WAIT FOR I/O COMPLETION
             LA     10,CHA12                  GET ADDRESS FOR CHAN 12 ROUTINE
             BAL    14,CHECK                  GO TO UNIT EXCEPTION SUBROUTINE
             B      READTAPE
CHECK        TM     4(1),1                    CHECK FOR UNIT EXEC. IN CCB
             BCR    1,10                       YES-GO TO PROPER ROUTINE
             BR     14                         NO-RETURN TO MAINLINE
CHA12        MVI    PRINTCCW,SKIPTO1          SET SEEK TO CHAN 1 OP CODE
             EXCP   (1)                       SEEK TO CHAN 1 IMMEDIATELY
             WAIT   (1)                       WAIT FOR I/O COMPLETION
             MVI    PRINTCCW,PRINT            SET PRINTER OP CODE TO WRITE
             BR     14                        RETURN TO MAINLINE
EOFTAPE      EOJ                              END OF JOB
             CNOP   0,4                       ALIGN CCB'S TO FULL WORD
PRINTCCB     CCB    SYS004,PRINTCCW,X'0400'
TAPECCB      CCB    SYS001,TAPECCW
PRINTCCW     CCW    PRINT,OUTAREA,SLI,L'OUTAREA
TAPECCW      CCW    READ,INAREA,SLI,L'INAREA
OUTAREA      DC     CL110' '
INAREA       DC     CL100' '
SLI          EQU    X'20'
READ         EQU    2
PRINT        EQU    9
SKIPTO1      EQU    X'8B'
             END    PROGRAM
```

Figure D-2. Self-relocating sample program.

Appendix E: American National Standard Code for Information Interchange (ASCII)

In addition to the EBCDIC mode, VSE accepts magnetic tape files written in ASCII, a 128-character 7-bit code. The high-order bit in this 8-bit environment is zero. ASCII is based on the specifications of the American National Standards Institute, Inc.

VSE processes ASCII files in EBCDIC with the help of two translate tables, which are loaded into the SVA. Using these tables, logical IOCS translates from ASCII to EBCDIC all data as it is read into the I/O area. For ASCII output, logical IOCS translates data from EBCDIC to ASCII just before writing the record.

Figure E-1 shows the relative bit positions of the ASCII character set. An ASCII character is described by its column/row position in the table. The columns across the top of Figure E-1 list the three high-order bits. The rows along the left side of Figure E-1 are the four low-order bits.

For example, the letter P in ASCII is under column 5 and row 0 and is described in ASCII notation as 5/0. ASCII 5/0 and EBCDIC X'50' represent the same binary configuration (B'01010000'). However, P graphically represents this configuration in ASCII and & in EBCDIC. ASCII notation is always expressed in decimal. For example, the ASCII Z is expressed as 5/10 (not 5/A).

For those EBCDIC characters that have no direct equivalent in ASCII, the substitute character (SUB) is provided during translation. See Figure E-2 for ASCII to EBCDIC correspondence.

Note: If an EBCDIC file is translated into ASCII, and you translate back into EBCDIC, this substitute character may not receive the expected value.

b_7 b_6 b_5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits ③ b_4	b_3	b_2	b_1	Column → / Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	! ①	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	-		M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	∧ ②	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

① The graphic | (Logical OR) may also be used instead of ! (Exclamation Point).

② The graphic ¬ (Logical NOT) may also be used instead of ∧ (Circumflex).

③ The 7 bit ASCII code expands to 8 bits when in storage by adding a high order 0 bit.

Example: Pound sign (#) is represented by

b_7	b_6	b_5	b_4	b_3	b_2	b_1
0	1	0	0	0	1	1

(The page prints "0 0 1 0 0 0 1 1" — 8 bits, the leading 0 being the added high order bit.)

Control Character Representations

NUL	Null	DLE	Data Link Escape (CC)
SOH	Start of Heading (CC)	DC1	Device Control 1
STX	Start of Text (CC)	DC2	Device Control 2
ETX	End of Text (CC)	DC3	Device Control 3
EOT	End of Transmission (CC)	DC4	Device Control 4
ENQ	Enquiry (CC)	NAK	Negative Acknowledge (CC)
ACK	Acknowledge (CC)	SYN	Synchronous Idle (CC)
BEL	Bell	ETB	End of Transmission Block (CC)
BS	Backspace (FE)	CAN	Cancel
HT	Horizontal Tabulation (FE)	EM	End of Medium
LF	Line Feed (FE)	SUB	Substitute
VT	Vertical Tabulation (FE)	ESC	Escape
FF	Form Feed (FE)	FS	File Separator (IS)
CR	Carriage Return (FE)	GS	Group Separator (IS)
SO	Shift Out	RS	Record Separator (IS)
SI	Shift In	US	Unit Separator (IS)
		DEL	Delete

(CC) Communication Control
(FE) Format Effector
(IS) Information Separator

Special Graphic Characters

SP	Space	<	Less Than
!	Exclamation Point	=	Equals
|	Logical OR	>	Greater Than
"	Quotation Marks	?	Question Mark
#	Number Sign	@	Commercial At
$	Dollar Sign	[	Opening Bracket
%	Percent	\	Reverse Slant
&	Ampersand	]	Closing Bracket
'	Apostrophe	∧	Circumflex
(	Opening Parenthesis	¬	Logical NOT
)	Closing Parenthesis	_	Underline
*	Asterisk	`	Grave Accent
+	Plus	{	Opening Brace
,	Comma	¦	Vertical Line (This graphic is stylized to distinguish it from Logical OR)
-	Hyphen (Minus)	}	Closing Brace
.	Period (Decimal Point)	~	Tilde
/	Slant		
:	Colon		
;	Semicolon		

Figure E-1. ASCII character set.

ASCII					EBCDIC				
Character	Col	Row	Bit Pattern		Col (in Hex)	Row (in Hex)	Bit Pattern		Comments
NUL	0	0	0000	0000	0	0	0000	0000	
SOH	0	1	0000	0001	0	1	0000	0001	
STX	0	2	0000	0010	0	2	0000	0010	
ETX	0	3	0000	0011	0	3	0000	0011	
EOT	0	4	0000	0100	3	7	0011	0111	
ENQ	0	5	0000	0101	2	D	0010	1101	
ACK	0	6	0000	0110	2	E	0010	1110	
BEL	0	7	0000	0111	2	F	0010	1111	
BS	0	8	0000	1000	1	6	0001	0110	
HT	0	9	0000	1001	0	5	0000	0101	
LF	0	10	0000	1010	2	5	0010	0101	
VT	0	11	0000	1011	0	B	0000	1011	
FF	0	12	0000	1100	0	C	0000	1100	
CR	0	13	0000	1101	0	D	0000	1101	
SO	0	14	0000	1110	0	E	0000	1110	
SI	0	15	0000	1111	0	F	0000	1111	
DLE	1	0	0001	0000	1	0	0001	0000	
DC1	1	1	0001	0001	1	1	0001	0001	
DC2	1	2	0001	0010	1	2	0001	0010	
DC3	1	3	0001	0011	1	3	0001	0011	
DC4	1	4	0001	0100	3	C	0011	1100	
NAK	1	5	0001	0101	3	D	0011	1101	
SYN	1	6	0001	0110	3	2	0011	0010	
ETB	1	7	0001	0111	2	6	0010	0110	
CAN	1	8	0001	1000	1	8	0001	1000	
EM	1	9	0001	1001	1	9	0001	1001	
SUB	1	10	0001	1010	3	F	0011	1111	
ESC	1	11	0001	1011	2	7	0010	0111	
FS	1	12	0001	1100	1	C	0001	1100	
GS	1	13	0001	1101	1	D	0001	1101	
RS	1	14	0001	1110	1	E	0001	1110	
US	1	15	0001	1111	1	F	0001	1111	
SP	2	0	0010	0000	4	0	0100	0000	
| ①	2	1	0010	0001	4	F	0100	1111	Logical OR
"	2	2	0010	0010	7	F	0111	1111	
#	2	3	0010	0011	7	B	0111	1011	
$	2	4	0010	0100	5	B	0101	1011	
%	2	5	0010	0101	6	C	0110	1100	
&	2	6	0010	0110	5	0	0101	0000	
'	2	7	0010	0111	7	D	0111	1101	
(	2	8	0010	1000	4	D	0100	1101	
)	2	9	0010	1001	5	D	0101	1101	
*	2	10	0010	1010	5	C	0101	1100	
+	2	11	0010	1011	4	E	0100	1110	
,	2	12	0010	1100	6	B	0110	1011	
-	2	13	0010	1101	6	0	0110	0000	Hyphen, Minus
.	2	14	0010	1110	4	B	0100	1011	
/	2	15	0010	1111	6	1	0110	0001	
0	3	0	0011	0000	F	0	1111	0000	
1	3	1	0011	0001	F	1	1111	0001	
2	3	2	0011	0010	F	2	1111	0010	
3	3	3	0011	0011	F	3	1111	0011	
4	3	4	0011	0100	F	4	1111	0100	
5	3	5	0011	0101	F	5	1111	0101	
6	3	6	0011	0110	F	6	1111	0110	
7	3	7	0011	0111	F	7	1111	0111	
8	3	8	0011	1000	F	8	1111	1000	
9	3	9	0011	1001	F	9	1111	1001	
:	3	10	0011	1010	7	A	0111	1010	
;	3	11	0011	1011	5	E	0101	1110	
<	3	12	0011	1100	4	C	0100	1100	
=	3	13	0011	1101	7	E	0111	1110	
>	3	14	0011	1110	6	E	0110	1110	
?	3	15	0011	1111	6	F	0110	1111	

Figure E-2. ASCII to EBCDIC correspondence (Part 1 of 2).

