GC33-5371-6
File No. S370-34

Systems

DOS/VS
System Management Guide

Release 34

IBM

GC33-5371-6
File No. S370-34

Systems

DOS/VS
System Management Guide

Release 34

IBM

Summary of Amendments

Release 34

Edition GC33-5371-6 documents:

- Full support of
  - IBM 3350 Direct Access Storage (DOS/VS previously supported the device only in 3330-11 compatibility mode).
  - IBM 3330-11.
- Common device class for 3211-compatible printers (IBM 3211 and IBM 3203-4).
- Support of IBM 3277 Display Station as Operator Console.
- Support of IBM 3540 Diskette Unit as an IPL communication device.
- Improvement of initial program load through the use of an IPL communication device list.
- Support for dynamically changing the blocking factor for a sequential disk file through the job control DLBL statement.
- Inclusion of the functions of the COPYSERV program into the CORGZ program and removal of the COPYSERV program for DOS/VS.
- Integration of support information on SYSTEM/370 CPU Models 135-3, 138, 145-3, and 148 and on the IBM 3203-4 printer.

In addition, corrections and editorial changes have been made to improve the manual's usability, and POWER/VS information has been removed.

Release 33

Edition GC33-5371-5 documents:

- Second label information cylinder for the IBM 3340
- POWER/VS enhancements
- Installation improvements
- Cardless system support
- Extended timer services

Release 32

Technical Newsletter GN33-8801 includes information on cross-partition event control and the fast CCW translation (FASTTR) option, as well as miscellaneous corrections and updates.

Seventh Edition (April, 1977)

This is a major revision of, and obsoletes, GC33-5371-5. This edition applies to Version 5, Release 34, of the IBM Disk Operating System/Virtual Storage, DOS/VS, and to all subsequent versions and releases until otherwise indicated in new editions or Technical Newsletters.

Changes and additions to the text or illustrations are indicated by a vertical line to the left of the changes. Changes are continually made to the information herein; before using this publication in connection with operation of IBM systems, consult the latest *IBM System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Laboratories, Publications Department, Schoenaicher Str. 220, 7030 Boeblingen, Germany. Comments become the property of IBM.

THIS MANUAL. . .

. . . is a guide to the IBM Disk Operating System/Virtual Storage (DOS/VS). The system in its entirety is discussed on a conceptual and functional level. System management refers not only to the way DOS/VS is organized, but also to the way the user can efficiently manage the system facilities at his disposal. This manual, therefore, does more than describe the functions and interaction of the system control and system service programs that constitute DOS/VS. It also describes how you -- as a systems planner, systems programmer, applications programmer, or operator -- can use DOS/VS to your best advantage.

Before you begin reading this manual, you should be familiar with the information contained in the *Introduction to DOS/VS,* GC33-5370.

This book is not a guide to data management; instead, a separate manual is provided for this purpose, called the *DOS/VS Data Management Guide,* GC33-5372.

A manual that complements both the *DOS/VS System Management Guide* and the *DOS/VS Data Management Guide* is also available at this time to meet your installation's planning requirements. It is called *DOS/VS Supervisor and I/O Macros,* GC33-5373.

After reading the above mentioned manuals, you should be able to turn directly to the DOS/VS library of reference manuals in order to work with your operating system. A reference manual is organized so that you can easily retrieve specific information on the formats of the control statements, macro instructions, labels, and messages, which you deal with daily.

This manual is divided into three parts:

Part I: The Organization of DOS/VS provides conceptual, descriptive, and planning information. Part I contains three chapters. The first chapter introduces the concepts of several of the main topics discussed throughout this part of the manual. The second chapter summarizes the standard and optional features of DOS/VS. The third chapter includes planning information for system generation.

Part II: Using the System provides the information on how to use the system. Part II contains five chapters, which consist of guidance information on using the IPL, job control, linkage editor, and librarian programs.

Part III: Designing Programs provides guidance in designing programs to be run under DOS/VS. Part III contains three chapters, which discuss how to design a program for execution in virtual mode, how to use the facilities of DOS/VS, and how to use the multitasking macros.

