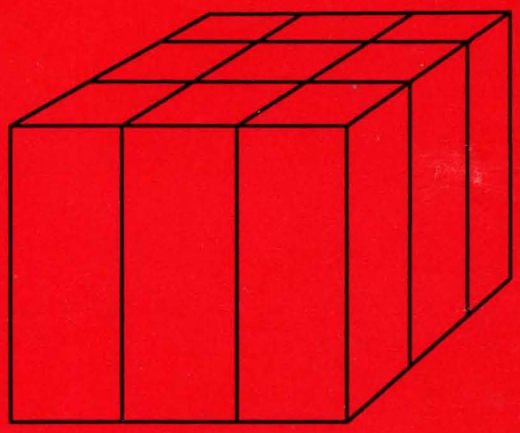
IBM

VSE/Advanced Functions

Application Programming Macro User's Guide



VSE/Advanced Functions

Application Programming Macro User's Guide

Program Number 5666-301

Order Number SC33-6196-1
File No. S370/4300-30

PREFACE

This book is a guide for programmers using the VSE/Advanced Functions macro instructions (macros). Use of both input/output control system (IOCS) macros and the system control macros is described.

After an introductory chapter on the types and use of macros, Chapter 2 gives a general description of how to define and process files with the Sequential Access Method (SAM). The subsequent chapters on file processing with SAM describe how to open, process, and close files on the different types of I/O devices, and also how to process device-independent system files. A chapter on system services includes discussions of such topics as virtual storage control, program linkage, and multitasking.

The use of the Direct Access Method (DAM) and Physical IOCS (PIOCS) is described in appendixes. (The Indexed Sequential Access Method, ISAM, is no longer described in this manual.) Several other appendixes on a variety of topics, and an index complete the book.

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

- IBM System/370 Principles of Operation, GA22-7000, or
- IBM 4300 Processors Principles of Operation, GA22-7070
- VSE/Advanced Functions, System Management Guide, SC33-6191
- VSE/Advanced Functions, Data Management Concepts, GC33-6192

All the macros mentioned in this manual are described in detail in:

- VSE/Advanced Functions Application Programming: Macro Reference, SC33-6197, referred to as Macro Reference manual in this publication.

You must be familiar with the assembler language as described in:

- OS/VS-DOS/VSE-VM/370 Assembler Language, GC33-4010.

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

- Introduction to IBM 3375 Direct Access Storage, GC26-1666.

CONTENTS

FIGURES

SUMMARY OF AMENDMENTS

Version 2, Release 1

This edition documents changes and additions resulting from the following new or changed functions:

- Virtual addressability extension
- New hardware support (IBM 3380)
- GETVIS area subpooling
- Device-independent full-track support
- Larger block size for DTFCD
- End-of-extent exit
- Adaptation to new librarian
- SAM tape file extension
- User interface for tape OPEN, CLOSE, and EOV
- Tape I/O module in the SVA
- Dropped paper tape support
- Performance improvement for PRT1/3800 printers
- ANSI security checking

APAR corrections and various editorial changes have also been included.

CHAPTER 1. MACRO TYPES AND THEIR USE

The use of macros simplifies the coding of programs and reduces the possibility of programming errors. A macro is a single assembler language instruction that generates a sequence of machine or assembler language instructions. The assembler uses what is called the macro definition to generate the sequence of instructions requested by the macro.

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 a sublibrary.

By its name, the source program macro indicates to the assembler which macro definition is to be called from the library. In the macro that you code in your program, you specify operands which the assembler uses, together with the called macro definition, to determine what instructions to generate. Figure 1-1 on page 1-2 shows a schematic example of a source program before and after a macro call.

There are two different types of macros: Data management or IOCS (Input/Output Control System) macros and control program macros.

IOCS Macros

IOCS macros are divided into declarative and imperative macros.

Declarative macros are of two related types - DTFxx macros and logic module generation (xxMOD) macros. The DTF macros define the files for the various access methods and I/O devices. The logic module generation macros define the logic modules that process the files. The purpose of the declarative macros is further discussed below.

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

Figure 1-2 on page 1-2 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. CRD is the name of the file. IJCFAOZ0 is the name of the logic module.

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

Control Program Macros

These macros, which are frequently called supervisor macros, enable you to make use of the system services provided by VSE/Advanced Functions, for example, the timer services or the multitasking functions. These macros are discussed in Chapter 8.

DTF MACROS

A DTF declarative macro must be coded for each logical file that your program wants to access by means of an imperative macro such as GET, PUT, READ, WRITE, CNTRL. The DTF 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. For example, if a GET macro is issued, the DTF macro supplies such information as:

- Record type and length.
- Logical unit name of the device from which the record is to be retrieved.
- Address of the area in storage where the record is to be made available for processing by your program.

Figure 1-3 on page 1-4 lists the imperative macros allowed for each DTF.

Figure 1-4 on page 1-6 shows an example of a DTF macro.

Logical IOCS (LIOCS)

The access methods SAM, DAM, and ISAM are collectively named LIOCS (Logical Input/Output Control System). LIOCS routines provide, on a logical level, all the functions needed to create, retrieve, and modify a data file.

For LIOCS operations, the DTF macro used depends upon the type of device used and on the type of processing that is to be performed:

- Processing with SAM: Applies to input/output with serial or diskette devices, or 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 1-5 on page 1-6. Processing files on the various types of devices by using SAM is described in Chapters 3 through 7.
- Processing with DAM: Whenever a file on a direct access device is to be processed by DAM, the DTFDA macro must be used. Processing with DAM is described in Appendix B.

DTF..	CD	CN	DA	DI	DR	DU	IS	MR	MT	OR	PH	PR	SD
READ			X		X		X	X	X[1]				X[1]
RELSE									X				X
RESCN										X			
SEOV									X				
SETDEV					X								
SETFL							X						
SETL							X						
TRUNC									X				X
WAITF			X		X		X	X		X			
WRITE			X				X		X[1]				X[1]

[1] Work files only
[2] Not for 2560 work files
[3] Data files only

Figure 1-3 (Part 2 of 2). IOCS Imperative Macros and DTFs

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

  Since ISAM does not support such devices as the 3375 or 3380, it should not be used for new programs; instead, VSAM should be used. Programs written for processing ISAM files can be used in a VSAM environment through the ISAM Interface Program. For reference purposes, the DTFIS macro is still described in the Macro Reference manual.

Physical IOCS (PIOCS)

As opposed to LIOCS, PIOCS (Physical Input/Output Control System) handles I/O operations on a more physical level by allowing direct control of the actual transfer of records through channel programs.

For PIOCS operations (EXCP, WAIT, etc.), the DTFPH macro is required 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. Processing with PIOCS is described in Appendix C.

DTF Table

A DTFxx macro generates a DTF table that contains indicators and constants describing the file. Generally, there is no need for an application program to access a DTF table. Should you need to reference a DTF table in your program (but only to test error information in the CCB or the labeled fields of the DTF expansion), you can reference this table by using the symbol 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.

Logical Units

In most of the DTF macros you can specify a logical unit name in the DEVADDR operand. This 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 logical 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 logical 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 logical 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 1-6 shows the relationship between the source program, the DTF table, and the job control I/O assignment.

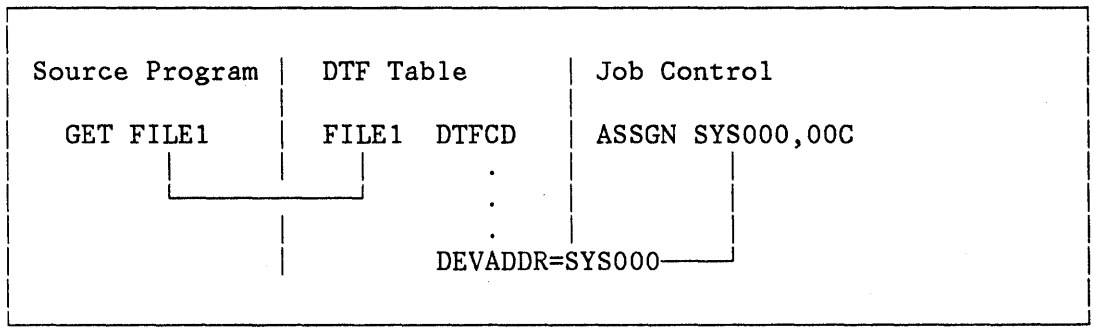


Figure 1-6. Relationship Between Source Program, DTF Table, and Job Control I/O Assignment

The fixed set of logical unit names that can be used with a DTF macro for a program in any partition is represented by SYSxxx. Logical units are divided into system logical units and programmer logical units.

Providing Logic Modules

To process a file for which you need a logic module, you must code a logic module generation macro (determined by your DTF) and assemble it either in-line with your program or supply it at link-edit time (see also Appendix A).

If the standard logic modules needed for your installation were assembled and cataloged at system generation time, you need not code them in your program. Instead, you can autolink the necessary modules from the library at link-edit time.

You need not code logic modules for:

- DTFSD, DTFDA, and DTFDI DASD files (except for ISAM files on DASDs without RPS)
- DTFMT magnetic tape files
- DTFPR, DTFDI when the actual device is a PRT1 or 3800 printer.

The support for these devices includes preassembled logic modules which are automatically loaded into the SVA (system virtual area) at IPL time and linked to the problem program during OPEN processing for the DTF.

Some of the module functions are provided on a selective basis, according to the operands 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 DTF indicating a module that does not contain a desired function, then an invalid supervisor call is generated, the job is terminated, and a message is displayed.

Subsetting/Supersetting of Logic Modules

Some modules may be subset modules to a superset module. A superset module is one that performs all the functions of its subset or component modules, avoiding duplication and thereby saving storage space. The functions required by several similar DTFs (that is, several DTFCDs, or several DTFPRs, etc.) are thus available via a single xxMOD macro, even if the DTFs have different operands. An example is shown in Figure 1-7 on page 1-10.

The supersetting/subsetting described below does not apply to DTFSD, DTFDA, or DTFDI DASD files or to DTFMT magnetic tape files. Any

Logic Module Names

As mentioned above, 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

If you omit the MODNAME operand from the DTF macro, IOCS generates a standard module name as determined by the functions required by the DTF.

Likewise, if you code your own logic module and omit the name from the name field, IOCS generates a standard module name matching that referenced in the DTF.

Standard module names used by IOCS are given under "Standard xxMOD Names," following the discussion of the appropriate xxMOD macro in the Macro Reference manual.

IOCS performs subsetting/supersetting of modules with standard module names by including, in a single module, the services required by a program's DTFxx macros for the same type of file.

If you are interested in seeing how IOCS forms subset/superset names, charts showing the name-building conventions are given for the various logic modules under "Subset/Superset xxMOD Names," following the discussion of the appropriate module in the Macro Reference manual. Figure 1-8 shows a model for these charts.

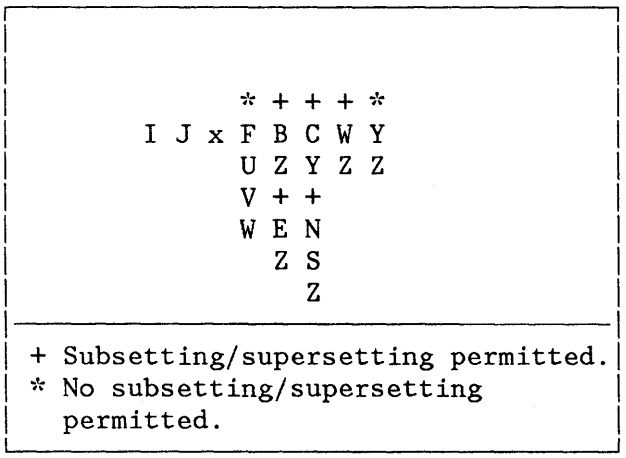


Figure 1-8. Model for a Subset/Superset Naming Chart

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 xxMOD Names" in the Macro Reference manual). If a module name were composed of letters from the top row exclusively,

registers may be used without restriction only for immediate computations.

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 beginning of each program phase, and leave it unchanged thereafter. However, when a reentrant (read-only) logic module is shared 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. IOCS does not save the contents of these registers before using them. If you use these registers, either save their contents (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 save area) in a subroutine, the save area back-chain pointer can be lost.

MACRO FORMAT

Macros, like assembler statements, have a name field, operation field, and operand field. Comments can also be included as in assembler statements. Macros require a comment to be preceded by a comma if they have no operand. Examples of these macros are CANCEL, DETACH, and GETIME.

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 operands in the operand field are either of positional format, keyword format, or they are mixed.

POSITIONAL OPERANDS: In this format, the operand 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 a work area named WORK is written:

```
GET CDFILE,WORK
```

CHAPTER 2. DEFINING FILES AND PROCESSING WITH SAM

You use the sequential access method (SAM) when your program requires that the records of a file are to be read as they are stored, one after the other, or that they are written to a file in the order in which they are built by that program. To process a file, SAM requires that you

- Define the file's characteristics using the appropriate DTFxx macro.
- Activate (open) the file.
- Issue the appropriate I/O request macros at those points in your program where the requested function is to be executed.
- Deactivate (close) the file after all of the file's records have been processed.

In the discussion of SAM, however, the use of FBA devices requires some special considerations, which are discussed below.

Control Interval Format

Control interval format is required by VSE/VSAM (Virtual Storage Access Method) and by FBA (Fixed Block Architecture) storage devices such as the IBM 3310. Since VSE/VSAM is not discussed in this manual, control interval format in this context refers to sequentially organized files that are stored on FBA devices.

Data is stored on FBA devices in fixed-length data blocks called FBA blocks, whose length is device-dependent. 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 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

This name is used in the macros that activate (OPEN) or deactivate (CLOSE) the file and request I/O operations to be executed (GET, PUT).