ASCII					EBCDIC				
Character	Col	Row	Bit Pattern		Col (in Hex)	Row (in Hex)	Bit Pattern		Comments
@	4	0	0100	0000	7	C	0111	1100	
A	4	1	0100	0001	C	1	1100	0001	
B	4	2	0100	0010	C	2	1100	0010	
C	4	3	0100	0011	C	3	1100	0011	
D	4	4	0100	0100	C	4	1100	0100	
E	4	5	0100	0101	C	5	1100	0101	
F	4	6	0100	0110	C	6	1100	0110	
G	4	7	0100	0111	C	7	1100	0111	
H	4	8	0100	1000	C	8	1100	1000	
I	4	9	0100	1001	C	9	1100	1001	
J	4	10	0100	1010	D	1	1101	0001	
K	4	11	0100	1011	D	2	1101	0010	
L	4	12	0100	1100	D	3	1101	0011	
M	4	13	0100	1101	D	4	1101	0100	
N	4	14	0100	1110	D	5	1101	0101	
O	4	15	0100	1111	D	6	1101	0110	
P	5	0	0101	0000	D	7	1101	0111	
Q	5	1	0101	0001	D	8	1101	1000	
R	5	2	0101	0010	D	9	1101	1001	
S	5	3	0101	0011	E	2	1110	0010	
T	5	4	0101	0100	E	3	1110	0011	
U	5	5	0101	0101	E	4	1110	0100	
V	5	6	0101	0110	E	5	1110	0101	
W	5	7	0101	0111	E	6	1110	0110	
X	5	8	0101	1000	E	7	1110	0111	
Y	5	9	0101	1001	E	8	1110	1000	
Z	5	10	0101	1010	E	9	1110	1001	
[	5	11	0101	1011	4	A	0100	1010	
\	5	12	0101	1100	E	0	1110	0000	Reverse Slant
]	5	13	0101	1101	5	A	0101	1010	
¬ ②	5	14	0101	1110	5	F	0101	1111	Logical NOT
_	5	15	0101	1111	6	D	0110	1101	Underscore
`	6	0	0110	0000	7	9	0111	1001	Grave Accent
a	6	1	0110	0001	8	1	1000	0001	
b	6	2	0110	0010	8	2	1000	0010	
c	6	3	0110	0011	8	3	1000	0011	
d	6	4	0110	0100	8	4	1000	0100	
e	6	5	0110	0101	8	5	1000	0101	
f	6	6	0110	0110	8	6	1000	0110	
g	6	7	0110	0111	8	7	1000	0111	
h	6	8	0110	1000	8	8	1000	1000	
i	6	9	0110	1001	8	9	1000	1001	
j	6	10	0110	1010	9	1	1001	0001	
k	6	11	0110	1011	9	2	1001	0010	
l	6	12	0110	1100	9	3	1001	0011	
m	6	13	0110	1101	9	4	1001	0100	
n	6	14	0110	1110	9	5	1001	0101	
o	6	15	0110	1111	9	6	1001	0110	
p	7	0	0111	0000	9	7	1001	0111	
q	7	1	0111	0001	9	8	1001	1000	
r	7	2	0111	0010	9	9	1001	1001	
s	7	3	0111	0011	A	2	1010	0010	
t	7	4	0111	0100	A	3	1010	0011	
u	7	5	0111	0101	A	4	1010	0100	
v	7	6	0111	0110	A	5	1010	0101	
w	7	7	0111	0111	A	6	1010	0110	
x	7	8	0111	1000	A	7	1010	0111	
y	7	9	0111	1001	A	8	1010	1000	
z	7	10	0111	1010	A	9	1010	1001	
{	7	11	0111	1011	C	0	1100	0000	
¦	7	12	0111	1100	6	A	0110	1010	Vertical Line
}	7	13	0111	1101	D	0	1101	0000	
~	7	14	0111	1110	A	1	1010	0001	Tilde
DEL	7	15	0111	1111	0	7	0000	0111	

① The graphic ! (Exclamation Point) can be used instead of | (Logical OR).

② The graphic ^ (Circumflex) can be used instead of ¬ (Logical NOT).

Figure E-2. ASCII to EBCDIC correspondence (Part 2 of 2).

Appendix F: Page Fault Handling Overlap

For some special types of processing, users may choose to write programs which, while executing as one VSE user task, provide for the asynchronous processing of several tasks. This multitasking, which is not to be confused with VSE multitasking, is generally used only in very sophisticated applications where no other technique would give acceptable performance.

This type of asynchronous processing (called 'private multitasking') is not supported by IBM and therefore is not documented as such. VSE does, however, provide one tool which aids the programmer doing private multitasking. This is the ability to overlap the handling of a page fault in one private subtask with the processing of another private subtask.

The user may set up an appendage routine which is to be entered whenever his VSE task causes a page fault. This appendage routine sets the conditions for dispatching a private subtask originating in the same VSE task. The routine is called by the VSE supervisor whenever the VSE task causes a page fault and whenever a page fault has just been handled for that task.

The VSE task is not put into the wait state when it causes a page fault as is usually the case, but remains dispatchable. The linkage to the appendage is established by issuing a SETPFA macro.

The appendage routine must not cause a page fault and, therefore, must be fixed in real storage using the PFIX macro before the SETPFA macro is issued. Also, the appendage routine must not issue an SVC instruction. The routine is given control in the supervisor state, with I/O interrupts disabled, and with protection key zero. Following is a description of the conventions observed by page fault appendages in VSE as well as suggestions on how an appendage should be set up.

Register Usage

The following registers are used to pass information between the supervisor and the page fault appendage:

- Register 7 contains the return address to the supervisor.
- Register 8 contains the address of the appendage routine and can, therefore, be used as the base register.
- Register 13 contains a parameter with information about the page fault to be handled. The information in register 13 has the following format:

reserved	address of page	reserved
0	1 2	3

Bytes 1-2: leftmost 16 bits of the address of the page (on page boundary) which has to be handled.

- The contents of the remaining registers (0 to 6, 9 to 12, and 14 to 15) are undefined. However, the contents of all registers are saved by the supervisor before it transfers control to the appendage routine.

Entry Linkage

The entry coding for the appendage should be as follows:

```
label  USING *,8   use reg.8 as base
                   register
       B ENTRYA    used after page
                   fault
       B ENTRYB    used after page
                   fault completion
```

The name specified for *label* is the entry point of the routine specified in the SETPFA macro. The branch to ENTRYA is taken whenever the task causes a page fault. The branch to ENTRYB is taken whenever a page fault for the task has been handled.

Page Fault Queue

The appendage must have a queue with a four-byte entry for each private subtask controlled by that VSE task. This queue is used to store the page fault information passed in register 13.

Processing at the Initiation of a Page Fault

When the routine is entered at ENTRYA, register 13 contains information on the page fault which has just occurred. The appendage must store the contents of register 13 in an internal page fault queue, put the task causing the page fault into an internal wait state, and, if possible, have a private subtask dispatched.

To be able to always dispatch a private subtask, a special subtask must exist that issues a WAIT or a WAITM with an unposted ECB. This special subtask will be dispatched if no "normal" private subtask is dispatchable.

To have a private subtask dispatched, the appendage routine must store, in bytes 8 to 15 of the

task's save area, an EC-mode PSW that contains the address of the subtask instruction which is to be executed first. Normally, this is accomplished by setting up this PSW (along with any values that should be loaded into specific registers for the subtask's execution) in the save area of the particular subtask and exchanging this save area's contents with the contents of the save area for the task that issued the SETPFA macro.

The routine must then return control to the supervisor.

Processing at the Completion of a Page Fault

Whenever a page fault request has been handled for the VSE task, control is passed to ENTRYB. The appendage routine should dequeue the request which has just been handled from the internal page fault queue. If there are any more page fault requests in the queue, the information for the next request to be handled must be returned in register 13; otherwise, register 13 must contain zero.

Each time control is passed to ENTRYB, the traffic bit (bit 0, byte 2) in the ECB used by the special subtask should be set on. Since the page manager will remove the VSE task from the WAIT state if it was put there because of the WAIT or WAITM issued by the special subtask, the special subtask will get control and can then resume internal dispatching.