For reference purposes the organization of the system residence disk file (SYSRES) is shown in Appendix A.

The following IBM manuals are referred to in the text of this manual:

Introduction to DOS/VS	GC33-5370
DOS/VS Data Management Guide	GC33-5372
DOS/VS Supervisor and I/O Macros	GC33-5373
DOS/VS Tape Labels	GC33-5374
DOS/VS DASD Labels	GC33-5375
DOS/VS System Control Statements	GC33-5376
DOS/VS System Generation	GC33-5377
DOS/VS Operating Procedures	GC33-5378
DOS/VS Messages	GC33-5379
DOS/VS Serviceability Aids and Debugging Procedures	GC33-5380
DOS/VS System Utilities	GC33-5381
1401/1440/1460 DOS/VS Emulator on System/370	GC33-5384
1410/7010 DOS/VS Emulator on System/370	GC33-5385
Model 20 DOS/VS Emulator on System/370	GC33-5388
Guide to the DOS/VS Assembler	GC33-4024
DOS/VS VTAM System Programmer's Guide	GC27-6957
IBM System/370 Principles of Operation	GA22-7000
DOS/VS Supervisor Logic	SY33-8551
DOS/VS Librarian Logic	SY33-8557
DOS/VS Access Method Services User's Guide	GC33-5382
DOS/VS POWER/VS Installation Guide and Reference	GC33-6048

Table of Contents

Part I: The Organization of DOS/VS

Part II: Using the System

List of Figures

Chapter 1: Understanding the System

Chapter 3: Planning the System

Chapter 4: Starting the System

Chapter 5: Controlling Jobs

Chapter 6: Linking Programs

Chapter 7: Using the Libraries

Chapter 9: Designing Programs for Virtual-Mode Execution

Chapter 10: Using the Facilities and Options of the Supervisor

Appendix A: System Layout on Disk

Part I: The Organization of DOS/VS

Part I introduces DOS/VS. DOS/VS is a complex combination of programs that interact with user programs running on a System/370 central processing unit. The main features of DOS/VS, what the supervisor does for you, and how you tailor the system are presented in this part in three chapters:

Chapter 1: Understanding the System presents all readers with a description of the key features of DOS/VS, in particular the concepts of multiprogramming, virtual storage, and multitasking.

Chapter 2: Summary of DOS/VS Features lists the standard and optional features of DOS/VS.

Chapter 3: Planning the System is of particular interest to system programmers. This chapter includes three topics: system generation, supervisor generation, and planning the libraries.

Chapter 1: Understanding the System

This chapter introduces and describes the major concepts of DOS/VS. After reading this information, you will have gained an understanding of the principles on which DOS/VS operates. You will also be familiar with many of the terms that are used throughout the manual.

The main topics described in this chapter are:

- Multiprogramming
- Virtual storage
- Multitasking

Multiprogramming

Multiprogramming is a technique that allows the concurrent execution of more than one program in a single computer system. Multiprogramming balances the difference between the speed of the central processing unit (CPU) and the relatively slower speed of the I/O devices, and thereby improves the overall throughput of the system.

When a single executing program requests an I/O operation, it may not be able to continue with any useful processing until the I/O request has been satisfied. During this time, the CPU stands idle. With multiprogramming the CPU is used more efficiently. When one program stops processing, the CPU is put at the disposal of another program.

A program is said to be *in control of the system* when its instructions are being executed by the CPU. A program can voluntarily yield control of the CPU, or control can be withdrawn from it.

Programs that share the use of the CPU in multiprogramming do not have an equal claim on the CPU. Instead, one program is given a greater priority than another.

When a program must wait for a given event to occur before it can continue processing, it yields control of the CPU. The supervisor then passes control to a program of lower priority. Conversely, the supervisor withdraws control from a program whenever a program with higher priority is ready to resume processing. This generally happens when the I/O operation for which the program has been waiting is now completed.

Multiprogramming, therefore, allows the I/O operations of one program to be overlapped by the processing of other programs. When a program has to wait for the completion of an I/O operation, the supervisor sets the program in the wait state and selects another program for execution on the basis of its priority and readiness to run. This process is called task selection.