File Type (TYPEFLE)

SAM needs to know the type of file that is to be processed. Specify one of the following:

TYPEFLE=INPUT: The file is used for input via the GET macro.

You can update a disk input file by specifying UPDATE=YES in addition to TYPEFLE=INPUT. In this case, records are read, processed, and then written back into the same record location from which they were read. For details, see "Processing Update Files" on page 2-19.

TYPEFLE=OUTPUT: The file is used for output via the PUT macro.

TYPEFLE=WORK: Specifies a magnetic tape or disk file that can be used as work file, that is, for both input and output. It can be used to pass intermediate results between successive phases or job steps or to be read and written within a single phase. Work files use unblocked records of fixed length or undefined records and can be processed with the READ/WRITE, NOTE/POINTx, and CHECK macros. For details, see "Processing Work Files with SAM" on page 2-20.

TYPEFLE=CMBND: Files on card read-punch devices can be combined input and output files (like update and work files), that is, an updated record is punched into the same card from which it was read. For details, see "Processing Update Files" on page 2-19.

Record Format (RECFORM)

SAM needs to know the format of your records, that is, whether they are of fixed or variable length and whether they are blocked or unblocked. Enter one of the following:

FIXUNB for fixed-length unblocked records

FIXBLK for fixed-length blocked records

VARUNB for variable-length unblocked records

VARBLK for variable-length blocked records

SPNUNB for spanned variable-length unblocked records

SPNBLK for spanned variable-length blocked records

If you do not specify an I/O area in your DTFSD macro, the system issues a GETVIS macro to obtain the area from the partition GETVIS area.

Block Size (BLKSIZE)

This operand specifies the length of the I/O area. If the record format is variable or undefined, enter the length of the largest block of records. Consult the discussions of the various devices in the later chapters of this manual for the permitted BLKSIZE ranges.

For optimum use of the storage capacity of your DASD device you can specify BLKSIZE=MAX in the DTFSD for a sequential disk file. This sets the length of the I/O area equal to one full track of the device on which your file resides.

For FBA devices, BLKSIZE=MAX sets the I/O area length equal to the maximum CISIZE (32K) minus seven bytes, except for FIXBLK record format, where the length of the I/O area will be the highest even multiple of RECSIZE that is not larger than 32K minus seven.

With BLKSIZE=MAX, the length of the I/O area depends on the DASD device type. After OPEN, the halfword labeled filenameB in the DTFSD contains the actual size of the block minus one byte for TYPEFLE=INPUT/OUTPUT, or it contains the actual size of the block for TYPEFLE=WORK.

I/O Register Specification (IOREG)

This operand specifies the register in which IOCS places the address of the logical record that is available for processing if:

- Two I/O areas (and no work area) are used, or
- No I/O area has been specified, or
- Blocked records are processed in the I/O area, or
- Undefined or variable-length magnetic tape records are read backwards.

If you have omitted the IOAREA operand (to let OPEN issue a GETVIS request for an I/O area of the required length), OPEN returns the address of the acquired I/O area in the IOREG register.

For output files, IOCS places, into the specified register, the address of the area where you can build a record.

The register specified in the IOREG operand always contains the absolute address of the next available input record or of the record area where you can build the next output record.

```
Name      Operation     Operand          Column 72
-------------------------------------------------
FNAME     DTFMT         ...                 X
                        IOAREA1=A1,         X
                        IOAREA2=A2,         X
                        WORKA=YES,          X
                        BLKSIZE=500,        x
                        RECSIZE=100,        X
                        ...
A1        DS            500C
A2        DS            500C
          .
          .
          .
          GET           FNAME,A3
          .
          .
          .
A3        DS            100C
          .
```

Figure 2-1. GET Macro Processing Example

Error Handling (ERROPT, WLRERR, and ERREXT)

The macro operands that you may use for processing I/O and record-length errors are ERROPT, WLRERR, and ERREXT. Not all of these operands apply to all DTF files; for complete information, refer to the subsequent chapters or to the Macro Reference manual.

The ERROPT operand is used when a read or write error is encountered. It has three parameters - IGNORE, SKIP, or the name of your error routine (for output files, only IGNORE and name are valid):

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

End-of-Extent Exit Specification (EOXPTR)

This operand specifies the name of the pointer to your end-of-extent routine. IOCS branches to this routine when the last (or only) extent is reached during an output operation on an output or work file.

ACTIVATING A FILE FOR PROCESSING

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

In the OPEN macro, enter the symbolic name of the file to be opened (DTF filename). Alternatively, you can load the address of the DTF filename into a register and specify the register using ordinary register notation. You can activate up to 16 files with one OPEN by entering additional filenames. Before you open a file, you should make sure that the file is not already open.

The OPEN macro, together with the ASSGN job control statement, 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 before processing is attempted; an exception is that an OPEN need not be issued for DTFCN 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" on page 2-26).

For an output file with two I/O areas or no I/O area specified, OPEN loads your IOREG register with the address of an I/O area.

OPEN also checks or writes standard or non-standard DASD or magnetic tape labels. Whenever you open an input/output DASD or magnetic tape file 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.

If OPEN attempts to activate a file whose device is unassigned, the job is terminated.

Self-relocating programs must use OPENR for file activation. OPEN and OPENR perform essentially the same functions, except that when OPENR is specified, the symbolic addresses that are generated are self-relocating. Throughout the manual, the term OPEN refers also to OPENR, unless stated otherwise.

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

2. Subsequent GETs either increment 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 operands RECFORM=SPNUNB or SPNBLK and WORKA=YES must be included in the DTF. GET assembles spanned record segments into logical records in your work area. Null segments are recognized but are not assembled into logical records; they are skipped. The length of the logical record is passed to you in the register specified in the DTF RECSIZE operand.

If you update logical records (UPDATE=YES), 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. (A record cannot span volumes.)

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 you choose to put several logical records into one undefined record. 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.

Issuing a GET macro after the last record of an input file has been accessed 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 volume. When a file occupies more than one area on a DASD volume, automatic switching from one extent to the next is also performed.

length of the output records from the length of the corresponding input records.

When variable-length blocked records are built in a **work area**, the PUT routines check the length of each output record to determine if the record fits into 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 to the output 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 into the remaining portion of the output area. This must be known before record processing for the next record begins, so that the completed block can be written.

The amount of space available in the output area at any time is supplied to your program in the register specified in the DTF VARBLD operand. (This register is in addition to the register specified in IOREG.) After each time a PUT macro has been executed, IOCS loads into the specified register the number of bytes remaining in the output area. You then compare the length of your next variable-length record with the available space to determine if the record fits in the area. If it does not fit, you must issue a TRUNC macro to transfer the completed 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 records in your work area are divided into spanned record segments according to the length specified in the BLKSIZE operand. For disk output, spanned records must not span volumes. If there is not enough space on the current volume to contain a spanned record, the logic module attempts to put the entire spanned record on the next volume.

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.

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.

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

Operating with GET and PUT without a separate work area provides faster performance, since there is no need for data transfer between the work area and the I/O areas. A combination of GET with a work area, and PUT without a work area 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.

Record Format	Number of I/O Areas	Separate Work Area	Amount of Maximum Achievable Overlap
Un-blocked	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 2-2. Overlap of Processing and I/O

Selective Processing of Blocked Records

Mostly, a program will process a file starting with the first logical record and proceed until end of file. 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. Also, blocked records allow a more effective use of I/O devices.

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 you process blocked spanned records, 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 TRUNC macro is used with blocked output records written on DASD or magnetic tape. It allows you to write a truncated 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 similar to that of the RELSE macro for input records. That is, when the end of a category of records is reached, the current block can be written and the new category can be started at the beginning of a new block. The CLOSE macro truncates the last block of a file.

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 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=CMBND in the DTFCD 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 macro.

GET obtains records from the file in the usual way. 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.

In the combined card file example of Figure 2-4 on page 2-20 information from each card is read, processed, and then punched into the same card to produce an updated record.

Processing is done in the input area. After processing, the records are returned from the input area. For card devices, the records are returned by PUT; for DASD, 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 work area is used for the file, PUT returns the records from the work area to the input area and then from the input area to the file. (When control interval format is used, the CI buffer is the primary input area and any user-specified input area is bypassed when a work area is specified.) For spanned records, you must have a work area that is sufficiently large to hold the entire spanned record.

- 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.
- If you use CI format, the number of logical blocks per control interval is limited to 255. This means that an error condition may occur if you attempt to use a CI-format work file that was not created or modified by SAM.
- An existing work file may not be extended.

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. The CLOSE routines then 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; 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 (for example, 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 and declared as a standard labeled work file (DTFMT FILABL=STD), the TLBL job control statement must be specified. OPEN/CLOSE reads/writes the standard header and trailer labels as for standard labeled data files.

For fixed-length unblocked records, the number of characters to be written is specified in the BLKSIZE operand of your DTFxx macro. You can change this number (which is stored in the DTF table) at any time by referencing the halfword filenameL.

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 is 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.

Note, however, that an end-of-file record is written only if the last operation before the CLOSE macro was a WRITE SQ; a CLOSE macro following a WRITE UPDATE protects the updated file by not writing an end-of-file record.

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. To do this you must use the NOTE and the POINTx macros.

During processing you may issue the **NOTE** macro. It returns information about the position of the block just read or written.

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 a 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.

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

```
        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 2-5. Example of POINTS Macro with Work File Processing

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 after 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

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 data 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 in the usual way.

Closing a File

You must issue a CLOSE macro to deactivate all files that were activated by an OPEN macro, with the exception of console (DTFCN) files. Up to 16 files may be deactivated with one CLOSE.

After you have issued a CLOSE macro, no further commands can be issued to the file unless it is reopened. Sequential DASD files can only be successfully reopened for output if 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.

Note that if you issue CLOSE to an unopened magnetic tape input file, the option specified in the DTF REWIND operand is performed. If you issue CLOSE to an unopened magnetic tape output 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, the symbolic addresses that are generated from the parameter list are self-relocating. Throughout the manual the term CLOSE refers also to CLOSER, unless stated otherwise.

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

[1] Not needed for DASD devices and PRT1 or 3800 printers.
[2] Not needed for PRT1 and 3800 printers.

Figure 2-6. Logic Modules for SAM

CHAPTER 3. PROCESSING DASD FILES WITH SAM

This chapter briefly describes how to open and close a DASD file with SAM and how to write your error processing routines. The information is intended as an addition to the general information given in Chapter 2. Processing a DASD file with DAM is described in Appendix B.

FBA (FIXED BLOCK ARCHITECTURE) DASD PROCESSING

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 re-evaluated 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 formatted WRITE or PUT is issued, the force-write (PWRITE=YES) operand must be specified on the DTFSD macro. Note, however, that writing a whole CI buffer for each record can cause severe performance degradation.

SAM automatically reformats CIs into logical blocks and vice versa, as it automatically blocks and deblocks.

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.

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
```

is 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 a file is opened, 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

represent the address of the DTF table, and the second four bytes 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), and the second address to access the record for error processing. (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 non-recoverable 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 it attempts to write the block with an ERET RETRY.

  - or 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 error routine when processing record length errors. For this condition, ERET RETRY results in job termination.

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 in the usual way.

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.

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 ensure 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 unless the DTFSD table is saved before the file is first opened, and restored between closing the file and reopening it again.

CHAPTER 4. PROCESSING DISKETTE FILES WITH SAM

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. Alternatively, you may use DTFDI to provide device independence for system logical units. (See Chapter 7 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 Appendix C).

There are two logic module generation macros associated with DTFDU: DUMODFI for input files, and DUMODFO for output files.

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.
- checks that there is only one file open per device and only one extent specified per volume.
- 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 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

processing. Therefore, command-chained records are handled as follows:

- 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.

- 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.

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 4-1.

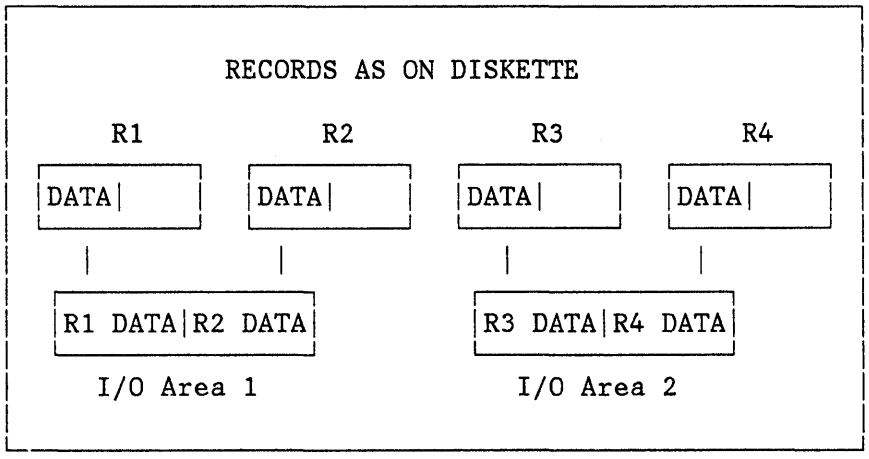


Figure 4-1. Diskette Data Transfer on Input

Physically, the diskette records are always 128 bytes in length (see Figure 4-3 on page 4-4). 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 4-2 on page 4-4.

absolute address of the first data record, or it transfers the first record to the specified work area. 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 in 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 handling 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. Specify either IGNORE, SKIP, or the name of an error routine.

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

CHAPTER 5. PROCESSING MAGNETIC TAPE FILES WITH SAM

This chapter describes how to process magnetic tape files with SAM. It discusses such topics as label processing, error handling, and non-data device operations. The information is intended as an addition to the general information on processing with SAM, as described in Chapter 2.