Figure F-1 is an example of how page faults are handled when the task causing the page faults has set up a page fault appendage. For the layout of a task's (or subtask's) save area, see the section "Save Areas" in *VSE/Advanced Functions Serviceability Aids and Debugging Procedures.*

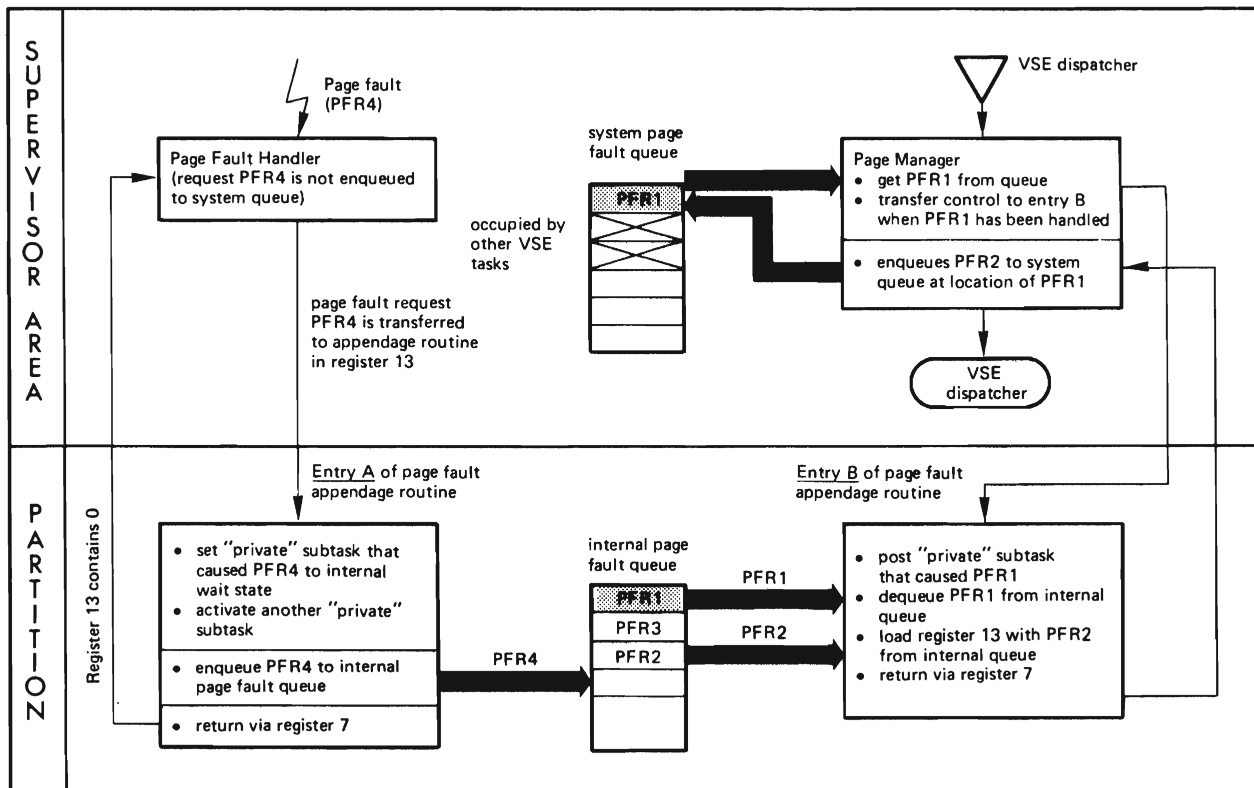


Figure F-1. Page fault handling overlap.

Appendix G: Using System Control Macros in Reenterable Programs

A reenterable program can be entered and used concurrently by several tasks without sacrificing the integrity of its instructions or data areas.

Data areas that may be modified by a reenterable program must be unique to each task executing that program. Examples of such areas are: save areas, I/O areas, control blocks. Consider a program containing the ATTACH macro:

```
                                                    column 72
                                                       ↓
          ATTACH SUBTASK,                              X
                 SAVE=LOCSAV,                          X
                 ECB=LOCECB,                           X
                 ABSAVE=LOCSAVAB
          •
          •
          •
          WAIT   LOCECB
          •
          •
          •
LOCSAV    DS     9D
LOCSAVAB  DS     9D
LOCECB    DC     F'0'
          •
          •
```

A task that executes the above ATTACH macro initializes the save area (LOCSAV) for the subtask to be attached. After the attached subtask has started processing, it can be interrupted. While the subtask remains in the wait state, another task may be dispatched and execute the macro. Because only one subtask save area exists, this task, in initializing the save area, would destroy whatever was saved there when the interrupt occurred. As coded above, the ATTACH macro is not reenterable because data areas are not unique to the tasks executing the macro.

A commonly used method of isolating data areas for individual tasks is to establish them outside the program's boundaries. Through the GETVIS macro, a task may dynamically acquire storage which it will use as a data area; this data area may be kept unique to the task that acquires it.

Fields in the dynamic storage area are addressable through registers. Figure G-1 shows how to acquire and address dynamic storage and how to refer to the individual fields in the ATTACH macro:

```
DYNSTOR   DSECT  ,            DYNAMIC STORAGE AREA
DYNPARM   DS     CL64         REENTERABLE MACRO PARM AREA
DYNSAV    DS     CL72         SUBTASK SAVE AREA
DYNSAVAB  DS     CL72         SUBTASK AB-EXIT SAVE AREA
DYNECB    DS     F            SUBTASK ECB
DYNSTORL  EQU    *-DYNSTOR    LENGTH OF DYN STORAGE
          •
          •
          •
          LA     0,DYNSTORL   LENGTH FOR GETVIS
          GETVIS ADDRESS=(10)
          •
          •
          USING  DYNSTOR,10    MAKE DYNAMIC STORAGE
*                       ADDRESSABLE THRU BASE REG.10
          LA     7,DYNSAV
          LA     8,DYNECB
          LA     9,DYNSAVAB
          ATTACH SUBTASK,SAVE=(7),ECB=(8),ABSAVE=(9)
          •
          •
          •
```

Figure G-1. Acquiring and addressing dynamic storage.

Writing the ATTACH macro as shown in Figure G-1 generates the following code (macro expansion):

```
     CNOP     2,4
A    ST       8,14+4+*
B    ST       9,10+8+*
     LR       0,7
     BAL      1,*+16
     DC       A(SUBTASK)
     DC       A(0)
     DC       A(0)
     SVC      38          ATTACH SUBTASK
```

Data areas (a parameter list) exist inside the macro expansion. They are common to all tasks concurrently executing this ATTACH macro and are modified via statements at **A** and **B**. In the context of reentrancy, they belong in the same category as save areas, I/O areas, or control blocks. A program containing a macro as written above is not reenterable.

For system control macros with the MFG operand (such as ATTACH, FETCH, GETIME, etc.) VSE builds a parameter list outside the macro expansion. The MFG operand points to this list. The list is a 64-byte area which the program provides for the macro's execution.

To make the program reenterable the parameter list must be unique to any task executing the macro. Again, a convenient method of establishing that uniqueness is to make the parameter list part of a dynamically obtained storage area. The ATTACH macro would then look as follows:

```
ATTACH SUBTASK,                               X
       SAVE=(7),ECB=(8),ABSAVE=(9),           X
       MFG=(10)
```

Note: MFG stands for "Macro Format: Generate", bearing some resemblance to the MF=(G,...) parameter in VSAM macros.

The macro expansion shows that the parameter list is no longer stored into the generated code; instead, the parameter list is only referenced (using register 10 in this example):

```
L      0,=A(SUBTASK)
ST     0,0+0(10)
ST     8,0+4+0(10)
ST     9,0+8+0(10)
LR     0,7
LA     1,0(10)
SVC    38
```

Register notation, as used in the preceding example, may be costly because each operand uses up a register and, in addition, each register has to be preloaded with the address of the pertinent field.

Where indicated in the macro format (refer to *VSE/Advanced Functions Macro Reference*), the operand may be specified in (S,address) notation instead of register notation. The ATTACH macro is one of the macros that allows this alternative:

```
ATTACH SUBTASK,                               X
       SAVE=(S,DYNSAV),                       X
       ECB=(S,DYNECB),                        X
       ABSAVE=(S,DYNSAVAB),                   X
       MFG=(S,DYNPARM)
```

An operand written in (S,address) notation assembles like an S-type address constant: its object code is an assembler instruction address in base register/displacement form. Example: for the above ATTACH macro,

```
DYNSAV   assembles into   X'A040',
DYNECB             into   X'A0D0'.
```

Observe that both addresses contain the same register (10). With (S,address) notation, only one register is used, the one that serves as the base register for the DSECT.