Efficient use of the system relates not only to the degree of CPU activity but also to storage management. During system generation, storage may be allocated to partitions to accommodate the programs that will be executed in them. At times, only a portion of the partition is used by the program being executed. Some programs require a large partition. DOS/VS can automatically balance the storage demands made by programs by making processor storage not being used by one program available to a program in another partition as required.

This storage management, which was not present in earlier versions of DOS, is not inherent to multiprogramming, but is implemented by certain virtual storage functions. It is described in more detail in the section *Virtual Storage,* later in this chapter.

Partitions

DOS/VS can support two or more partitions (depending on the number generated) in each of which a problem program can be executed. The number of partitions supported equals the number of problem programs that can be executed concurrently within the system. The actual number of partitions in a particular configuration is a supervisor generation option, and as such is described in the section *Tailoring the Supervisor* in *Chapter 3: Planning the System.*

Each program gets the priority associated with the partition in which it is executed. Priorities are assigned to partitions during supervisor generation, but may be altered by an operator command during processing to accelerate the execution of a particular program.

In any particular configuration, there is always one background (BG) partition; optionally, there can also be one or more foreground (FGx) partitions. The number of foreground partitions that can be specified is four (see Figure 1.1).

The background partition differs from the foreground partitions in the following respects:

- The background partition is automatically activated by IPL. A foreground partition must be activated via the BATCH or START operator command. (The BATCH and START operator commands are discussed in detail in *DOS/VS Operating Procedures.*)
- Certain IBM-supplied programs can be executed only in the background partition. These programs are OLTEP, discussed under Tailoring the Supervisor; CORGZ (merging into SYSRES functions); and MAINT (except deleting, renaming and condensing functions for a private core image library). Refer to the chapter *Using the Libraries.*
- To link-edit in a foreground partition, a private core image library must be assigned to that partition. To link-edit in the background partition, no private core image library need be assigned.

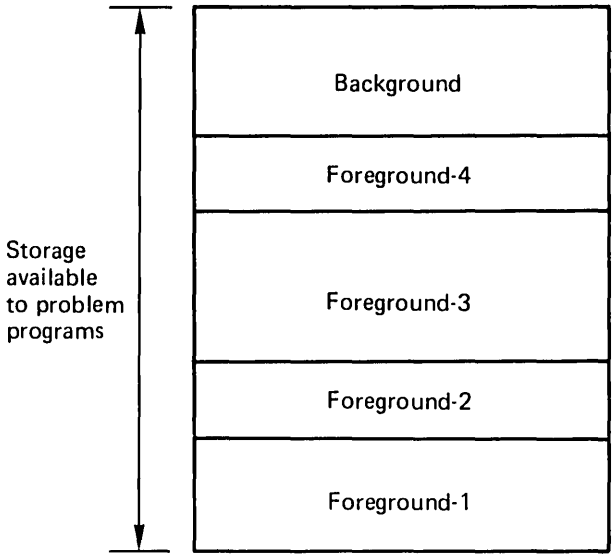


Figure 1.1. The Five Partitions

Storage Protection

Storage protection, which is standard on all System/370 models, ensures that the instructions and data of one program in a given partition do not interfere with those of another program in another partition.

Partition Priorities

During supervisor generation, priorities are established for each partition defined in the system. The default priorities are (from low to high): BG, F4, F3, F2, F1.

During processing the operator can display the partition priorities and change them dynamically by issuing the PRTY command. This can be used to accelerate the execution of a given program. However, the priorities should be reset to the installation standards as soon as possible to handle the normal flow of jobs through the system. Changing priorities in the middle of a job stream should be used with special care if POWER/VS or teleprocessing, which normally run in a high-priority partition, are active in the system. (Refer to *DOS/VS POWER/VS Installation Guide and Reference*.)

Executing a Program in Any Partition

When the relocating loader is generated in the system, most programs can be executed in any partition. Provided that a program being link-edited does not have an origin specified as an absolute address, the program produced for inclusion in the core image library is relocatable.

A relocatable program can be executed in any partition that is large enough to accommodate it.