SECURITY (ACCESSIBILITY) CHECKING FOR ANSI STANDARD LABELS

In accordance with the ANSI Standard, Level 3, VSE/Advanced Functions denies access to ASCII files if these are access-protected.

To this purpose, a dummy installation phase in the SVA ($IJJTSEC) is delivered with your system. This phase checks the accessibility byte in the VOL1 and HDR1 labels. If this byte is not blank (X'40'), X'00', or C'0', the job will be canceled.

You can adapt this phase to your own installation requirements by changing the A-type macro IJJT$SEC. Reassemble the installation phase by preparing and executing the following job stream:

```
// JOB IJJTSEC
// OPTION CATAL,NODECK
// EXEC ASSEMBLY
   COPY IJJT$SEC
   END
/*
// EXEC LNKEDT,PARM='MSHP'
/&
```

SAM TAPE FILE EXTENSION

You can append additional data records to an already existing standard-labeled data file by

- Defining the file with the parameters FILABL=STD and TYPEFLE=OUTPUT in the DTFMT macro and
- Specifying the // TLBL job control statement with the DISP= operand:

```
// TLBL file-name,,,,,,,,,DISP=<OLD|NEW|MOD>
```

where

DISP=OLD specifies that an existing file

- If the tape is not positioned at load point and the file-id specified in the // TLBL statement matches the file-id in the succeeding file.

- When a file sequence number is specified in the // TLBL statement:

  - If a file with this sequence number is on the volume set and the file-id of this file matches the file-id specified in the // TLBL statement.

In any of the above cases, the file to be extended may span more than one volume. IOCS positions the tape to the end of the last data record in the file before extension starts.

Processing Standard Labels

- For DISP=NEW (and when DISP=MOD has been changed to DISP=NEW):

  - As for standard labeled OUTPUT files when they are created.

- For DISP=OLD (and when DISP=MOD has been changed to DISP=OLD):

  1. For the file to be extended:

     - All fields in the standard header label will be compared with the appropriate specifications in the // TLBL statement.

     - Error messages will be printed when there are any discrepancies.

       Note: The standard header label (HDR1) of the file to be extended will not be changed.

       Originally, the creation date in the standard header label (HDR1) as well as in the standard trailer label (EOF1) reflects the date when the file was created.

       After the file has been extended, the creation date in the standard trailer label (EOF1) reflects the date when the file was extended.

       The expiration date in the standard trailer label (EOF1) will not be changed.

  2. For the file following the one to be extended:

     - If it is a standard labeled file, the expiration date of this file will be checked and error message 4119D FILE UNEXPIRED will be given if the file has not expired. On reply=IGNORE, the file will be overwritten.

- Register 15: Entry point of $IJJTXIT

On return from $IJJTXIT, registers 0-14 will be restored and register 15 has to contain one of the following return codes:

- 0: Always returned except for end-of-volume (EOV) processing.
- 0 (EOV processing): Normal alternate tape assignment processing is to be done.
- 4 (EOV processing): Normal alternate tape assignment processing is to be skipped (indicating that the tape management code has made the alternate assignment and ensured that the correct volume is now mounted).

USER LABEL PROCESSING

You can issue an LBRET macro in your program when you have completed label processing and wish to return control to IOCS. LBRET is used for subroutines that write or check magnetic tape user-standard or nonstandard labels. The operand used (1 or 2 for tape labels) depends on the function to be performed. The functions and operands are explained below.

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 Non-Standard Tape Labels: You must process all your non-standard labels together. Use LBRET 2 after all label processing is completed and you want to return control to IOCS.

BLOCK SIZE

The BLKSIZE operand of the DTFMT macro specifies the number of bytes transferred between the I/O area and the 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 is truncated to fit into 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-length records, the minimum physical record length is 18 bytes. If less than 18 bytes are

non-standard 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).
- Standard trailer label (and user-standard labels, if any).
- 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 statements). If non-standard labels are specified, FEOV allows you to write trailer labels on the completed volume and header labels on the new volume, if desired.

PROGRAMMING YOUR ERROR PROCESSING ROUTINES

IOCS branches to your error processing routine named in the ERROPT=name operand when

- a non-recoverable I/O error is encountered (ERREXT=YES specified)
- a tape read data check is encountered (ERREXT=YES not specified).

You may perform any kind of error processing in your error routine; however, you must observe the following 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 a 2-part parameter list when an error condition occurs. The first four bytes of the list contain the address of the DTF table, and the second four bytes 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

RECIZE 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 LIOCS 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 5-1 on page 5-10 summarizes the DTFMT error options for various combinations of error specifications and errors.

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:

- The FSR (or BSR) function permits you to skip over a physical tape record (from one inter-record 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).
- The functions of FSR, FSF, BSR, and BSF always start at an inter-record gap.
- 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.
- 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.
- If FSR or BSR is used, LIOCS does not update the block count. Furthermore, IOCS cannot sense tapemarks on an FSR or BSR

CHAPTER 6. PROCESSING UNIT RECORD FILES WITH SAM

Unit record files are, in general, characterized by utilizing a wide variety of storage media. These range from punched card 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 files, 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

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" on page 6-32 for more information on processing 3881 files.

The range of punched card equipment provided by IBM allows you to select devices that best support your 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" on page 6-4.

Programming 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 assigned IGN.

READ-PUNCH ASSOCIATED FILES: For RP associated files, the GET for the read file and the PUT for the punch file must both be issued for each card. If punching is not desired, the output area or the work area must be filled with blanks. LIOCS tests for blanks in the output area or work area 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 work area; 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 work area 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 work area; only the data portion following the

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 'b' indicates the blanks:

  MbMbMbbM

- 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:

  MbHbHHH

- 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:

  MbMbbMbM

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 6-1 on page 6-6 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 your 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

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.

For the card devices, there are two ways of specifying in the DTFCD that such updating is desired; which way is used depends on the 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 in Figure 2-4 on page 2-20. 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. The first PUT must, of course, also preceded by a GET. GETs 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" on page 6-16.

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 2-4 on page 2-20 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).

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 equivalent 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.

Closing a File

For the 2560, 3525, or 5424/5425, CLOSE 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 6-2 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 6-2. CLOSE Card Movement for the 3525

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 RECFORM=UNDEF, the length of the record must be at least 2, if RECFORM=VARUNB, it must be at least 6.

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 F 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 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

- 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 the Macro Reference manual. (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 selects 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.

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 6-3. GET/CNTRL/PUT Macro Usage. This macro sequence processes one card of an associated file.

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" on page 6-20.

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 the DTFPR macro in the Macro Reference manual.

When you issue a PUT macro for a printer file, this PUT causes the 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 to advance the paper on the printer by one line.

Note: Printing without spacing can only be done with the CTLCHR=code operand.

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 sent to the device is used as follows:

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 (print train). 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 will take place. For any UCS printer, the NOCHK option of the SYSBUFLD program and the UCS parameter in the DTFPR 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 operated 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 for any character, and the upper-case equivalent of bits 2 to 7 is printed (for characters other than A through Z, there may not be an upper-case equivalent). 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.

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; 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 (PRT1). 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 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; however, you should not issue any imperative macro instruction for the file invoking the error exit. 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.

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. When you use the PUTR macro, do not use register 2 as base register.

You may also use PUTR with the 3210 or 3215 console printer-keyboard, in which case PUTR functions in the same way 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.

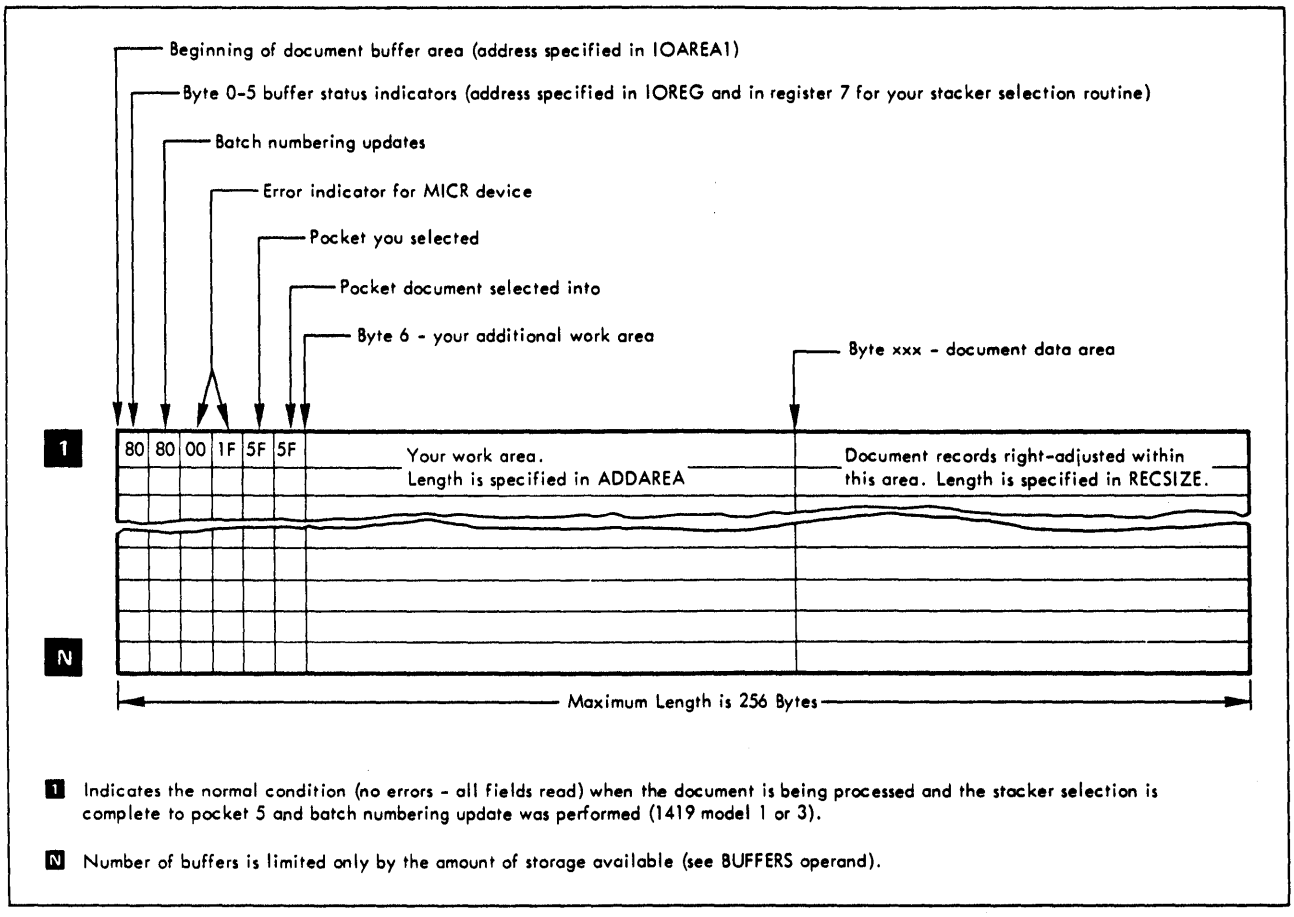


Figure 6-4. MICR Document Buffer

The MICR document buffer format is given in detail in the Macro Reference manual.

Stacker Selection Routine

Your stacker selection routine is 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:

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 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 a 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 auto-selected 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

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, 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 (under the CHECK macro in the Macro Reference manual).

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 6-5 on page 6-30) within

(1) your 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.

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 non-data 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 non-data 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.) 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.

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 issued 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

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 the CNTRL macro in the Macro Reference manual).

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 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').

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.

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

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 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

The bit configuration for the pocket light switch area is shown under the LITE macro in the Macro Reference manual. 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 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

When the RESCN 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

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 non-recoverable 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 type of 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 non-recovery 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:

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

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'	Non-recovery 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
- Non-recovery error
- Permanent error.