Index

A-type address constants D-3
abnormal end (see STXIT macro)
abnormal end
    of task, subtask 10-12
abnormal termination (see STXIT macro)
abnormal termination user exit 10-8
access control
    tape label processing C-7
Access Methods 3-1
activating (initializing) a DAM file 4-9
activating (opening) a DASD file 4-3
activating a SAM file 3-3, 4-3
activating (initializing) an ISAM file 3-26, 4-24
adding new records
    DAM 4-16
    ISAM 4-29, 4-27
adding overflow areas to an ISAM file 3-28
adding records to a DAM file 3-21
adding records to an ISAM file 3-28, 3-29, 4-27, 4-29
address constants in self-relocating programs D-3
advanced page-in 10-4
advanced printer buffering (3800) 8-1
advantages of self-relocating programs D-2
algorithm
    choice of 2-3
    DAM transformation 2-1, 2-2
Alternate Tape Switching 9-12
appendage routine for page faults F-1
ASA option for control character codes A-1
ASCII E-1
    character set E-2
    files C-4, E-1
assembling a format record
    3886 7-32, 7-34
assembling DTFs and logic modules 1-6
assembling your program, DTFs, and logic modules B-1
assembly examples B-2 - B-19
Assigning and Releasing I/O Units 10-5
Assigning and Releasing Tape Drives 10-6
ASSIGN macro 10-6
associated files
    card 7-1
    GET/CNTRL/PUT Sequence 7-10
    printer 7-10
    processing considerations for card files 7-3
    Read-Punch-Print 7-3
asynchronous processing F-1
attaching a subtask 10-23
ATTACH macro 10-21, 10-22
    in reenterable programs G-1

backward reading of tapes C-6
balancing teleprocessing 10-4
block size, magnetic tape file 6-1
block
    FBA 3-2
    logical 3-2
    physical 3-2
blocked record
    processing (SAM) 3-8
    selective processing 3-10
blocked records
    ISAM 3-25
    SAM 3-6, 3-7
blocking diskette files 5-2
branching between phases 10-16
buffer
    CI (control interval) 3-2
    forms control (FCB) 10-33

CALL macro 10-16, 10-20
called program 10-17
calling program 10-17
CANCEL macro 10-12
capacity record, DAM 3-24
Card Device Control 7-7
card files
    end-of-file handling 7-5
    error handling 7-6
card-to-disk operations B-1
CATALR card B-5
CCB (Command Control Block) 9-3, 9-5
CCB
    conditions indicated 9-6
    format 9-5
    macro 9-3
CDLOAD macro 10-1
CDMOD macro 7-1
changing processing priority 10-22
channel program example 9-12
channel programs
    CKD DASD, with PIOCS 9-11
    diskette, with PIOCS 9-12
    FBA devices 9-11
CHAP macro 10-22
character set, ASCII E-2
CHECK macro
    MICR 7-18, 7-19
    work file 3-14, 3-15
checking DASD extents (PIOCS) 9-8
checking nonstandard labels 9-8, C-6
checking standard labels C-2, C-6
checking user standard tape labels 6-1, 9-8
checkpoint records
    bypassing, with PIOCS 9-13
checkpoint
    choosing a 10-12
    files 10-14
    information not saved 10-14
    information saved 10-14
    on CKD DASD 10-15
    on FBA DASD 10-15
    on tape 10-15
    operator verification table 10-16
    repositioning files 10-15
    restarting 10-14
    saving data for restart 10-14
    timing 10-12
Checkpointing a Program 10-12
CHKPT macro 10-12
CI 4-3, 3-1
    buffer 3-2
    logical blocks 4-3
CISIZE
    DLBL parameter 4-3
CKD architecture 3-2
CKD DASD 3-2
CKD DASD Channel Programs, with PIOCS 9-11
Clear Disk utility program 3-21, 4-11
clearing a track, with DAM 4-17
CLOSE macro
    DAM 4-22
    ISAM 4-33
    PIOCS 9-9
    SAM 3-17
    SAM, on DASD 4-8
CLOSER macro
    DAM 4-22
    ISAM 4-34
    SAM 3-17
    self-relocating programs D-3

closing a file
DAM 4-22
diskette 5-3
ISAM 4-33
PIOCS 9-9
SAM 3-17, 4-8
CNTRL macro
card devices 7-7, 7-8
DAM 4-19
magnetic tape 6-6
printer 7-10, 7-11, 7-12, 7-14
SAM 3-17
1287/1288 7-24, 7-25, 7-26
3881 7-27
3886 7-26
COCR (Cylinder Overflow Control Record) 2-10
code translation on input (paper tape) 7-38, 7-39
codes, control A-1
combined card file processing example 3-8
combined files (SAM) 3-12
combining sequential and direct retrieval (ISAM) 3-30
command chaining
diskette 5-2
FBA device 9-12
PIOCS 9-10
retry bit 9-10
Command Control Block (CCB) 9-5
Command Control Block format 9-5
communication between job steps 10-4, 10-5
communication between tasks 10-28
communication region 10-4, 10-5
COMRG macro 10-5
console buffering
PIOCS 9-12
console files
processing 7-14
programming considerations 7-15
control character codes A-1
control characters A-1
control functions 10-1
Control Interval (see also CI)
control interval
buffer 3-2
format 3-1
hold function 10-29
Control Program Function Macros 1-1
count area
DAM 3-21
creating a file
DAM 3-21
ISAM 3-27
CTLCHR operand A-1
cylinder address (DAM) 4-11
cylinder index 2-4, 2-7
cylinder index area (ISAM) 4-27
cylinder overflow area 2-2
Cylinder Overflow Control Record (COCR) 2-10

DAM (Direct Access Method) 3-18
DAM
adding new records 4-16
adding records to a file 3-21
capacity record 3-24
closing a file 4-22
count area 3-21
creating a file 3-21
data area 3-20, 3-21
error handling 4-19
error status bits 4-20, 4-21
I/O area format 3-19, 3-20
I/O Area Specification 3-19
identifier (ID) reference 3-24
initialization 4-9
key reference 3-23
locating data 3-21
Locating Free Space 3-24
logic modules 3-24
non-data-device command 4-19
opening a file 4-9
overflow organization 4-15
overwriting existing records 4-17
processing 4-8, 4-10
reading blocks of data 4-14
record reference by AFTER 4-18
record reference by ID 4-16, 4-18
record reference by Key 4-16, 4-17
record reference by RZERO 4-18
record types 3-19
record zero (R0) 3-21, 3-22
reference methods 3-21
spanned record 3-19
writing blocks of data 4-16
writing over existing records 4-17
DAMOD 4-8
DAMODV 4-8
DASD Record Protection (Track Hold) 10-29
DASD standard labels C-1
DASD
capacities 4-1, 4-2
checkpoint considerations 10-14
checkpoints on disk 10-15
error handling by DAM 4-19
error handling by ISAM 4-26
error handling by SAM 4-6
label processing by SAM 4-5
processing with SAM 4-2
record capacities 4-1
track capacities 4-2
data area 2-4
DAM 3-21
ISAM 2-4
data areas 2-2
direct organization 2-2
prime 2-4
data base 2-1
Data Chaining
PIOCS 9-11, 9-12
data file processing (with SAM) 3-4
data organization 2-1
direct 2-1
sequential with index 2-3
sequential without index 2-1
data records, reading 3886 7-31
data transfer
FBA device 3-2
diskette 5-2
deactivating a DAM file 4-22
deactivating a sequential DASD file 4-8
deactivating a PIOCS file 9-9
deactivating a SAM file 3-16, 4-8
deactivating an ISAM file 3-30, 4-34
Declarative IOCS Macros 1-1
Declarative Macro Statements 1-10
declarative macros (SAM) 3-18
Define The Lock (DTL) macro 10-27
defining DAM files 4-8
defining ISAM files 4-22
defining SAM files 3-2
deleting records (ISAM) 3-30
DEQ macro 10-21, 10-24 - 10-29
dequeueing a resource 10-24
describing SAM files 3-2
DETACH macro 10-12, 10-24
detaching a subtask 10-24
device independence 8-1