The relocating loader, as a supervisor generation option, is described in the section *Tailoring the Supervisor* in *Chapter 3: Planning the System.*

Device Considerations

Generally, the same physical I/O device (or extent of a direct access or diskette device) may not be used concurrently by programs being executed in different partitions. The exceptions to this are:

- The device or extents assigned to the system logical units SYSRES, SYSREC, SYSLOG, SYSVIS, and SYSCAT. These devices (extents) are considered to belong to the system as a whole, rather than to individual partitions. (A brief description of these system logical units is contained in the section *Symbolic I/O Assignment* in *Chapter 5: Controlling Jobs.*)
- Private libraries which may be shared for read-only operations (for more information refer to *Using Private Libraries* in chapter 7: *Using the Libraries.*
- A file on a direct access device can be accessed across partitions, providing it is not being created simultaneously by programs in more than one partition (see *Track Hold Option* in *Chapter 3: Planning the System* for information on protection when updating a file concurrently by separate tasks).

If, for example, you wish to link-edit programs in different partitions concurrently, different physical devices or extents (except for SYSRES and SYSLOG) must be assigned for each partition to all logical units used by the linkage editor program. Figure 1.2 shows how devices may be assigned in order to link-edit in two partitions concurrently.

Logical Unit	F1 Partition	BG Partition
SYSIN	X'181'	X'00C'
SYSLST	X'182'	X'00E'
SYSLOG	X'01F'	X'01F'
SYSLNK	X'131'	X'132'
SYS001	X'131'	X'132'
SYSCLB	X'130'	--
SYSRES	X'130'	X'130'

Figure 1.2. Assigning Different Physical Devices to the Same Logical Units

In this case, the output on SYSLST in F1 is written on a tape. A listing of this output can be obtained by printing the tape after the job is completed. If POWER/VS is used, the listing could be automatically obtained whenever a printer becomes available. (Refer to *DOS/VS POWER/VS Installation Guide and Reference.*)

Virtual Storage

Through a combination of System/370 hardware design and programming support, DOS/VS has an address space, called *virtual storage*, that can extend to the maximum allowed by the system's addressing scheme, which is 16,777,216 bytes (16M bytes).

Virtual storage consists of two distinct areas; the real and the virtual address area.

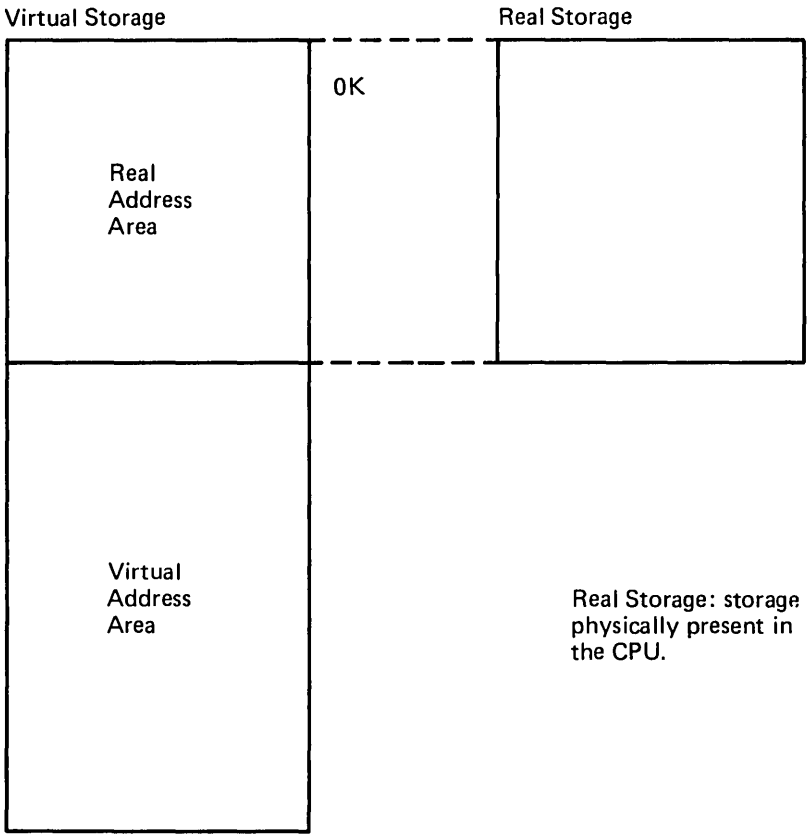


Figure 1.3. Interrelationship of Real and Virtual Storage, Real and Virtual Address Area