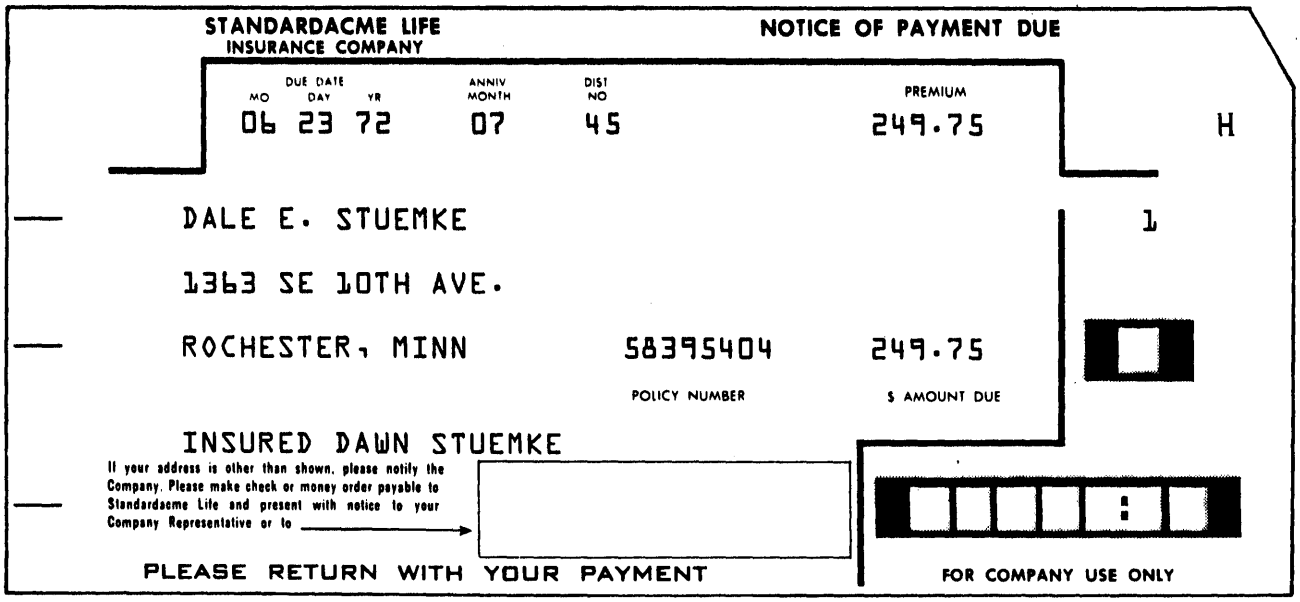


Figure 6-6. Premium Notice Example

To process documents like that in Figure 6-6, 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 6-7 on page 6-50. The numbers at the right 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 @ 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:

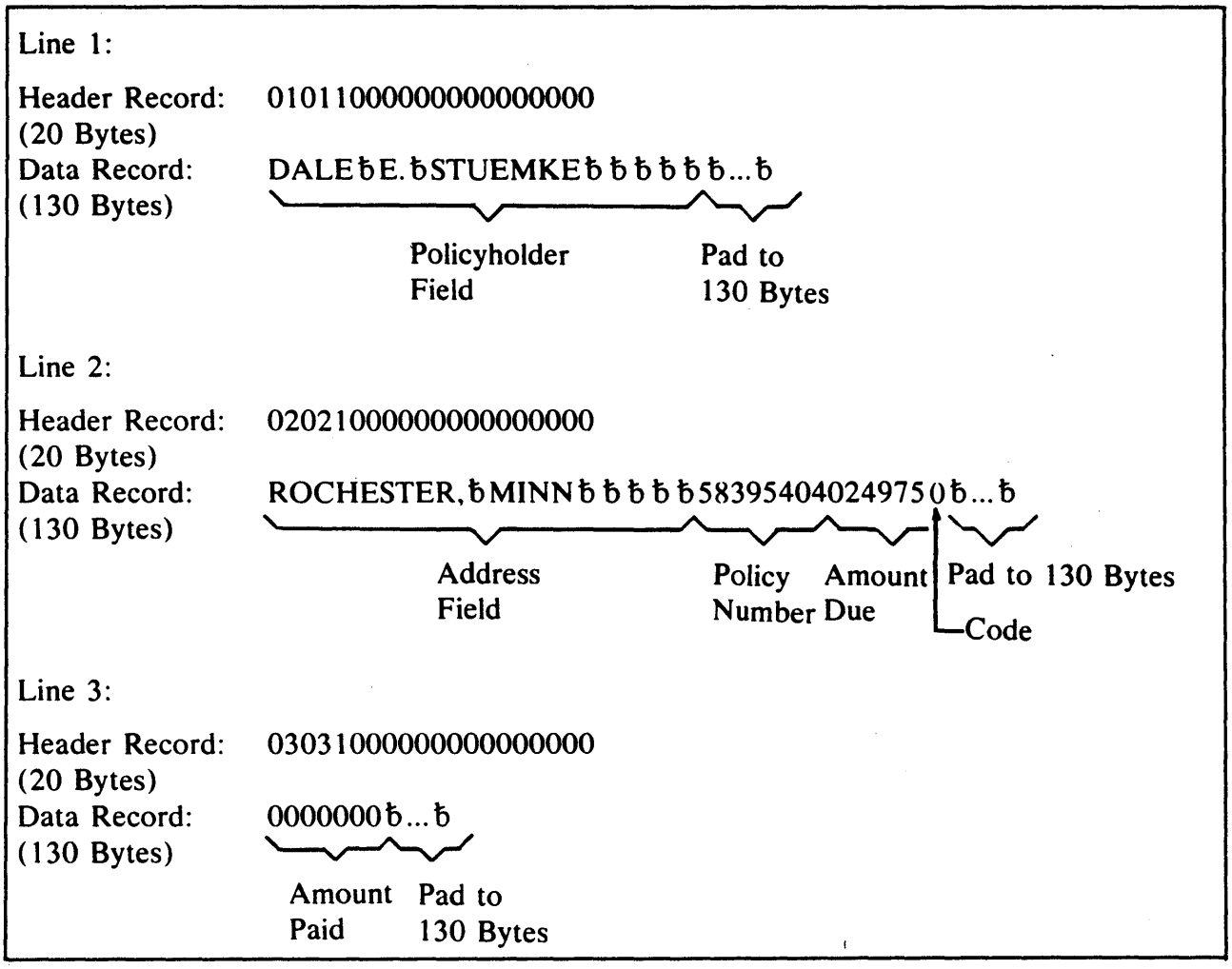


Figure 6-8. Sample Data

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.

- 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 (WORKA=YES) is not permitted with the 3881.

Only fixed, unblocked, input records are valid for the 3881.

CHAPTER 7. PROCESSING DEVICE-INDEPENDENT SYSTEM FILES WITH SAM

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 less storage than device-dependent LIOCS macros.

If you are using a DASD device or a PRT1 or 3800 printer, you do not need to specify DIMOD. Support for these devices includes pre-assembled logic modules that are automatically 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 and for PRT1 or 3800 printers will override the DIMOD linkage if the file is assigned to such a device.

The DTFDI macro should always be used to read SYSIPT data if the program might be invoked by a cataloged procedure, because in that case the input data might be part of the 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.

80 bytes for SYSIPT.
80 bytes for SYSRDR.
81 bytes for SYSPCH.
121 bytes for SYSLST.

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 operand.

- 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.

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 8. REQUESTING CONTROL FUNCTIONS

PROGRAM LOADING

Phases are normally loaded by the supervisor in response to the job control statement // EXEC phasename. 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 sublibrary, or a private sublibrary, 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

```
         .
         .
         .
         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
```

Figure 8-1. LOAD Macro Example

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).
- determine the mode of execution of a program (RUNMODE macro).
- extract 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.
- allocate and release virtual storage dynamically.

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 VSE/Advanced Functions, System Management Guide.

```
            .
            .
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 8-2. PFIX and PFREE Example

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 system sublibrary, 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.

the specified pages are already in real storage when the macro is issued, they are given the lowest priority for page-out.

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 operand 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 8-3 on page 8-8 shows the portion of the communication region that may be of interest to you. This information is described below.

Field Name	Length (bytes)	Information
JOBDATE	8	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 or dd/mm/yy where mm is month, dd is day, yy is year. This date can be temporarily overridden by a DATE statement.
	4	Reserved
COMUSCR	11	User area for communication within a job step or between job steps. All 11 bytes are set to zero whenever a JOB or end-of-job statement for a job is encountered.
UPSI	1	UPSI (user program switch indicators). Set to binary zero when a JOB or end-of-job statement for the job is encountered. Initialized by the UPSI job control statement.

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.

The macro unassigns any specified programmer logical 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.

A magnetic tape, disk or unit record device 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 by using the ASSIGN macro. This macro is also used for dynamic release of the

If ASPL DSECT=NO (default) has been specified, the following is generated:

```
name      DS      0XL5
ASGFUNCT  DS      XL1       User has to provide
                            function code
```

The following function codes can be specified:

```
ASGDPT    EQU     X'80'     Temp. progr. unit - disk
ASGDST    EQU     X'84'     Temp. system unit - disk
ASGDSP    EQU     X'8C'     Perm. system unit - disk
ASGTPT    EQU     X'40'     Temp. progr. unit - tape
ASGTPD    EQU     X'42'     Temp. progr. unit:
                            specific tape
ASGUAP    EQU     X'28'     Unassign progr. unit
ASGUAS    EQU     X'2C      Unassign system unit
ASGCHG    EQU     X'10'     Change temp. to perm.
ASGURT    EQU     X'02'     Device other than temp.
                            disk/tape progr. unit
ASGLUNO   DS      0XL2      Logical unit number
ASGCLASS  DS      XL1       Logical unit class
ASGPROG   EQU     X'01'     Programmer class
ASGSYST   EQU     X'00'     System class
ASGLUNDX  DS      XL1       Logical unit index
ASGCUU    DS      XL2       Physical unit number
ASGLNG    EQU     *-ASGFUNCT  Length of ASPL
```

TIMER SERVICES AND EXIT CONTROL

Timer Services

VSE provides the following timing facilities:

- Time-of-day clock
- Interval timer
- Task timer

Time-of-Day Clock

The time-of-day (TOD) clock is a standard high-resolution hardware feature. Any program executing under VSE can obtain the time of the 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.
- In microseconds.

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 seconds, 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.

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 in 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 interrrupt occurs, the interrupt is ignored.

When a program is restarted from a checkpoint, the timer interval set by the SETT macro is not restarted.

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, in hundredths of milliseconds (in binary), in register 0.

The time remaining in the interval can be canceled by specifying CANCEL in the TESTT macro. This prevents the task timer exit routine from being entered.

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 8-6 on page 8-16 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. To return from a user exit routine, always use the EXIT macro.

```
TIMECHK   START X'78'
          BALR  R9,0
BASEADDR  EQU   *
          USING BASEADDR,R9        Establish addressability
          STXIT IT,TIMINTR,TIMSA.  Set up link to timer routine
          SETIME 1800              Timer interrupt every 30 min.
          .
          .
PROCESS   EQU   *
          .
          .
(perform normal processing)
          .
          .
          B     PROCESS
*
* TIMER INTERRUPT ROUTINE
*
TIMINTR   EQU    *
          BALR  R9,0
          L     R9,ABASE-*(R9)     Establish addressability
          .
          .
(perform IT exit processing)
          .
          .
          SETIME 1800              Set up next interval
          EXIT   IT                Return to interrupted point
*
* CONSTANTS
*
ABASE     DC      A(BASEADDR)
TIMSA     DS      0CL72            IT exit routine save area
SAPSW     DS      D                Interrupt status information
SA00      DS      9F               Registers 0 to 8
SA09      DS      7F               Registers 9 to 15
*
R9        EQU     9
          END
```

Figure 8-7. Example of Using the Interval Timer Exit

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 8-8 on page 8-18 illustrates this approach.

After the appropriate action is taken, your abnormal termination routine may either resume processing using the EXIT AB macro (main task only) or terminate the task with CANCEL, DETACH, DUMP, JDUMP, EOJ, or RETURN (if RETURN=YES in the ATTACH macro).

For a main task, the whole job is terminated if OPTION=DUMP has been specified explicitly or by default. Only the current job step is terminated if OPTION=NODUMP and the termination macro used was either DUMP or EOJ.

If OPTION=EARLY is specified in the STXIT AB macro, the abnormal termination routine will be invoked for any type of termination (normal or abnormal) and, for a main task, before its subtasks are terminated.

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.

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 8-9 on page 8-20 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 enters MSG followed by the partition identifier (such as BG or F2), which sets the linkage to the user's operator-communication exit routine, which may perform any processing.

Since an operator communication exit routine is performed asynchronously with the main routine of your program, be careful when using the same resources (such as data and instructions) in

Task-Timer User Exit

Task timer support may be generated via the FOPT generation macro.

The time interval for the task timer is specified in the SETT macro. When the interval has elapsed, the exit routine specified in the STXIT TT macro is entered. The linkage to the exit routine must have been established before an interrupt occurs; otherwise, the interrupt will be ignored. The macro can be issued only by 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 content of the registers are restored from the save area. It is important, therefore, that the content of the save area specified in the associated STXIT TT macro is not destroyed.

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 Diagnosis: Service Aids.

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, 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 (if issued by a subtask, then only that subtask is terminated).

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 in the section "Requesting Storage Dumps" on page 8-21. When issued by the main task, the DUMP macro causes the job step, and the JDUMP macro the entire job, to be terminated. If one of these macros 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 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 in the section "Timer Services and Exit Control" on page 8-13) causes control to return to the point of interruption within the main-line routine; thus the task continues processing.

For an AB exit routine, control is returned to the instruction following the EXIT AB macro.

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 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. 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.

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 8-11 on page 8-26 and described below.

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 8-11 (Part 2 of 2). 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.

- 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.

```
Code in calling routine:
            LA          13,SAVAREA1                 A
            .
            CALL        SUBROUT,(PAR1,PAR2)         B
            C           12,ZERO                     J
            .
SAVAREA1    DS          9D
            .
PAR1        DC          C'ABCDEF'
PAR2        DS          F
ZERO        DC          F'0'

Code in called routine:
SUBROUT     SAVE        (14,11)                     C
            BALR        . . .                       D
            USING       . . .                       D
            ST          13,SAVAREA2+4               E
            LA          13,SAVAREA2                 F
            .
        processing
            .
            L           13, SAVAREA2+4              G
            RETURN      (14,11)                     H
SAVAREA2    DS          9D                          I
            .
```

A Points to save area in calling program.
B Passes parameters PAR1, PAR2.
C Saves registers of calling program.
D Establishes addressability.
E Provides a backpointer to the calling program's save area.
F Points to new save area (for tracing 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 8-12. Use of CALL, SAVE, and RETURN Macros

program. Specifying register 15 preceded by a LOAD macro is most useful when the same program is called several times during

- you have to make sure that only one subtask is issuing an I/O request at a time and that only one I/O area is specified in the DTF.

- With the extended buffering support for 3800/3200 printers, no I/O should be attempted out of an asynchronous user exit, since termination of the task may occur.

Subtask Initiation

The maximum possible number of subtasks that can be initiated at any one time in the system is 208. Up to 31 subtasks can run concurrently within a partition, provided the overall limitation of 208 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. The subtask can also be a separate phase, in which case the phase must first be read into storage with the LOAD or CDLOAD macro before the ATTACH macro is issued.

Figure 8-13 on page 8-32 includes an example of attaching a subtask.