device-independent system files 8-1
end-of-file handling 8-3
error handling 8-2

processing 8-1
record size 8-1
Device Support Facility 4-10
DFR macro operands 7-23
DFR macro, with 3886 7-31
DIMOD macro 8-1
direct (random) processing, ISAM 3-30
direct (random) retrieval, ISAM 3-30
Direct Access Method (DAM) 3-18
direct linkage (between routines) 10-18
direct organization 2-1
DISEN macro
MICR 7-19
with 1270/1275 7-27
disk utility programs 4-10, 4-11
diskette files 5-1
blocking 5-2
command chaining 5-2
processing 5-1
with PIOCS 5-1
diskette 5-1
channel programs 9-11, 9-12
checkpoint considerations 10-14
data transfer 5-2
error handling 5-3
labels C-3
standard labels 9-3
DLINT macro operands 7-23
DLINT macro
with 3886 7-31
document control on 3886 7-31
document marking on 3886 7-31
DSPLY macro
with 1287/1288 7-29
DTF table 1-2
DTFCD macro 7-1, 7-2
DTFCN macro operands 7-15
DTFDA files
hold function 10-29
module name 1-5
DTFDA macro operands 4-9
DTFDA 4-8
DTFDI macro operands 8-2
DTFDI
macro 8-1
module name 1-5
restrictions 8-1
DTFDR macro operands 7-23
DTFDR macro
with 3886 7-31
DTFDU error options 5-4
DTFDU macro 5-1
DTFIS files
hold function 10-29
DTFIS macro operands 4-23
DTFMR macro operands 7-16
DTFMR macro
with 1270/1275 7-27
DTFMT error options 6-6
DTFMT macro operands 6-2
DTFMT macro sample 1-3
DTFOR macro operands 7-24
DTFPH macro 9-1
DTFPH macro operands 9-2
DTFPR macro 7-10
DTFPR macro operands 7-12
DTFPT macro operands 7-36
DTFs, assembling 1-6, B-1
DTFSD data files, hold function 10-29
DTFSD macro 4-2
DTFSD macro operands 4-4
DTFSD work files
hold function 10-29
DTFSD
error handling 4-6
module name 1-5
DTL macro 10-27
DUMP macro 10-10, 10-11
dumps, requesting storage 10-10
Dynamic Allocation of Virtual Storage 10-4
dynamic storage area G-1

EBCDIC to ASCII correspondence E-2
ECB (event control block) 10-21
specifying, for multitasking 10-21
end-of-file handling
card files 7-5
DASD standard labels C-2
device-independent system files 8-3
diskette labels C-3
end-of-volume processing
DASD standard labels C-1
diskette labels C-3
ENDFL macro 4-29, 4-27
ending a
job 10-11
subtask 10-12
task 10-11
ENQ macro 10-21, 10-24 - 10-25
enqueueing a resource 10-24
entry linkage for page fault appendage F-1
EOJ macro 10-11
ERET macro 4-6, 4-26, 6-4, 8-3
error handling
card files 7-6
DAM 4-19
device-independent system files 8-2
diskette 5-3
DTFMT 6-4
ISAM 4-26
magnetic tape files 6-4
paper tape 7-39
printer files 7-14
SAM, on DASD 4-6
3886 7-32
error options
DTFDU 5-4
DTFMT 6-4, 6-6
error processing routines
magnetic tape 6-5
sequential DASD 4-6
error status bits (for ERRBYTE on DAM) 4-20, 4-21
ESETL macro 4-29, 4-32
ESETL macro
issuing a hold 10-31
event control block (see ECB)
example
assembling DTFs and logic modules B-1 - B-19
attaching a subtask 10-23
channel program 9-12
checkpointing 10-9, 10-13
combined card file processing 3-8
DAM overflow organization 4-15
detaching a subtask 10-24
interval timer 10-9
ISAM file 2-8
multitask linkage 10-10
OMR coding 7-5
PFIX and PFREE 10-3
POST macro 10-30
program check user exit 10-11
restarting checkpointed program 10-13
sharing a resource 10-26
supersetting/subsetting 1-5
track hold 10-32
UNLOCK macro 10-28
EXCP macro 9-4
execution mode

determining the 10-3
EXIT macro 10-8, 10-12
extending an ISAM file 3-28, 4-27
extent checking (on DASD) 4-3
extent checking, with PIOCS 9-8

FBA (Fixed Block Architecture) 3-1
FBA DASD 3-2
FBA
blocks 3-2, 4-3
channel programs 9-11
DASD Processing 4-3
data transfer 3-2
FCB (Forms Control Buffer), loading an 10-33
FCB 10-11, 10-33
FCEPGOUT macro 10-3
FEOV macro 3-16, 6-1
PIOCS 9-9, 9-12
FEOVD macro 3-16
SAM, on DASD 4-8
FETCH macro 10-1
file management system (ISAM) 3-24
file organization 2-1
direct 2-1
sequential, with index 2-3
sequential, without index 2-1
file reorganization (ISAM) 3-25
file-protected DASD files 9-10
FilenameC (ISAM) 4-28
Fixed Block Architecture (see also FBA) 3-1
fixing pages in real storage 10-2
forcing end-of-volume 3-16
magnetic tape 6-1
PIOCS 9-9, 9-12
SAM 3-11
SAM on DASD 4-8
forcing page-in 10-4
forcing page-out 10-3
format record
assembling (3886) 7-32, 7-34
format, macro 1-8
formatting write (WRITE SQ) 3-14
forms control buffer (FCB), loading a 10-33
forms control buffer (see FCB)
FREE macro 10-31
freeing pages in real storage 10-2
FREEVIS macro 10-4

generate macro format G-2
generation of system pack B-12
GENDTL macro 10-27
GENIORB macro 9-3
GENL macro 10-1
GET macro
ISAM 4-32
MICR 7-18, 7-19
SAM 3-5
with 1270/1275 7-27
GETIME macro 10-6
GETVIS area 10-1, 10-4
GETVIS macro 10-4
GETVIS macro
in reenterable programs G-1

hashing 2-2
header labels (see tape labels)
hold function 10-29

I/O area format
DAM 3-19, 3-20
I/O area requirements (ISAM) 4-26
I/O area specification, ISAM 4-22
I/O Area Specification
DAM 3-19
SAM 3-3
I/O areas
ISAM 4-25
paper tape 7-39
I/O Request Block (IORB) 9-3
I/O units
assigning/releasing 10-5
identifier (ID) reference with DAM 3-24
Imperative IOCS Macros 1-1
independent overflow area 2-2
index area 2-4
index structure for ISAM file 2-9
Indexed Sequential Access Method (ISAM) 3-24
indexes, ISAM 2-4
influencing the paging mechanism 10-3
initialization with DAM 4-9
initialization
ISAM 4-24
initializing disk (Device Support Facility) 4-10
initiation, subtask 10-21
interrelationship of macros 1-6
Intertask Communication 10-28
interval timer (see STXIT macro)
interval timer 10-6
interrupt 10-7
setting 10-6
testing 10-7
unexpired time 10-7
user exit 10-8
IOCS logic modules
assembling B-1
IOCS Macros 1-1
IORB (I/O Request Block) 9-3
IORB macro 9-3
ISAM (Indexed Sequential Access Method) 3-24
ISAM
activating a file for processing 3-26
adding overflow areas to a file 3-28
adding records to a file 3-28, 3-29, 4-29
blocked records 3-25
closing a file 3-30, 4-33
creating a file 3-27
cylinder index 2-7, 2-8, 2-4
cylinder index area 4-27
data area 2-4
deleting records 3-30
disk storage space formulas 4-35
error handling 4-26
extending a file 3-28, 4-27
file example 2-8
file index structure 2-9
file reorganization 3-30
indexes 2-4
key area 2-4
key areas 3-26
loading a file 4-27
logic modules 3-31
logical records 3-26
master index area 4-27
opening a file 3-26, 4-24
prime data area 4-27
processing a file 3-26
processing files 4-26
programming considerations 4-34
record types 3-26
reorganizing a file 3-30, 3-32
restrictions 3-25
sequential processing 3-25
sequential retrieval 4-32
storage areas 3-26
ISMOD macro operands 4-24

JDUMP macro 10-11, 10-12
JOBCOM macro 10-5
job end 10-11
job steps, communication between 10-4, 10-5

key area 2-4
key areas
  ISAM 3-26
key reference
  DAM 3-23
keyword operands 1-9

LABADDR routines
  for DASD files C-1
  for tape input files C-5
  for tape output files C-3
label processing C-1
  magnetic tape 6-1
label processing
  SAM, on DASD 4-5
  with access control C-7