Figure 1.3 shows that the area of virtual storage where the virtual addresses match the real addresses is called the *real address area,* and the area that begins at the end of the real address area and extends to the end of virtual storage is called the *virtual address area.* Addresses in this area have no direct equivalent to addresses in real storage.

How much of the maximum address space (16 M bytes) will be used in a particular system depends on a number of factors: the size of the computer's real storage, the amount of disk storage available, the number of partitions, their sizes, and the characteristics of the installation's programs and operating environment.

Both the real address area and the virtual address area are available for use when writing your programs, but not both together for a single program. Some of your programs can be considered to be loaded into the virtual address area, and others into the real address area. Of course, each instruction of a program must be in real storage at the moment it is executed, and so must the data it manipulates. The other instructions and data of a program loaded into the virtual address area need not be in real storage at that same moment; they can reside on auxiliary storage until needed. The file used for this purpose is called the *page data set.* This makes it possible to execute programs that are larger than any real partition, or even real storage.

Some programs can be loaded at IPL time into a special area, called the *shared virtual area* (SVA). Those programs can then be executed directly (without subsequent loading) by any job in any partition, and may be executed concurrently from more than one partition. The shared virtual area is located in the virtual address area and, therefore, is represented on the page data set.

It would be inefficient, however, to bring every instruction and its associated data into real storage individually. Programs in virtual storage are manipulated in sections called *pages;* the size of a page in DOS/VS is 2K bytes. Real storage is divided into 2K byte sections; these are called *page frames.* Page frames accommodate pages of a program during execution. This is illustrated in Figure 1.4.

The DOS/VS supervisor will occupy the low order page frames, while the remaining page frames are available for the execution of processing programs. Those page frames unoccupied by the supervisor and available for execution of programs in the virtual area, are collectively called the *page pool.*

When a program is loaded from the core image library into virtual storage, all its pages are brought into page frames of the page pool. If there are not enough page frames available to contain all the pages of a program being loaded into the virtual address area, the system moves the contents of some page frames to a disk extent called the *page data set.* The remaining pages of the program can then be loaded.

During execution of the program, whenever a required instruction or some data is not present in real storage, execution is interrupted by a so-called *page fault.* The system must then bring the requested page into real storage.

For programs loaded into the virtual address area, pages can be placed into any available page frame during execution. Since the system does not anticipate where in real storage a page will be loaded, the virtual addresses must be translated into real addresses when required for execution. The address translation is performed by a combination of the System/370 Dynamic Address Translation (DAT) facility and DOS/VS.