```
*  REGISTER EQUIVALENTS
          EJECT
R0        EQU   0
R1        EQU   1
R2        EQU   2
R3        EQU   3
R4        EQU   4
R5        EQU   5
R6        EQU   6
R7        EQU   7
R8        EQU   8
R9        EQU   9
R10       EQU   10
R11       EQU   11
R12       EQU   12
R13       EQU   13
R14       EQU   14
R15       EQU   15
          SPACE 2
*********************************************************************************************
*  DEFINITION AREA FOR MAINTASK
MSGEND    DC    CL80'MAIN NORMAL ENDED'
MSGABM    DC    CL80'MAIN ABNORMAL ENDED'
          DS    0D
MTSAVE    DS    16D           MAINTASK SAVE AREA
MTABSV    DS    9D            MAINTASK ABNORMAL END SAVE AREA
MTECB2    DC    F'0'          MAINTASK ECB FOR POST FROM SUBTASK2
ST1ECBM   DC    F'0'          SUBTASK1 ECB FOR POST FROM MAINTASK
          SPACE 2
*********************************************************************************************
*  FILE DEFINITION AREA
CONSIOA   DTFCN
                DEVADDR=SYSLOG,
                IOAREA1=CONINOUT,
                BLKSIZE=80,
                INPSIZE=80,
                TYPEFLE=CMBND,
                RECFORM=FIXUNB,
                WORKA=YES
CONINOUT  DC    CL80' '
*********************************************************************************************
*  OUTPUT ON CONSOLE ALPHANUMERIC
PUTCONS   EQU   *
          PUT   CONSIOA,(R5)
          BR    R10
          EJECT
```

Figure 8-13 (Part 2 of 7). Multitasking Sample Program

```
*********************************************************************************
*        ATTACH SUBTASK 2 (SEE NOTES *2 THROUGH *4 ABOVE)                       *
*********************************************************************************
ATTST2   EQU   *
         MVC   ST2SAVE(8),ST2NAME PROVIDE SUBTASK NAME IN ST-SAVE AREA
         ATTACH SUBTASK2,ECB=ST2ECB,SAVE=ST2SAVE,ABSAVE=ST2ABSV
         LTR   1,1              TEST IF ATTACH IS SUCCESSFUL
         BNM   ATTST20          BRANCH IF SUCCESSFUL
         WAIT  (1)              WAIT TO RETRY ATTACH
         B     ATTST2           BRANCH TO RETRY
ATTST20  BCTR  R0,R0            GET END OF SAVE AREA SUBTASK2 AND
         ST    R0,ST2SVEND      STORE THE END ADDRESS
         BR    R10              BRANCH AND LINK RETURN VIA REGISTER 10
ST2SAVE  DC    16D'0'           SAVE AREA SUBTASK2 WITH FLOAT REGS
ST2ABSV  DC    9D'0'            AB SAVE AREA SUBTASK2
ST2ECB   DC    F'0'             ECB SUBTASK2
ST2SVEND DC    F'0'             END ADDRESS SUBTASK2 SAVE AREA
ST2NAME  DC    CL8'SUBTASK2'    NAME OF SUBTASK2
         EJECT
*  ESTABLISH ABNORMAL END EXIT
ABXITM   EQU   *
         STXIT AB,ABEXIT,MTABSV,OPTION=DUMP
         BR    R10
*  ABNORMAL EXIT ROUTINE REFERENCED BY STXIT MACRO
ABEXIT   EQU   *
         BALR  R12,0
         USING *,R12            ESTABLISH ADDRESSABILITY TO ABASE PTR
         L     R12,ABASE
         USING BASEADDR,R12     ESTABLISH ADDRESSABILITY TO BASE ADDR
         USING ABSAVE,R1
         STC   R0,ABCODE  STORE LAST BYTE OF R0 ==> ABEND-CODE (SEE *5)
*5  STORES THE ABNORMAL END CODE FOR LATER USE. THIS ROUTINE IS SHARED
*5  BY THE MAINTASK AND THE SUBTASKS.
         C     R1,=A(ST1ABSV) SUBTASK1 ABNORMALLY ENDED?
         BE    ST1ABEND       IF YES GO TO ST1ABEND
         C     R1,=A(ST2ABSV) SUBTASK2 ABNORMALLY ENDED?
         BE    ST2ABEND       IF YES GO TO ST2ABEND
         EXIT  AB             MUST BE MAINTASK
         B     ABMAIN         CONTINUE MAINTASK PROCESSING
ST1ABEND EQU   *
         POST  ST2ECB
         JDUMP
         SPACE 2
```

Figure 8-13 (Part 4 of 7). Multitasking Sample Program

```
*********************************************************************************************
*           S U B T A S K  1                                                                *
*********************************************************************************************
          CNOP  0,4
SUBTASK1 EQU   *       SUBTASK1 = ENTRY POINT GIVEN BY ATTACH MACRO
          BALR  R3,0  (SEE *6)
          USING *,R3   (SEE *6)
*-----
*6  THE MAINTASK AND THE SUBTASK MAY USE DIFFERENT BASE REGISTERS.
*6  THIS IS NOT NECESSARY IF ADDRESSABILITY IS ENSURED BY
*6  THE MAINTASK'S BASE REGISTER (R12 IN THIS EXAMPLE).
*-----
          LA    R5,MSG101         ADDRESS OF ATTACH MESSAGE ==> R5
          BAL   R10,PUTCONS       OUTPUT OF MESSAGE TO CONSOLE
          WAIT  ST1ECBM           WAIT TO BE POSTED FROM MAINTASK
          MVI   ST1ECBM+2,X'00'   RESET ST1ECBM IN WAIT STATE
          POST  ST2ECB            RELEASE WAIT STATE OF SUBTASK2 ECB10
*
*   STATEMENTS OF YOUR PROGRAM
*
          WAIT  ST1ECB20          WAIT TO BE POSTED FROM SUBTASK2
          LA    R5,MSG102         ADDRESS OF SUBTASK1 END MESSAGE ==> R5
          BAL   R10,PUTCONS       OUTPUT OF END MESSAGE ON CONSOLE
          DETACH                  SUBTASK1 DETACHES ITSELF AND POSTS ST1ECB
          SPACE
          DS    0F                SET FULLWORD BOUNDARY
ST1ECB20 DC     F'0'              ECB TO BE POSTED IN SUBTASK1
ESUB1    DC     X'00'             END OF SUBTASK1 INDICATOR
MSG101   DC     CL80'SUBTASK1 ATTACHED'
MSG102   DC     CL80'END OF SUBTASK1 REACHED'
          EJECT
```

Figure 8-13 (Part 6 of 7). Multitasking Sample Program

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. This save area is specified in the SAVE=savearea operand of the ATTACH macro. 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 (ABSAVE=absavearea) 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.

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.

Specifying an Event Control Block

In the ATTACH macro, the name of an event control block (ECB) can be specified (ECB=ecbname). The ECB is a fullword whose format is shown in Figure 8-14 on page 8-40. The ECB operand is required if other tasks can be affected by the subtask's termination, or if resources are controlled by ENQ and DEQ macros within the subtask.

Note: Do not post (with the POST macro) the ECB specified by the ATTACH macro, because any task which is waiting on this ECB for subtask termination will be set ready-to-run, although the subtask has not yet terminated. For inter-task synchronization introduce a second ECB which can then be used by the POST macro.

Intertask Communication

Tasks communicate with each other through the event control blocks (ECBs) described above. A task sets itself into the wait state by issuing a WAIT macro with an ECB specified. To release that task from the wait state, another task issues a POST macro with the same ECB specified.

The task that issues the WAIT macro remains in the wait state until the designated ECB gets posted by the POST macro, that is: bit 0 of byte 2 is 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 for one of a number of events to occur (WAITM), register 1 points to the posted ECB. This allows the task to test 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 8-15 on page 8-42, 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 successful completion of SUBTASK1, which is indicated by POST ECBPREF. If the subtask is terminated before it can finish its processing, the supervisor posts the ECB defined via ATTACH...,ECB=ECBSEC, which is the secondary event. With either event, the address of the posted ECB is in register 1 after the WAITM macro has been issued. With this address you can, for example, 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   ST1ECB
```

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 ST1ECB+2,X'00' so that tasks testing that ECB can be reset into the wait state.

Figure 8-16 on page 8-44 illustrates the use of the POST macro. The example shows three subtasks: SUBTASK1, SUBTASK2, and SUBTASK3. SUBTASK1 depends on input which can be supplied by SUBTASK2 or SUBTASK3 and, therefore, issues a WAITM macro on the ECBs for those subtasks.

Initially, SUBTASK1 is placed into the wait state by the WAITM macro. Control then passes to SUBTASK2 and then to SUBTASK3. When either of the two subtasks has the necessary data for SUBTASK1, it posts its ECB that removes SUBTASK1 from the wait state. When SUBTASK1 finishes processing, it posts its ECB, thus causing the main task to be taken out of the wait state. The main task can then detach SUBTASK1.

```
SUBTASK2    EQU     *
            .
            .
ST2A        EQU     *
            .
            .
            POST    ST2ECB          POST ECB for subtask 1
            .
            .
            B       ST2A
            .
            .
SUBTASK3    EQU     *
            .
            .
ST3A        EQU     *
            .
            .
            POST    ST3ECB          Post ECB for subtask 1
            .
            .
            B       ST3A
            .
            .
MTSVADR1    DC      F'0'            Save area address for
                                    main task
ECB1A       DC      F'0'            Dummy ECB for subtask 1
ST1ECB      DC      F'0'            ECB for subtask 1
ST2ECB      DC      F'0'            ECB for subtask 2
ST3ECB      DC      F'0'            ECB for subtask 3
```

Figure 8-16 (Part 2 of 2). Use of the POST Macro

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, DUMP or JDUMP macro, or the RETURN macro if RETURN=YES is specified in the ATTACH 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

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 it. The task with the highest priority obtains control. If no other tasks are waiting for the RCB, control returns to the dequeueing task.

The following figures show examples of the use of the ENQ, DEQ, and RCB macros and the resource control block.

Figure 8-18 on page 8-49 shows a main task with two subtasks sharing the same resource and protecting it 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 8-19 on page 8-50, 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 8-20 on page 8-51, 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.

```
MAINTASK   BALR    12,0
           USING   *,12
           .
           .
SUBTASK1   EQU     *
           .
           .
SBTASK1A   ENQ     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 8-18. Sharing a Resource in a Common Subroutine

```
MAINTASK  START 0
          .
          .
          ATTACH SUBTASK1,SAVE=ST1SAVE,ECB=ST1ECB
          .
          .
          ATTACH SUBTASK2,SAVE=ST2SAVE,ECB=ST2ECB
          .
          .
SUBTASK1  EQU     *
          .
          .
          ENQ     RCBA            Protect resource RESRCA
          .
          .
* Update RESRCA
          .
          .
          DEQ     RCBA            Release resource RESRCA
          .
          .
SUBTASK2  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 8-20. 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.

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 8-21 on page 8-54 illustrates the two occurrences of the UNLOCK macro.

DASD Record Protection (Track Hold)

When a record is being modified by one task, it must be prevented from being accessed by another task. For a CKD device, the data transmission unit is one block; for an FBA device, this unit is an integral 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 or DTFDA macros. In these cases, DASD record protection can be obtained within the entire system if the TRKHLD operand of the FOPT 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 three specific situations:

1. Updating DTFSD data files.

2. Updating DTFSD work files.

3. Processing DTFDA 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

For DTFDA files using WRITE or WRITE AFTER, LIOCS 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, LIOCS holds the track but does not free it automatically. This must be done in the user program by the FREE macro.

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.

```
MAINTASK   START    0
           .
           .
           ATTACH   SUBTASK1,SAVE=ST1SAVE,ECB=ST1ECB
           .
           .
           ATTACH   SUBTASK2,SAVE=ST2SAVE,ECB=ST2ECB
           .
           .
SUBTASK1   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,...
           .
           .
SUBTASK2   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
```

Figure 8-22 (Part 1 of 2). Using the Track Hold Facility

tasks may attempt to use a single non-entrant 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.

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, 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 a sublibrary 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 has been changed by means of this macro retains its 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;

```
          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
          LTR    15,15
          BZ     CONTINUE
          B      CANCL
CONTINUE  .
          .
          .
          EOJ
CANCL     CANCEL ALL
```

Figure 8-23. Example for Loading an Alternate FCB

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 by using the SUBSID INQUIRY macro. The macro retrieves a byte string, which can be interpreted using the mapping DSECT generated by the MAPSSID macro (for details, see the Macro Reference manual). 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. 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.

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.

PROGRAM, DTF, AND LOGIC MODULE ASSEMBLED SEPARATELY

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

Supplying the SEPASMB=YES operand in a DTF macro causes a CATALOG command 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 CATALOG command 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.

Cataloging DTFs and Modules

Considerable coding effort is saved if logic modules are cataloged in a sublibrary. The same applies to DTFs. Using cataloged DTFs requires that you name the DTFs, and then use these names in all references your program makes to the DTFs. If you name the 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. The linkage editor can perform an autolink only if there is an exact match of module names specified in the DTF and the names of the modules themselves.

If, during installation 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

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 on 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.

Finally, an FBA DASD example is shown with a DTF assembled together with the program and a pre-assembled logic module.