LBRET macro
  DAM 4-10
  magnetic tape 6-1
  PIOCS 9-4, 9-8
LFCB macro 10-33
line skipping on printer 7-11
line spacing on printer 7-11, 10-33
link-editing programs 1-6
linkage editor 10-17
linkage
  direct, (between routines) 10-18
  registers 10-19
LIOCS functions in PIOCS processing 9-10
LITE macro
  MICR 7-20
  with 1270/1275 7-27
LOAD macro 10-1
loading a phase 10-1
loading a program 10-1
loading an ISAM file 4-27
loading DAM files 4-10
Locating Data (DAM) 3-21
Locating Free Space (DAM) 3-24
lock control block 10-27
LOCK macro 10-27
LOCKOPT, resource sharing 10-27
lock request count 10-27
lock table size 10-27
Logic Module Generation Macros 1-4
logic modules, supplying the name for 1-5
logic modules
  assembling B-1, 1-6
  DAM 3-24
  ISAM 1-4, 3-31
  preassembled 1-4
  providing 1-4
  RPS 1-5
  SAM 3-17
  standard module names 1-5
logical block 3-2
logical blocks
  in FBA blocks 4-3
logical records
  ISAM 3-25

macro format 1-8
macro format: generate (MFG) G-2
macro processing 1-1
macro statements, declarative 1-10
macro types 1-1
macro
  ASSIGN 10-6
  ATTACH 10-22
  CALL 10-16, 10-20
  CANCEL 10-12
  CCB 9-3
  CDLOAD 10-1
  CDMOD 7-1
  CHAP 10-22
  CHECK 3-15, 7-19
  CHKPT 10-12
  CLOSE (see CLOSE)
  CLOSER (see CLOSER)
  CNTRL (magnetic tape) 6-6
  CNTRL (see CNTRL)
  COMRG 10-5
  DEQ 10-21, 10-24
  DETACH 10-12
  DFR 7-21, 7-23
  DIMOD 8-1
  DISEN 7-19, 7-27
  DLINT 7-21, 7-23
  DSPLY 7-29
  DTFCD 7-1, 7-2
  DTFCN 7-14, 7-15
  DTFDI 8-1
  DTFDR 7-21, 7-23
  DTFDU 5-1
  DTFIS 4-22, 4-23
  DTFMR 7-15, 7-16
  DTFOR 7-24
  DTFPH 9-1
  DTFPR 7-10, 7-12
  DTFPT 7-35, 7-36
  DTL 10-27
  DUMP 10-10, 10-11
  ENDFL 4-27, 4-29
  ENQ 10-21, 10-24
  EOJ 10-11
  ERET 4-6, 4-26, 6-5, 8-3
  ESETL macro 4-29, 4-32
  EXCP 9-4
  EXIT 10-8, 10-12
  FCEPGOUT 10-3
  FEOV 6-1
  FEOV (see FEOV)
  FETCH 10-1
  FREE 10-30
  FREEVIS 10-4
  GENDTL 10-27
  GENIORB 9-3
  GENL 10-1
  GET (see GET)
  GETIME 10-6
  GETVIS 10-4
  IORB 9-3
  ISMOD 4-24
  JDUMP 10-11
  JOBCOM 10-5
  LBRET 4-10, 6-1, 9-4, 9-8
  LFCB 10-33
  LITE 7-20, 7-27
  LOAD 10-1
  LOCK 10-27
  MODDTL 10-27
  MRMOD 7-15
  MVCOM 10-5
  NOTE 3-15
  OPEN (see OPEN)
  OPENR (see OPENR)
  PAGEIN 10-4
  PDUMP 10-11
  PFIX 10-2
  PFREE 10-2
  POINTR 3-15, 3-16
  POINTS 3-15, 3-16

POINTW 3-15, 3-17
POST 10-28, 10-29
PRMOD 7-10, 7-12
PRTOV 7-10
PTMOD 7-35, 7-36
PUT (see PUT)
PUTR 7-15
RCB 10-24
READ (see READ)
RELEASE 10-5
RELPAG 10-3
RELSE 3-11
RESCN 7-29
RETURN 10-16, 10-20, 10-21
RUNMODE 10-2
SAVE 10-16, 10-20
SECTVAL 9-4
SETDEV 10-14, 7-31
SETFL 4-27
SETIME 10-6
SETL macro 4-32
SETPFA F-1
SETT 10-7
STXIT 10-8, 10-10
TECB 10-7
TESTT 10-7
TPIN 10-4
TPOUT 10-4
TRUNC 3-11
TTIMER 10-7
UNLOCK 10-27, 10-28
WAIT 10-7, 10-28
WAIT (PIOCS) 9-4
WAITF (see WAITF)
WAITM 10-28
WRITE (see WRITE)
macros, interrelationship of 1-6
macros
control program function 1-1
declarative (DTF) 1-1, 1-10
declarative IOCS 1-1
imperative IOCS 1-1
IOCS 1-1
source-program 1-1
supervisor 1-1
VSAM 1-1
Magnetic Ink Character Reader (MICR) 7-15
magnetic reader files
processing 7-15
magnetic tape device characteristics 6-3
magnetic tape labels C-3
magnetic tape
label processing C-3
reading backwards 6-1, C-6
master index 2-4, 2-7
master index area (ISAM) 4-27
MFG operand G-2
MICR (Magnetic Ink Character Reader) 7-15
MICR
characteristics 7-15
checkpoint considerations 10-14
document buffer 7-15, 7-16, 7-17
document buffer format 7-20, 7-21
document processing 7-22
programming considerations 7-18
stacker selection routine 7-16
stacker selection timing 7-18
mixed format (macro operands) 1-9
MODDTL macro 10-27
module names, standard 1-5
modules, supplying the name for 1-5, 1-6
MRMOD macro 7-15
MRMOD macro
with 1270/1275 7-27
MTC job control statement C-6
MTMOD macro operands 6-3
multitask linkage example 10-10
Multitasking Functions 10-21
multitasking
attaching a subtask 10-23
private F-1
save areas required 10-21
subtask initiation 10-21
terminating a subtask 10-24
multivolume file processing (SAM) 3-11
MVCOM macro 10-5

non-data device operations
DAM 4-19
magnetic tape 6-6
optical readers 7-24
SAM 3-17
non-formatting write (WRITE UPDATE) 3-14
non-standard labels
checking, with PIOCS 9-8
writing, with PIOCS 9-8
non-standard tape labels 6-1
nonstandard labels C-6
normal end
of subtask 10-12
of task 10-11
notation, register 1-9
notation
S,address 1-10
notational conventions (macro operands) 1-9
NOTE macro
work file 3-15

obtaining a record (SAM) 3-5
OCR document processing 7-22
OMR coding example 7-5
OMR Data 7-3
OMR data
data card 7-4
data record 7-4
format descriptor card 7-3
OPEN macro processing
DASD standard labels C-1
diskette labels C-3
OPEN macro
DAM 4-9
DASD 4-3
ISAM 4-24
PIOCS 9-1
SAM 3-3, 3-4
with 1287/1288 7-28
opening a file
DAM 4-9
DASD 4-3
diskette 5-1
ISAM 3-26, 4-24
PIOCS 9-1
SAM 3-3
opening a work file 3-13
OPENR macro
DAM 4-10
ISAM 4-24
SAM 3-4
self-relocating programs D-3
operands
keyword 1-9
mixed format 1-9
positional 1-8
operator communication (see STXIT macro)
operator communication user exit 10-10
operator verification table 10-16
Optical Mark Reader(see OMR)
optical reader files

processing 7-20
optical readers, programming considerations 7-27
output area requirements (ISAM) 4-26
overflow area 2-2, 2-3, 2-4, 2-7, 2-8
cylinder 2-2
independent 2-2
ISAM 2-7, 2-8
overwriting existing records
DAM 4-17

page fault appendage F-1
page fault handling overlap F-1
page fault queue F-1
page fixing 10-2
page freeing 10-2
page-fix counter 10-2
page-in in advance 10-4
page-out; forcing 10-3
PAGEIN macro 10-4
paging 10-2, 10-3
paging mechanism 10-2, 10-3
paper tape
code translation on input 7-38
code translation on output 7-39
end-of-file (input) 7-38
error handling 7-39
file processing 7-35
I/O areas 7-39
input 7-36
output 7-38
programming considerations 7-35
record format (input) 7-37
record format (output) 7-38
trailer length (input) 7-38
partition related information 10-2, 10-3
Partition Communication Region 10-4, 10-5
passing parameters between phases 10-17
PDUMP macro 10-11
PFIX macro 10-2
PFREE macro 10-2
phase loading 10-1
physical block 3-2
physical DASD address (DAM) 3-21
Physical IOCS (PIOCS) 9-1
physical track address (DAM) 3-22
PIOCS (Physical IOCS) 3-33, 9-1
PIOCS macros, relationship between 9-9
PIOCS 3-33
activating a file 9-1
Alternate Tape Switching 9-12
checkpoint records, bypassing 9-13
CKD DASD Channel Programs 9-11
CLOSE macro 9-9
console buffering 9-12
Data Chaining 9-11, 9-12
diskette channel programs 9-12
extent processing 9-8
FBA channel programs 9-11
FEOV macro 9-9, 9-12
forcing end-of-volume 9-9
Initialization 9-1
label processing 9-2, 9-8
OPEN macro 9-1
opening a file 9-1
Processing 9-3
processing, LIOCS functions in 9-10
Programming Considerations 9-10
RPS 9-11
Termination 9-9
3800 Printing Subsystem restrictions 9-11
pocket light switches 7-22
POINTR macro