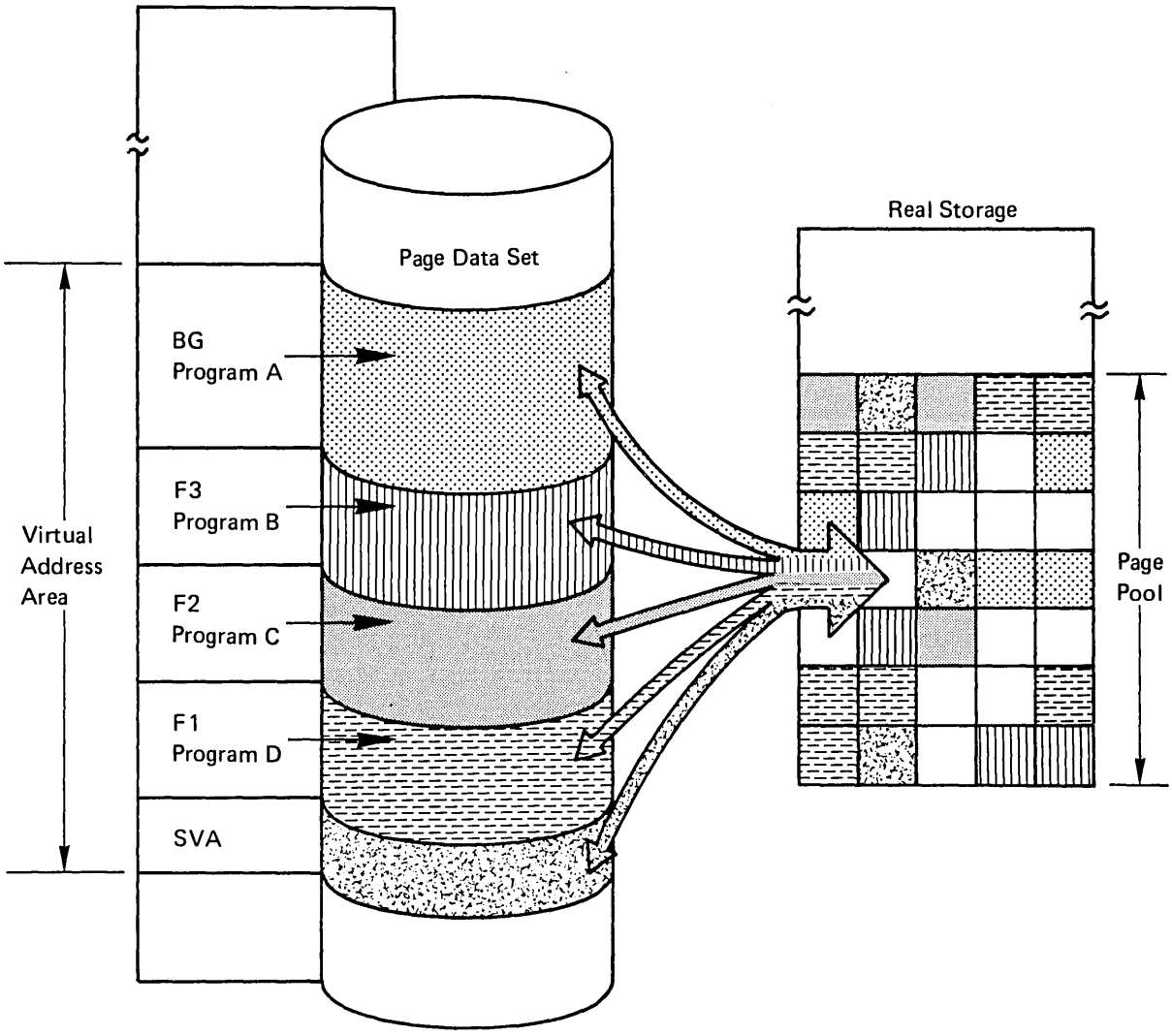


Figure 1.4. Four Programs Being Paged

Assignment of page frames is done by the supervisor which works toward keeping the most frequently-used pages of each program in real storage.

Any or all of the four programs being paged may also concurrently use phases in the shared virtual area (SVA).

Real and Virtual Partitions

During system generation, the number of partitions is defined for the system. A certain amount of address space must be associated with (allocated to) each partition. Each partition in which a program is to be loaded for execution is required to have address space in the virtual address area; this space is called a *virtual partition.* Each partition may also have address space in the real address area; this space is called a *real partition.* Because the job control program (which is necessary to start the execution of each problem program) requires a virtual partition for its execution, a real partition always has a corresponding virtual partition.

Figure 1.5 assumes that five partitions have been defined in the system. On the left is a system without real partitions; on the right is a system with real partitions. It is unlikely that you will have allocated five real partitions, but they are illustrated here to show their relative position in storage.

The Shared Virtual Area

In multiprogramming systems, a system directory list (SDL) and certain frequently used programs can be loaded into the shared virtual area (SVA), which is located in the highest address space in the virtual address area. Such programs (or parts of programs), which are relocatable and reenterable, are available for concurrent use by programs running in virtual or real mode. Programs in the SVA are always executed in virtual mode in the page pool.