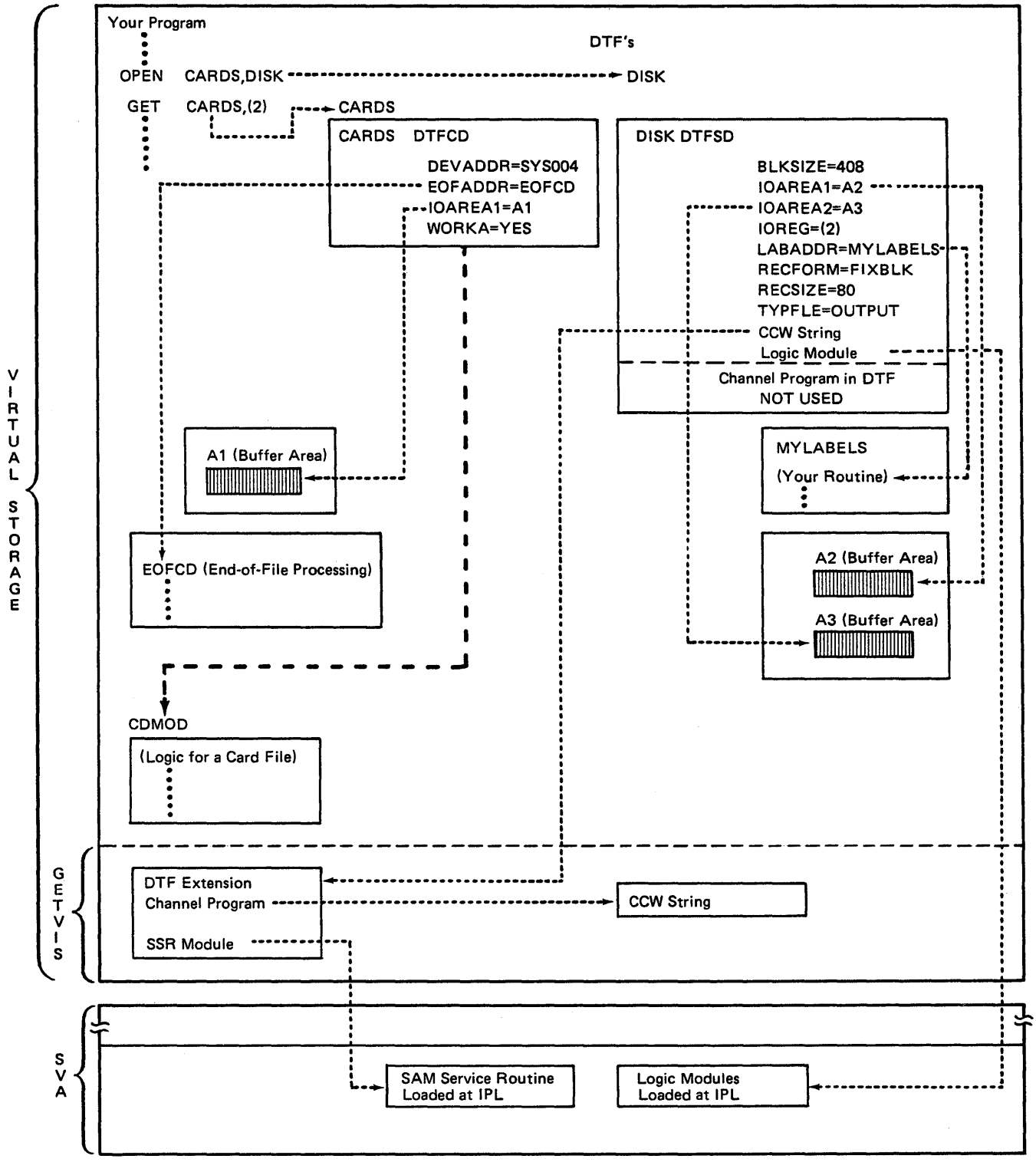


Figure A-1. Assembling Your Programs, DTFs, and Logic Modules Together (Example 1)

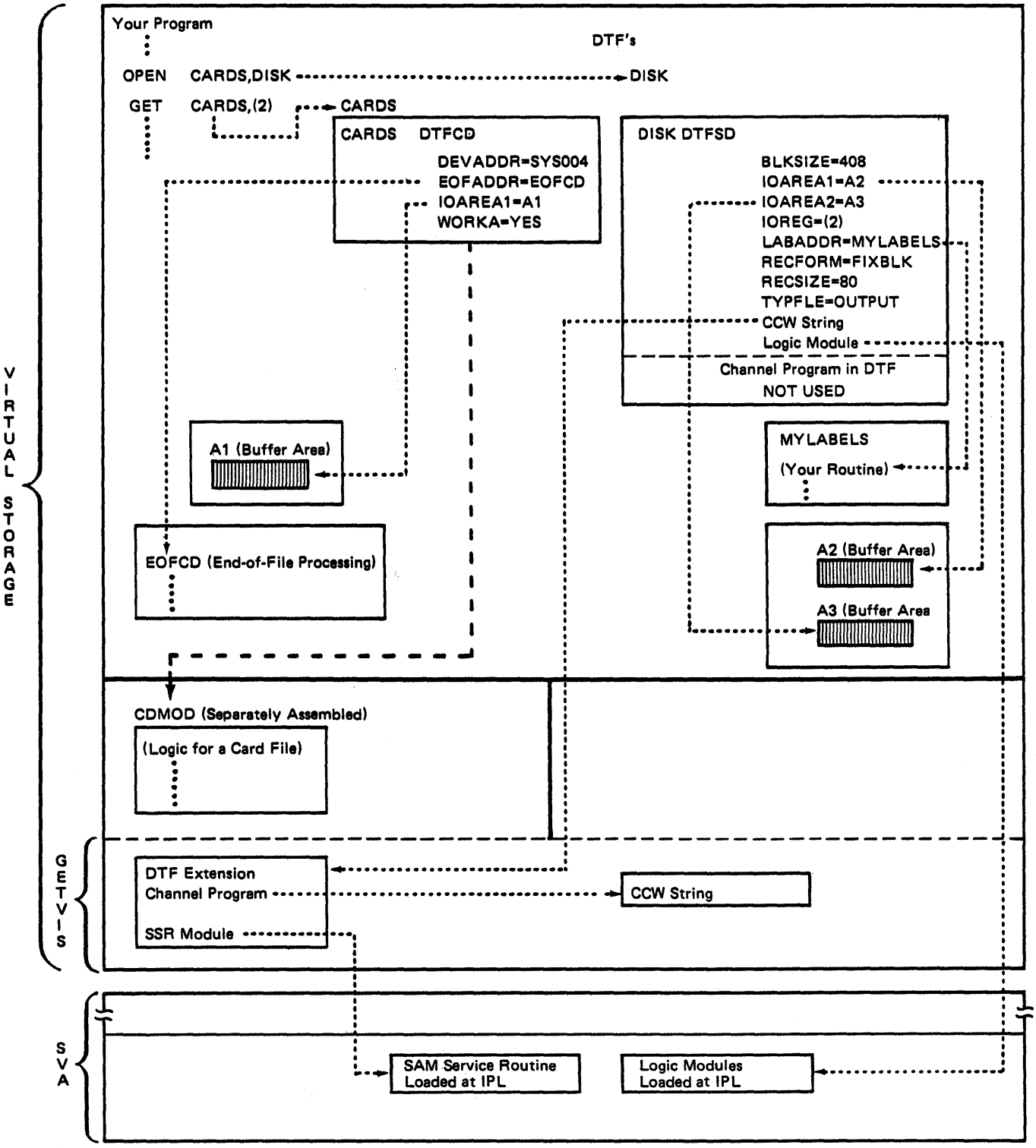


Figure A-2. Logic Modules Assembled Separately (Example 2)

```
                                                    Column 72
DISK      DTFSD                                          X
                  BLKSIZE=408,                           X
                  SEPASMB=YES,                           X
                  IOAREA1=A2,                            X
                  IOAREA2=A3,                            X
                  IOREG=(2),                             X
                  LABADDR=MYLABELS,                      X
                  RECFORM=FIXBLK,                        X
                  RECSIZE=80,                            X
                  DEVICE=3330,                           X
                  TYPEFLE=OUTPUT
MYLABELS  BALR    10,0
          USING   *,10
          .
          .
          LBRET   2
A2        DS      408C                  (4)
A3        DS      408C                  (5)
          END
```

In the card-file and the disk-file assemblies, a USING statement was added because certain routines are separated from the main program and moved into the DTF assembly.

When your routines, such as error, label processing, or EOF routines, are separated from the main program, it is necessary to establish addressability for these routines. You can provide this addressability by assigning and initializing 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 separate your error routines, it is your responsibility to establish addressability for them.

Figure A-4 on page A-12 contains the printer output to show how the coding for Example 3 would look when assembled.

In Figure A-4 on page A-12, 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 CATALOG command. The two object decks can be cataloged ,together with the logic modules, as follows into a sublibrary

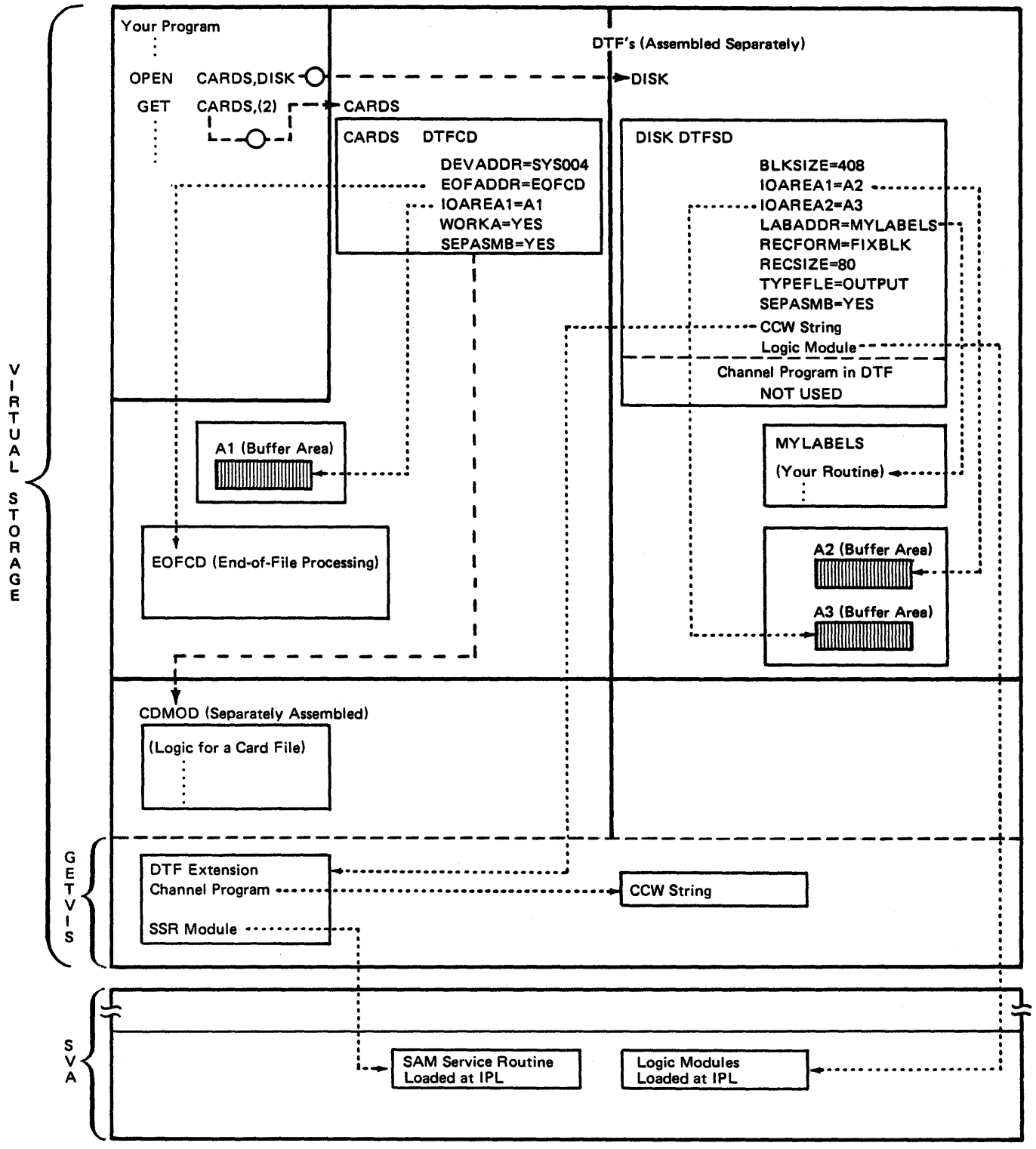


Figure A-3. Logic Modules and DTFs Assembled Separately (Example 3)

```
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). |
IJCFZIWJ  ER  02                        External reference.  Corresponds to V-type address constant generated in DTFCD.
DISK      ER  03                        External reference.  Defined by EXTRN statement.
```