work file 3-15, 3-16
POINTS macro
work file 3-15, 3-16
POINTW macro
work file 3-15, 3-17
positional operands 1-8
POST macro 10-28, 10-29
preassembled logic modules 1-4
primary keys 2-1, 2-2
prime data area (DAM) 4-13
prime data area (ISAM) 4-27
prime data area
DAM 2-2
ISAM 2-4
prime data records 2-4
Printer Control 7-7, 7-11
printer control codes 7-12, A-1, A-2
Printer Overflow 7-10
printer-keyboard (see also console files) 7-14
printer
associated files 7-10
control codes 7-12, 7-13, A-1, A-2
error handling 7-14
file processing 7-10
UCS feature 7-13
priority, changing processing 10-22
private multitasking F-1
private subtask F-1
PRMOD macro 7-10
PRMOD macro operands 7-12
Processing Blocked Records (SAM) 3-8
processing considerations
associated files 7-3
Processing Console Files 7-14
processing DAM files 4-8, 4-10
processing DAM records
with Key area 4-11
without Key area 4-12
Processing DASD Files 4-1
processing data files (with SAM) 3-4
Processing Device-Independent System Files 8-1
processing diskette files 5-1
processing ISAM files 3-26, 4-26
Processing labels, magnetic tape 6-1
Processing Magnetic Reader Files 7-15
processing magnetic tape files 6-1
Processing Optical Reader Files 7-20
Processing Paper Tape Files 7-35
Processing Printer Files 7-10
Processing Punched Card Files 7-1
processing standard labels
PIOCS 9-8
Processing Unit Record Files 7-1
processing update files (SAM) 3-12
Processing with DAM 4-8, 4-10
Processing with SAM
DASD 4-2
processing work files (with SAM) 3-12
processing
at the completion of a page fault F-2
at the initiation of a page fault F-1
DAM input 4-10
DAM output 4-10
device-independent system files 8-1
Processing
PIOCS 9-3
program assembling B-1
program check (see STXIT macro)
program check user exit 10-8
Program Communication 10-4
Program Linkage 10-16
program loading 10-1
programmer logical units 1-4
programmer logical units
releasing 10-5
programming considerations

console files 7-15
interval timer with multitasking 10-8
ISAM 4-34
MICR 7-18
optical readers 7-27
paper tape 7-35
PIOCS 9-10
program, phase loading 10-1
2560 printing 7-6
3525 printing 7-6
5424/5425 printing 7-7
programming error processing routines
magnetic tape 6-5
sequential DASD 4-6
programming techniques for self-relocating programs D-2
programs
assembling with DTFs and logic modules B-1
link-editing 1-6
reenterable G-1
self-relocating D-1, 1-7
protecting resources 10-27
providing logic modules 1-4
PRTOV macro 7-10
PTMOD macro operands 7-36, 7-37
punch-read-feed feature (on card devices) 7-4
PUT macro
ISAM 4-33
printer 7-11
SAM 3-6
PUTR macro 7-15

random processing with ISAM 3-25, 3-30
random (direct) retrieval (ISAM) 3-30, 4-31
randomizing 2-2
methods, summary 4-16
to cylinder address with DAM 4-11
to track address with DAM 4-11
RCB (Resource Control Block) 10-24
RCB macro 10-24
RDLNE macro
with 1287/1288 7-29
read backward 6-1, C-6
READ macro
MICR 7-18, 7-19
with 1270/1275 7-27
with 1287/1288 7-28
with 3886 7-31
work file (SAM) 3-13
read-only module 10-32
reading a tape backwards C-6,6-1
reading blocks of data
DAM 4-14
reading data records
3886 7-31
Reading Magnetic Tape Backwards 6-1
real storage
fixing pages in 10-2
freeing pages in 10-2
record format
SAM 3-3
record ID (DAM) 3-21
record identifier (DAM) 3-21
record key (DAM) 3-18
record reference
by AFTER (DAM) 4-18
by ID 3-24, 4-16, 4-18
by Key 3-23, 4-16, 4-17
DAM 3-23, 4-16
record size
device-independent system files 8-1
SAM 3-3
system logical units 8-1
record types
DAM 3-19
ISAM 3-26
SAM 3-3
record zero (R0) 2-10, 3-21
DAM 3-21
reenterable programs G-1
reentrant module 10-33
reference methods (DAM) 3-21
register notation 1-9
register usage 1-8
register usage in page fault appendage F-1
registers, linkage 10-19
relative record 2-1
file 2-2
number 2-1
relative track address (DAM) 3-22, 3-21
RELEASE macro 10-5
Releasing I/O Units 10-5
releasing pages 10-3
relocatable library, using the 1-7
relocating address constants D-2
relocating loader D-1
RELPAG macro 10-3
RELSE macro (SAM) 3-11
reopening a file C-1
reorganizing a file (ISAM) 3-30
repositioning I/O files 10-15
repositioning magnetic tape 10-16
Requesting Control Functions 10-1
Requesting Storage Dumps 10-10
RESCN macro with 1287/1288 7-29
Resource Control Block (see RCB) 10-24
Resource Control Macros 10-27
Resource Definition Macros 10-27
Resource Protection 10-24, 10-27
Resource Sharing 10-27
restarting
a checkpointed program 10-14
repositioning files 10-15
restrictions
DTFDI 8-1
ISAM 3-25
3800 Printing Subsystem,with PIOCS 9-11
RETURN macro 10-16, 10-20, 10-21
rewinding magnetic tape C-3
RPS (rotational position sensing) example B-15
RPS DTF extension B-15
RPS
ISAM files 1-7
logic modules 1-7
PIOCS 9-11
support 1-7
RSTRT job control statement 10-14
rules for writing self-relocating programs D-1
RUNMODE macro 10-3
R0 (record zero) 2-10, 3-21

S,address notation 1-10
SAM (Sequential Access Method) 3-1
SAM declarative macros 1-3
SAM files, describing 3-2
SAM logic modules 3-17
SAM
activating (opening) a file 3-3
combined files 3-12
control interval format 3-1
deactivating (closing) a file 3-16, 3-17
declarative macros 3-18
describing records 3-2
forcing end of volume 3-11
GET macro 3-5
I/O area specification 3-3
logic modules 3-17
multivolume file processing 3-11
non-data device operations 3-17

processing data files 3-4
processing update files 3-12
processing work files 3-12
PUT macro 3-6
record format 3-3
record size 3-3
record types 3-3
work area specification 3-3
work files 3-12
sample DTFMT macro 1-3
save areas 10-19
required for multitasking 10-21
SAVE macro 10-16, 10-20
saving data for checkpoint/restart 10-14
SDL (Systems Directory List) 10-1
SECTVAL macro 9-4
seek, with DAM 4-17
Selective Processing of Blocked Records (SAM) 3-10
selective processing of work files 3-14
self-relocating programs D-1, 1-7
SEOF macro
with FBA DASD 4-8
sequence-link (SL) field 2-8
Sequential Access Method (SAM) 3-1
sequential organization, with index 2-3
sequential organization, without index 2-1
sequential processing
ISAM 3-25, 3-28
SAM work files 3-13
sequential retrieval (ISAM) 3-28, 4-32
sequentially organized files 2-1
SETDEV macro 10-14, 7-31
SETFL macro 4-27
SETIME macro 10-6
SETL macro 4-32
issuing a hold 10-31
SETPFA macro F-1
SETT macro 10-7
share control 10-27
shared files 10-32
shared modules 10-32
shared resources 10-27
shared track 2-8
Shared Virtual Area (SVA) 10-1, 10-4
sharing a resource among tasks 10-27
sharing a resource by subtasks 10-24, 10-25
skipping on printer 7-11
SL (sequence-link) field 2-8
Source-Program Macros 1-1
spacing on printer 7-11
spanned records
DAM 3-19
SAM 3-6, 3-7
stacker selection control codes A-1
standard labels
DASD C-1
diskette 9-3
magnetic tape C-4, C-6
processing with PIOCS 9-8, 9-10
standard module names 1-5
storage areas
ISAM 3-26