Executing Programs in Real and in Virtual Mode

Programs can be executed in two modes:

- *Virtual Mode:* the program's addresses refer to addresses in the virtual address area, and the program executes in the page pool; the precise location a page occupies is not known until it is needed for execution. Paging can take place.
- *Real Mode:* the program's addresses refer to addresses in the real address area and the program executes in a contiguous, defined block of real storage: the real partition. No paging takes place.

For either mode, sufficient address space must be allocated to the partition to accommodate the program to be executed. Sufficient page frames must be available in the main page pool to execute programs from the shared virtual area.

Under DOS/VS certain programs - such as those with critical time dependencies - may have to run in real mode. The DOS/VS supervisor also always runs in real mode.

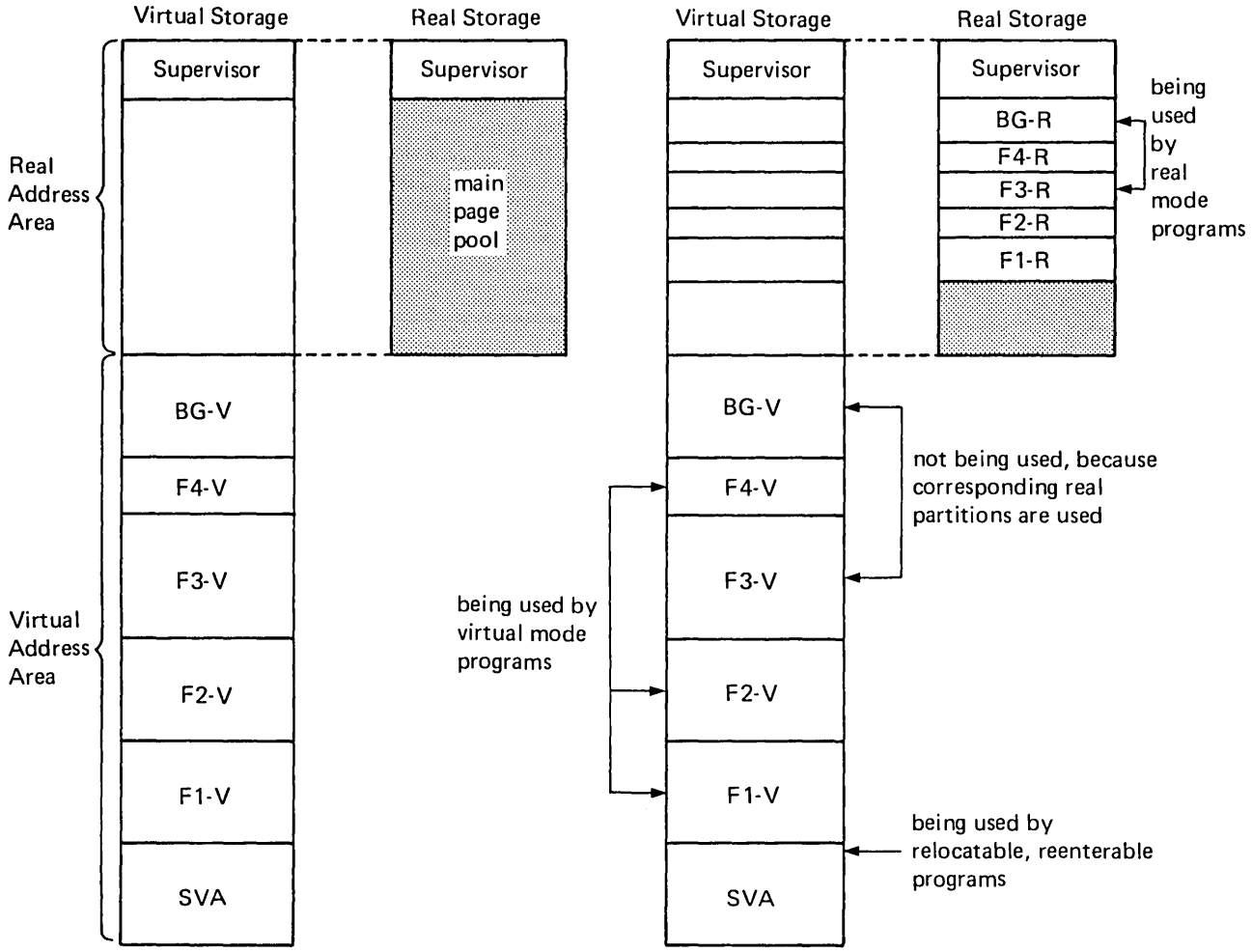


Figure 1.5. A 5-Partition System With and Without Real Partitions

In both systems the heavily shaded parts of real storage are not allocated to any particular partition. These parts are called the main page pool, which (in the system on the right) is augmented by the address space of the real partitions that are not being used (lightly shaded), to form the page pool.