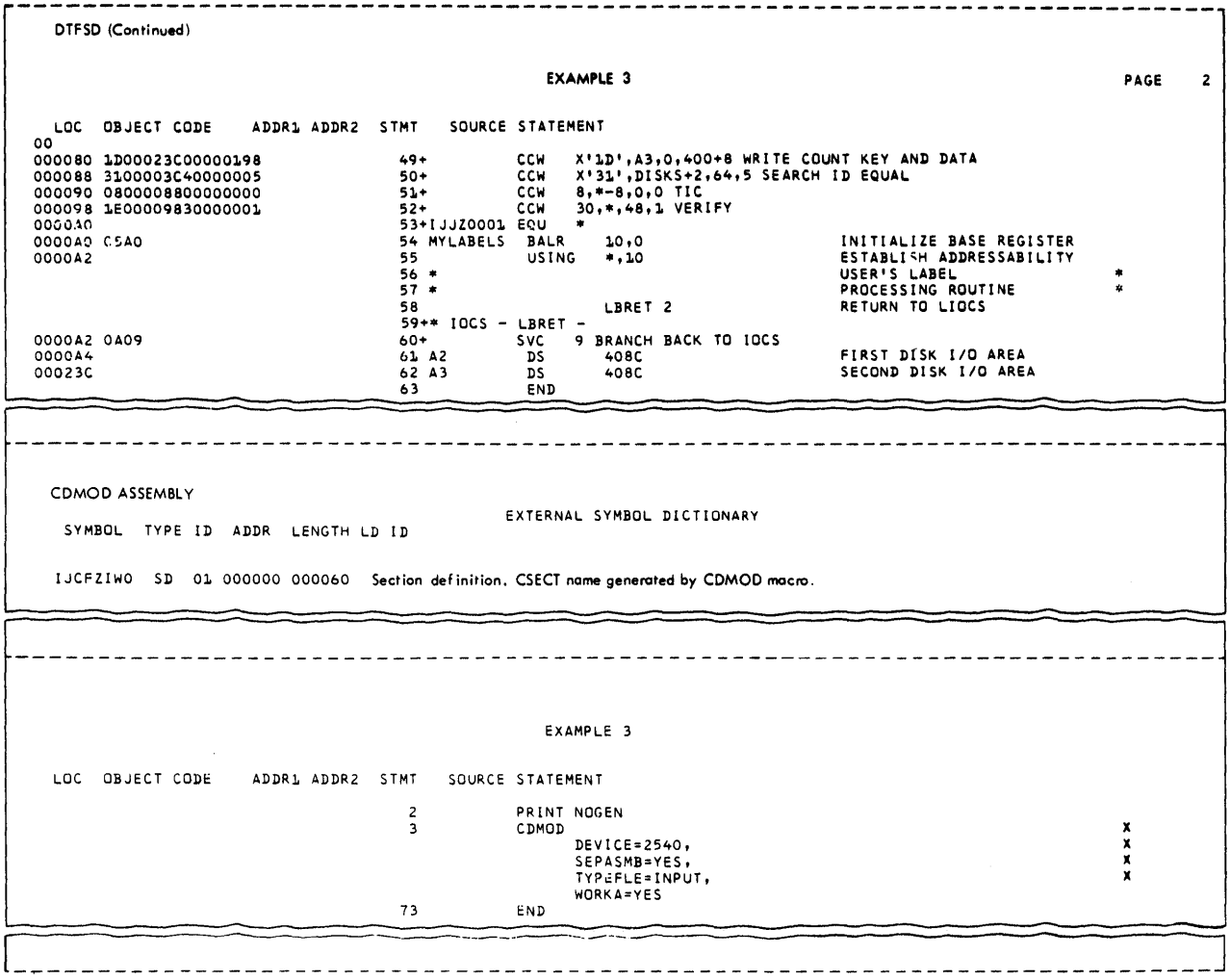
```
                                                                              PAGE    1

                                        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(IJCFZIW0) 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 A-4 (Part 2 of 4). Separate Assemblies (Example 3)



```
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

IJCFZIW0   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 A-4 (Part 4 of 4). Separate Assemblies (Example 3)

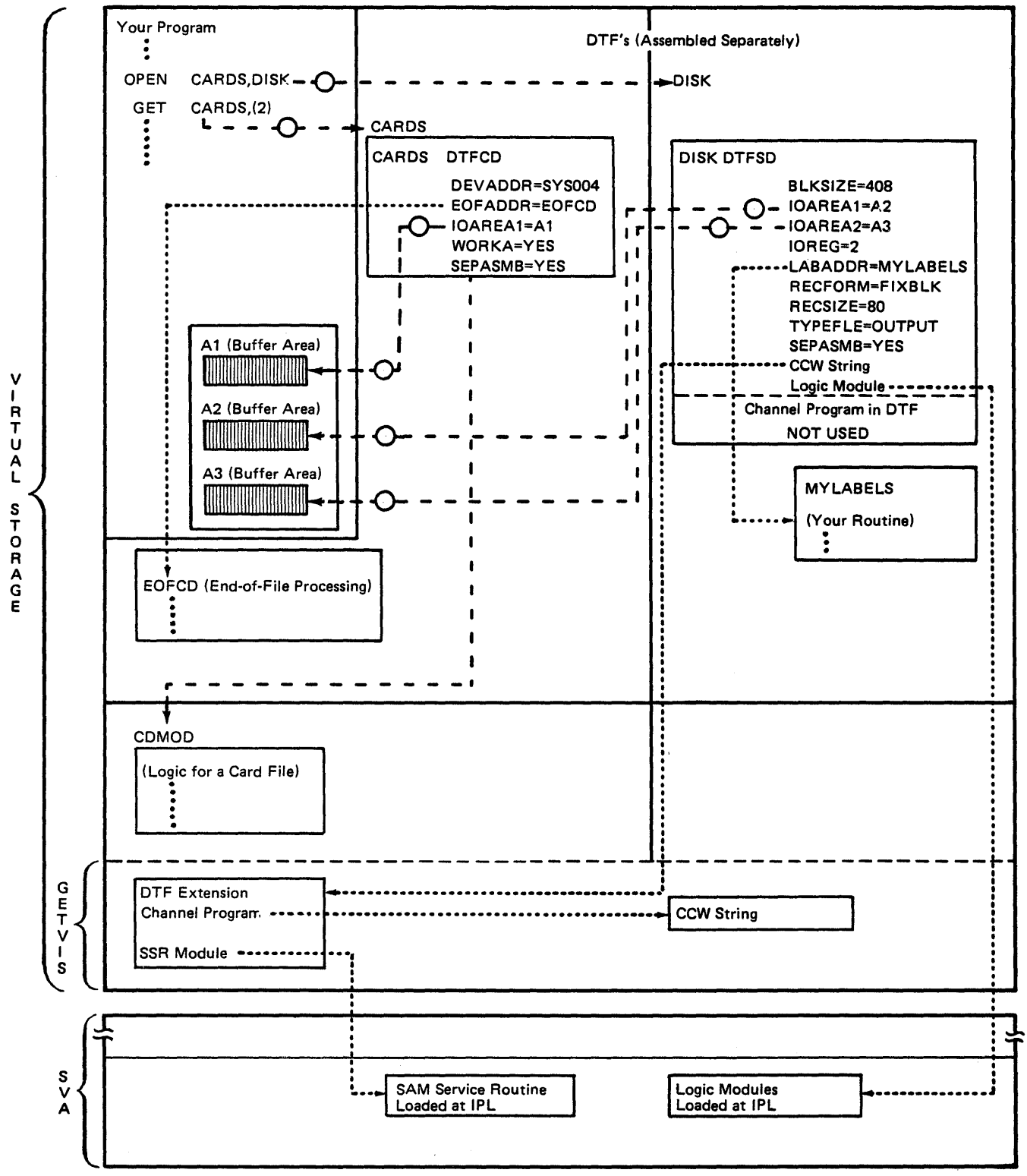


Figure A-5. Logic Modules and DTFs Assembled Separately, I/O Areas with Main Program (Example 4)

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
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: Separates 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 a sublibrary 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 a sublibrary.

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 method 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.

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.

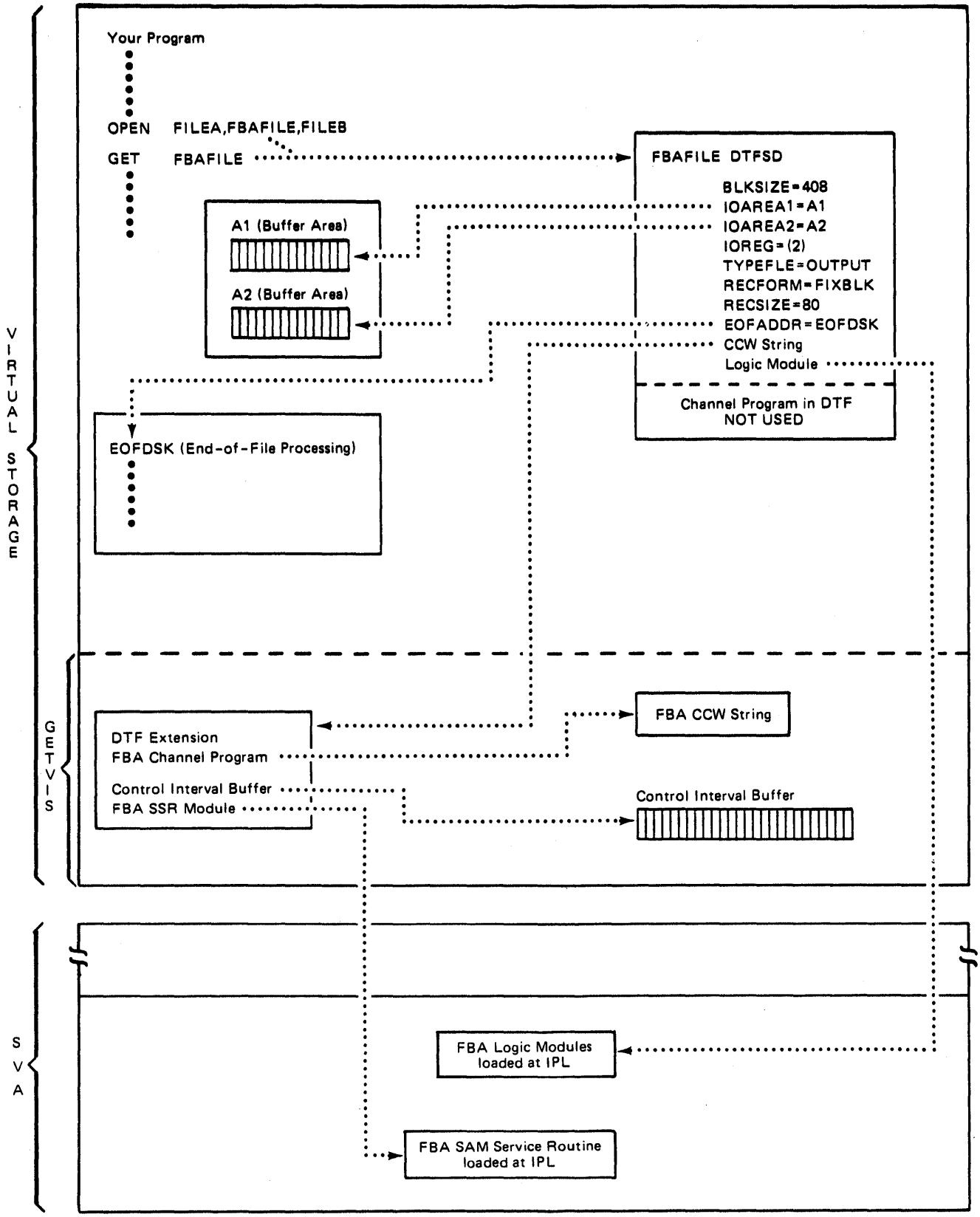


Figure A-7. DTFs and Logic Modules Assembled Jointly With FBA DASD Support Included (Example 6)

APPENDIX B. DIRECT ACCESS METHOD (DAM)

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.

With the Direct Access Method you can process records in random order. The records may be read, written, updated or 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
IBM 3375 Direct Access Storage
IBM 3380 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.

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.

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.

register (specified by the DTFDA RECSIZE operand) before issuing the WRITE macro for that record.

I/O Area Specification

The DTFDA IOAREA1 operand defines 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 B-1 on page B-4 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

Figure B-1 on page B-4 and Figure B-2 on page B-4 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 shown in these figures 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 B-1 on page B-4, 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 operand of the DTFDA macro, the key is transmitted to or from the field you specified in the KEYARG operand. The count field, if desired, is taken from an area reserved automatically by logical IOCS.

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. In that case you must 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 B-3 on page B-5.

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
3375 3380	35,616 47,476	32767 32767

Figure B-3. 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 non-formatting 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.

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) 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 B-5 on page B-9. For reference to records by record number,

location to another. In such cases the relative addressing scheme remains the same, and the actual addresses are automatically converted by IOCS.

Bytes	Decimal Identifier	Contents in Zoned Decimal	Information
0-7	tttttttt	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.

Figure B-5 (Part 1 of 2). Types of Track Reference Fields

ttt or tttttttt 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 ttttttttrr.

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.

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 requesters having the type of file for which the logic module was generated. Any other logic module referenced by the DTF will be ignored.

PROCESSING FILES WITH DAM

After the DAM files are defined with the DTFDA macro, the imperative macros are used to operate on the files. These macros are divided into three groups: those for initialization, processing, and termination.

Initialization

The OPEN macro 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. This association remains in effect throughout processing of the file until you issue a CLOSE macro.

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 DTF byte 16, bit 2, to indicate the file is not activated. No input/output operations should 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 and gives control to your extent routine. You can save this information for use in specifying record addresses. Then the next volume is opened (on an input file, only after the requested user labels are written). When all volumes are open, the file is ready for processing. If the DASD device is file-protected, all extents specified in EXTENT statements are available for use.

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 (creating) a direct access file and processing it (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 Facilities

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 Facilities program cannot clear a portion of a volume, you can do this by writing a complete file consisting 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.

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.

Variable-Length Blocks With a Key Area: The file should not be preformatted with the Clear Disk utility program. The Device Support Facilities 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 if you want to retrieve them later.

Since records cannot be stored in the space occupied by deleted records, the cylinder overflow areas themselves may also overflow. 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 can 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.