storage dumps, requesting 10-10
storage space formulas for ISAM 4-35
STXIT macro 10-8, 10-10
subtask termination 10-24
subtask
testing for attachment 10-22
subtasks sharing a resource 10-24, 10-25
superset/subset module names 1-5
supersetting/subsetting IOCS modules 1-5
supervisor area dump 10-10
supervisor macros 1-1
supplying the name for logic modules 1-5, 1-6
SVA (Shared Virtual Area) 10-1, 10-4
symbolic unit address 1-3
SYSnnn 1-4
SYSxxx 1-3
system control macros in reenterable programs G-2
system files, processing device-independent 8-1
system logical units 1-3
record sizes 8-1
system pack generation B-12
Systems Directory List (SDL) 10-1

table, DTF 1-2
tape drives, assigning/releasing 10-6
tape input files C-5
tape labels C-3
tape output files C-3
tapemarks C-3
task end 10-11
task timer 10-7, 10-8
setting 10-7
testing 10-7
unexpired time 10-7
user exit 10-8
TECB (timer event control block) 10-7
TECB macro 10-7
teleprocessing balancing 10-4
TESTT macro 10-7
time-of-day clock (TOD) 10-6
timer event control block
(see TECB) 10-7
timer services 10-6
TOD (time-of-day) clock 10-6
TPIN macro 10-4
TPOUT macro 10-4
track address (DAM) 4-11
Track Hold (see also DASD Record Protection)
track hold example 10-32
track index 2-5
track index entries 2-5
track reference
DAM 3-21
field types 3-23
trailer labels (see tape labels)
transmission information (CCB) 9-3
TRUNC macro
SAM 3-11
TTIMER macro 10-7

UCS
feature 7-13
job control command 7-13
unblocked records, SAM 3-6, 3-7
undefined records, SAM 3-6, 3-7
unexpired time
interval timer 10-7
task timer 10-7
unlabeled input files C-7
UNLOCK macro 10-27
update files (SAM) 3-7, 3-12
updating records on card devices 7-4
user exit (see STXIT macro)
user exit control 10-8
user exit
abnormal termination 10-8
interval timer 10-8
operator communication 10-10
program check 10-8
task timer 10-8
user option bits (CCB) 9-3
user standard labels C-2
checking, on disk C-2
checking, with PIOCS 9-8
DASD 4-5
tape 6-1

writing, on disk C-2
writing, with PIOCS 9-8
using registers 1-8
using RPS support 1-7
using system control macros in reenterable programs G-2
using the relocatable library 1-7
utility programs 4-10, 4-11

V-type address constants D-3
Virtual Storage Access Method 1-1
Virtual Storage Control 10-2
virtual storage dynamic allocation 10-4
VSAM macros 1-1
VSE/VSAM catalog management C-1
VSE/VSAM Macros 1-1
VTOC (volume table of contents) C-1

WAIT (PIOCS) macro 9-4
WAIT macro 10-7, 10-28
WAITF macro
DAM 4-19
MICR 7-18, 7-19
with 1270/1275 7-28
with 1287/1288 7-29
with 3886 7-32
waiting for a time interval to elapse 10-7
WAITM macro 10-28
work area
SAM 3-3
work file processing (with SAM) 3-12
work file (SAM)
deleting 3-13
DTF macro entries 3-13
macros 3-13
opening a 3-13
processing 3-12
retaining 3-13
work files
selective processing 3-14
sequential processing 3-12
WRITE macro
DAM 4-16
ISAM 4-28, 4-30
work file (SAM) 3-14
WRITE SQ (formatting write) 3-14
WRITE UPDATE (non-formatting write) 3-14
write verification, with DAM 4-17
writing blocks of data
DAM 4-16
writing self-relocating programs D-1
writing
nonstandard labels C-5
standard labels C-4
unlabeled files C-5
user standard tape labels 6-1

1255
processing 7-15
1259
processing 7-15
1270 7-15
programming considerations 7-27
1275 7-15
pocket light switch bits 7-22
programming considerations 7-27
1287/1288
CNTRL macro 7-24, 7-25
optical reader codes 7-24
programming considerations 7-27, 7-28
1419
pocket light switch bits 7-22
processing 7-15
stacker selection 7-16
1442
card read punch codes 7-8
stacker selection codes A-1
updating records 7-4
2311
capacities 4-1, 4-2
storage space formula (ISAM) 4-35
2314
capacities 4-1, 4-2
storage space formula (ISAM) 4-36
2319
capacities 4-1, 4-2
storage space formula (ISAM) 4-36
2401 characteristics 6-3
2415 characteristics 6-3
2420 characteristics 6-3
2520
card read punch codes 7-8
stacker selection codes A-1
updating records 7-4
2540
card read punch codes 7-9
stacker selection codes A-1
updating records 7-4
2560 7-1
card read punch codes 7-9
printing 7-6
updating records 7-4
2596
card read punch codes 7-9
3210 7-15
3211 7-12, 7-13
3211-compatible printers 7-12
3215 7-15
3262 printer 7-12
3289 7-12, 7-13
3310
capacity 4-2
3330
capacities 4-1, 4-2
storage space formula (ISAM) 4-37
3333
capacities 4-1, 4-2
storage space formula (ISAM) 4-37
3340
capacities 4-1, 4-2
storage space formula (ISAM) 4-38
3350
capacities 4-1, 4-2
3370
capacity 4-2
3410/3411 characteristics 6-3
3420 characteristics 6-3
3504/3505 7-1
card read codes 7-9
OMR Data 7-3
stacker selection codes A-1
3525 7-1
card punch codes 7-9
card printing codes 7-10
printing 7-6
updating records 7-4
3800 Printing Subsystem restrictions 9-11
3881 7-34
CNTRL macro 7-27
optical mark reader codes 7-27
programming considerations 7-34
3886 7-31
assembling a format record 7-32, 7-34
checkpoint considerations 10-14
CNTRL macro 7-26
COREXIT routine functions 7-33
document control 7-31
document marking 7-31

error handling 7-32
optical reader codes 7-26
programming considerations 7-31
5424/5425 7-1
card read punch codes 7-9
printing 7-7
updating records 7-4
8809
characteristics 6-3

SC24-5210-0

VSE/Advanced Functions Macro User's Guide (File No. S370/4300-30) Printed in U.S.A. SC24-5210-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

VSE/Advanced Functions
Macro User's Guide
SC24-5210-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name and mailing address:

____

____

____

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

What is your occupation? ______________________________

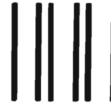
Number of latest Newsletter associated with this publication: ______________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC24-5210-0

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 812 BP
1133 Westchester Avenue
White Plains, New York 10604

Fold and tape | Please Do Not Staple | Fold and tape

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Cut or Fold Along Line

VSE/Advanced Functions
Macro User's Guide
SC24-5210-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name and mailing address:

____

____

____

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

What is your occupation? ______________________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC24-5210-0

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 812 BP
1133 Westchester Avenue
White Plains, New York 10604

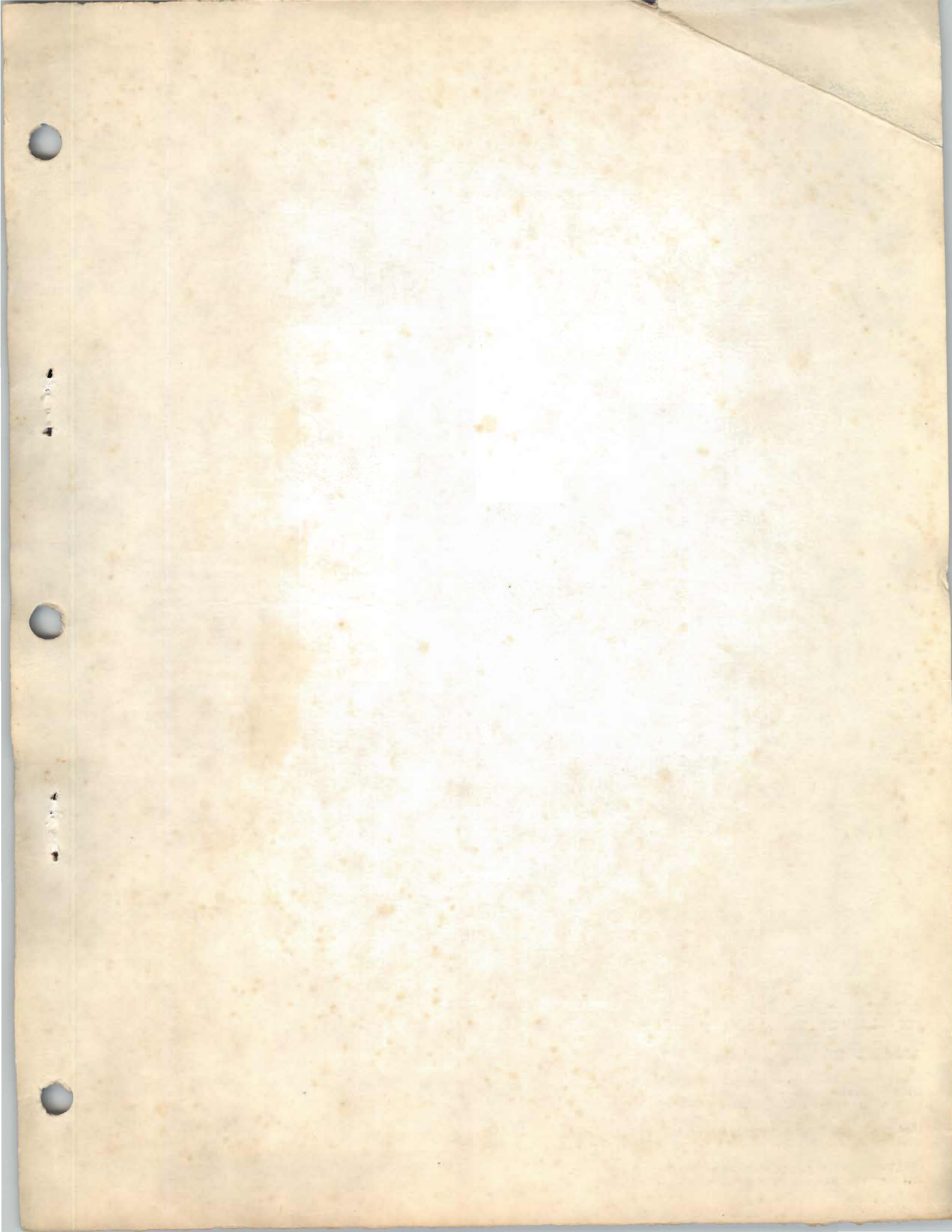
Fold and tape | Please Do Not Staple | Fold and tape

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

VSE/Advanced Functions Macro User's Guide (File No. S370/4300-30) Printed in U.S.A. SC24-5210-0



SC24-5210-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

VSE/Advanced Functions Macro User's Guide (File No. S370/4300-30) Printed in U.S.A. SC24-5210-0