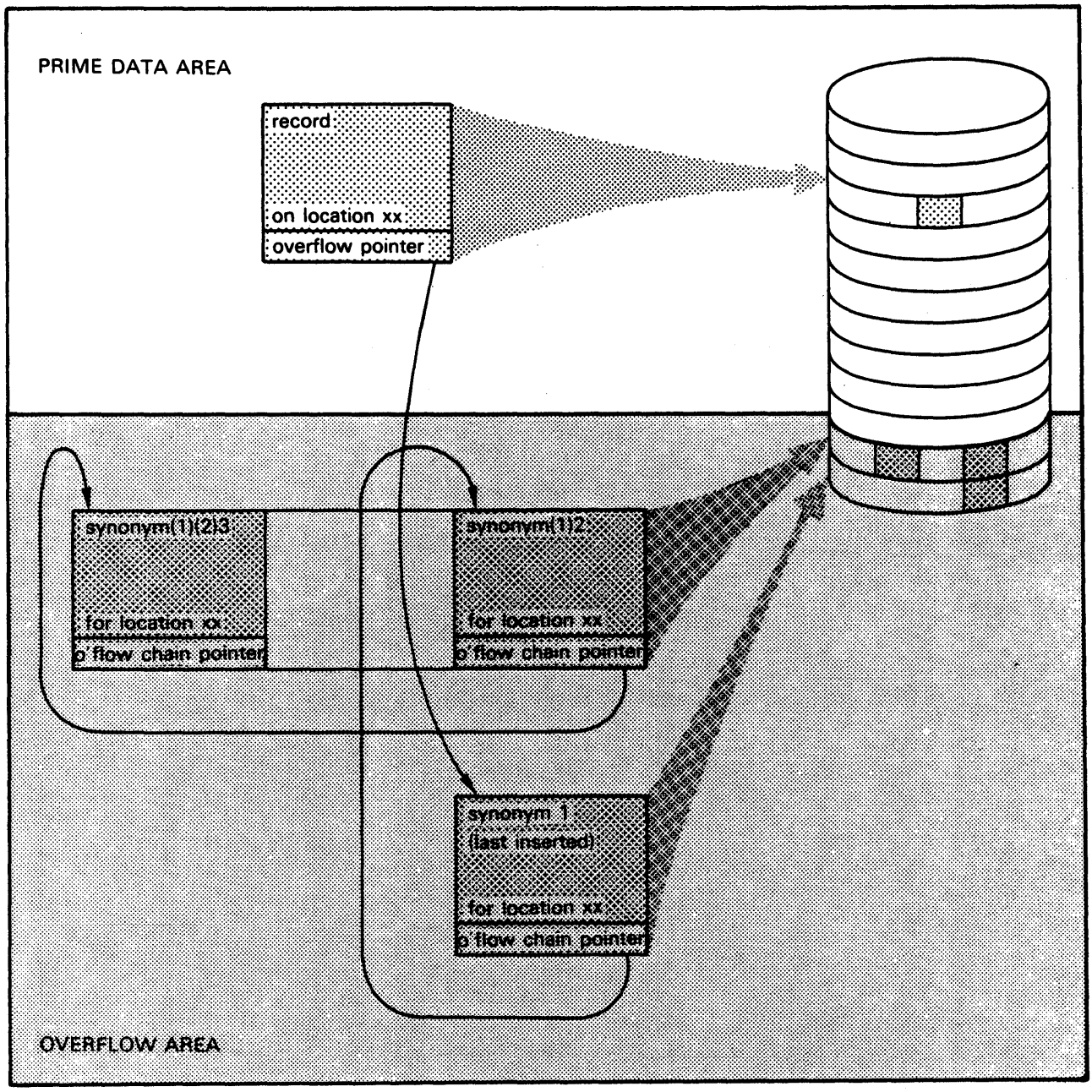


Figure B-6. Prime Data Record and Related Overflow Records

the prime data block to its original location on disk and write the new record as the first overflow record.

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

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 B-7 on page B-22 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 B-7 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 here is presented in Figure B-8 on page B-23.

Reading Records

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

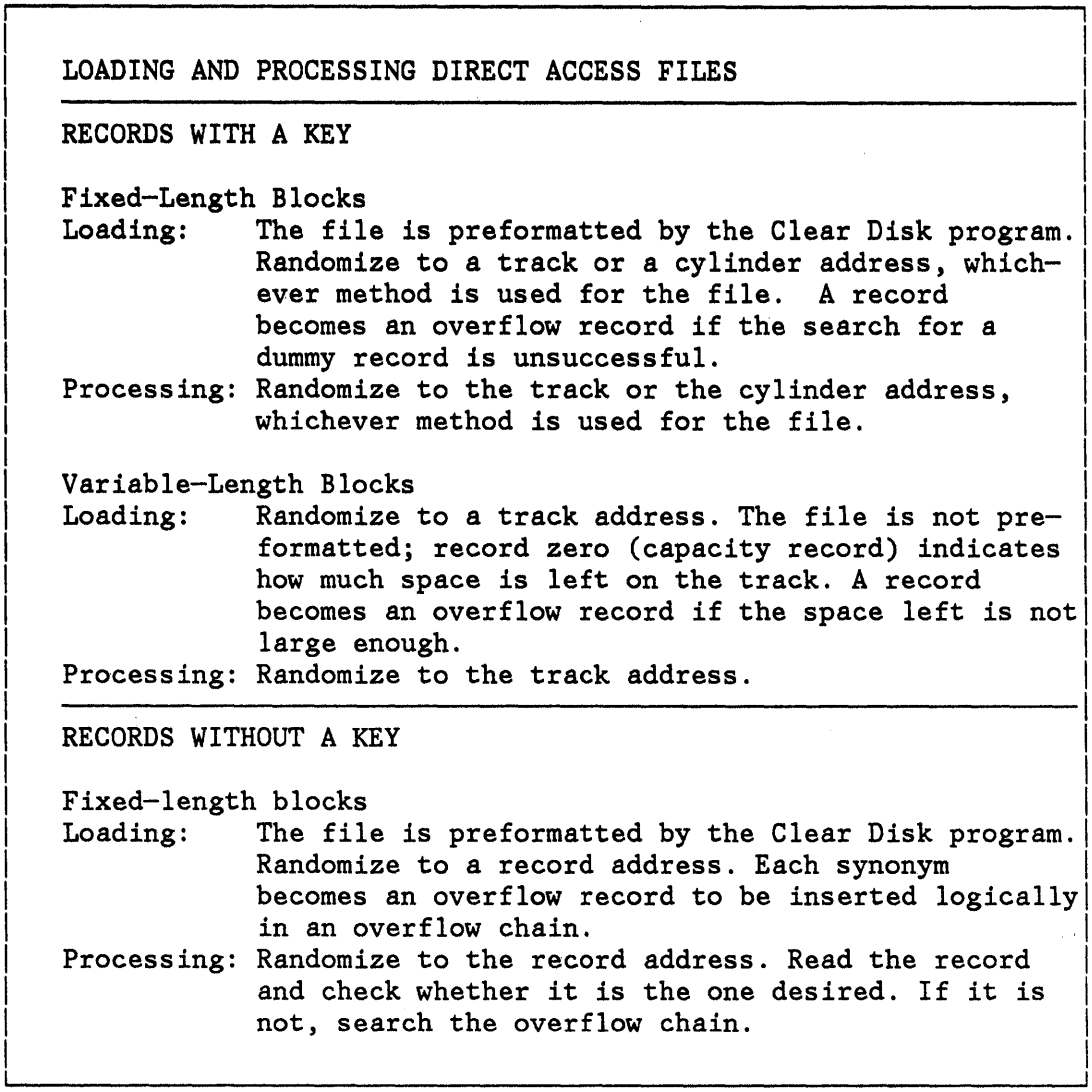


Figure B-8. Summary of Randomizing Methods

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.

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
```

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 one 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 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

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 until 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

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.

Byte	Bit	Error/Status Code Indication	Explanation
0	1	Wrong-length record (continued)	• 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 when: • 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.

Figure B-9 (Part 2 of 8). ERRBYTE Error Status Indication Bits

Byte	Bit	Error/Status Code Indication	Explanation
0	1	Wrong-length record (continued)	• A non-formatting 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 is also 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. • 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 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. • A non-formatting 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.

Figure B-9 (Part 4 of 8). ERRBYTE Error Status Indication Bits

Byte	Bit	Error/Status Code Indication	Explanation
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. This bit is also turned on if IDLOC=name has been specified and the record to be read or written is the last record of the cylinder. However, the address returned is that of the first record of the next cylinder.
1	3	Data check when reading key or data	This is an unrecoverable error.

Figure B-9 (Part 6 of 8). ERRBYTE Error Status Indication Bits

Byte	Bit	Error/Status Code Indication	Explanation
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 for 3375: cylinder 945, head 11 for 3380: cylinder 884, head 14 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 B-9 (Part 8 of 8). ERRBYTE Error Status Indication Bits

APPENDIX C. 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 macro. No logic module generation macro is needed. The DTFPH macro must also be used for a checkpoint file on disk.

In physical IOCS, the logical 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 C-1 shows the relationship between the source program and the job control I/O assignment.

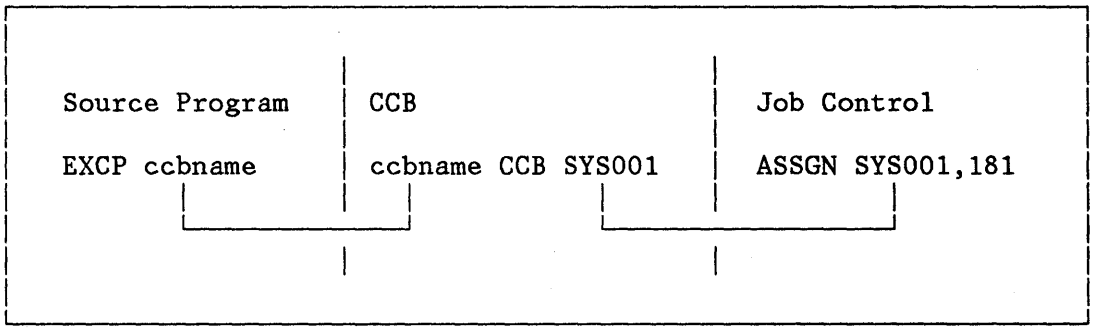


Figure C-1. Relationship Between Source Program and Job Control I/O Assignment

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 termination.

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,

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 an output file is created 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 statements (for example, it checks that the extent limit address for the device being opened is valid). OPEN also checks the standard VOL1 label and 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.

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 the Macro Reference manual 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

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 C-2 on page C-8.

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 an 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, which is discussed in the following section.

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,

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.

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.

```
filename  DTFPH  (parameters, among others SYSxxx and ccbname)
            .
            .
            .
          OPEN    filename
            .
          EXCP    filename
          WAIT    filename
            .
            .
          CLOSE   filename
            .
            .
ccbname   CCB     SYSxxx,ccwname,optional operands
            .
ccwname   CCW     cc,data-addr,flags,count
          CCW     cc,data-addr,flags,count

          etc.
```

Figure C-3. Relationship Between DTFPH and Other PIOCS Macros

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 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

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 cancelation 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 C-4), 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.

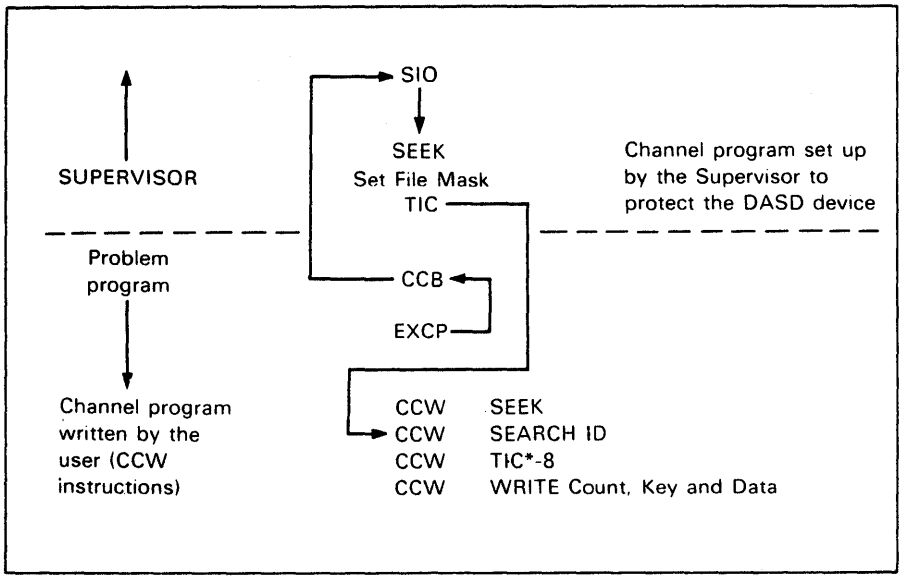


Figure C-4. Example of Channel Programming a File-Protected CDK DASD File

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.

APPENDIX D. 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      16D
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 requires it.

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:

```
                                        Column 72

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 the Macro Reference manual), 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:

```
                                        Column 72

ATTACH  SUBTASK,                                X
        SAVE=(S,DYNSAV)                         X
        ECB=(S,DYNECB)                          X
        ABSAVE=(S,DYNSAVAB)                     X
        MFG=(S,DYNPARM)
```

APPENDIX E. 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:

```
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 E-1 on page E-4 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.

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      ADCON
          .
          .
ADCON     DC      A(ADCONAME)
```

Technique 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 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 relocation
                ST      4,ADCON         factor
                .
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 E-2 on page E-8 shows an example of a self-relocating program.

```
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 CCBs TO FULLWORD
PRINTCCB    CCB     SYS004,PRINTCCW,X'0400'
TAPECCB     CCB     SYS001,TAPECCW
PRINTCCW    CCW     PRINT,OUTAREA,SLI,L'OUTAREA
TAPECCW     CCB     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 E-2 (Part 2 of 2). Self-Relocating Sample Program

APPENDIX F. 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 listed in Figure F-1 on page F-2, the stacker selection codes in Figure F-2 on page F-3.

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

Figure F-2. Stacker Selection Codes

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. The control characters are listed in Figure F-3 on page F-4.

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

Figure F-3 (Part 2 of 2). Printer Control Codes

INDEX

M

N

O

V

W

Numerics

VSE/Advanced Functions
Application Programming
Macro User's Guide
Order No. SC33-6196-1

READER'S COMMENT FORM

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 comments 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 is 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).

IBM

SC33-6196-1