When a real partition is being used, the address space in the corresponding virtual partition cannot be used.

Programs in the shared virtual area (SVA) can be shared concurrently by programs running in either virtual or real mode. The programs from the SVA are executed in the page pool.

Real partitions are used not only for programs running in real mode, but also for programs running in virtual mode that fix a set of instructions or data (using the PFIX macro, which is discussed in more detail under *Fixing Pages in Real Storage* in the section *Tailoring the Supervisor* in Chapter 3). Such pages of a virtual-mode program are fixed in page frames of the real partition that corresponds with the virtual partition in which the program is running.

Page Pool

As shown in Figure 1.5, the real storage not allocated to any real partition (or occupied by the supervisor) is always available for paging activities. It forms the *main page pool.* Other page frames may also belong to the page pool:

- When not occupied by a program running in real mode, the area allocated to a real partition is available to virtual-mode programs.
- When a program running in real mode does not require the entire real partition, the unused part of the real partition may be made available to the page pool by specifying the required amount of storage in the SIZE operand of the EXEC job control statement for the real-mode program.

Advantages of Virtual Storage

In summary, executing programs in virtual storage has two main advantages:

- It allows execution of programs that are larger than the available real partition, or even larger than real storage.
- The real storage available is better utilized: programs running in a virtual partition are not confined to a particular area of real storage, but may use *all* available page frames.

Partition and system performance requirements should be considered as you relate these advantages to your particular installation.

Multitasking

At the beginning of this chapter, we defined multiprogramming as the ability to execute more than one program concurrently in separate partitions within a single computer system. Multitasking can be regarded as an extension of multiprogramming in that it provides the ability to execute more than one program concurrently in a single partition. In simple terms, therefore, multitasking can be regarded as multiprogramming within a partition.

Multitasking presupposes the existence of the multiprogramming facilities in the supervisor (in particular, the task selection routines). Multitasking is, therefore, possible only in a multiple-partition environment. As a supervisor generation option, multitasking is described in the section *Tailoring the Supervisor* in *Chapter 3: Planning the System.*

Some installations using former versions of DOS, employed multitasking to run more than three programs in a three-partition system. The additional partitions that DOS/VS provides may serve the same purpose. You should note that running programs concurrently in separate partitions usually requires less preparation than running programs concurrently in the same partition.

Two Types of Multitasking

Programs (or parts of a program) that are executed concurrently in a given partition are called tasks. A distinction is drawn between the main task in a partition and one or more subtasks in the same partition. The main task is that program (or program part) initiated by job control. The subtasks are those programs (or program parts) initiated by the main task through the use of the ATTACH macro instruction. To use the multitasking facilities of DOS/VS it is necessary to code the main task in the assembler language.

The subtasks executed in a given partition may be: (1) logically independent, or (2) logically dependent.

In the first case, the main task monitors the execution of the subtasks, treating them as independent programs. Such subtasks may be coded in any programming language. This type of multitasking is sometimes called multiprogramming within a partition. It is suitable technique to use, for example, for concurrent execution of more programs than partitions available.

In the second case, both the main task and the subtasks are program routines that are logically part of the same program. Thus, the tasks can communicate with one another. In this case the subtasks are likely to be coded in assembler language to allow the use of the task intercommunication macros. They can share code (in particular, an access method or subroutines), provided that it is of a read-only nature (that is, that the code or subroutines are not modified during execution). This technique is complex and can best be understood after studying the first type of multitasking.

Cross-Partition Event Control

Highly complex applications may have a need for communication between programs executing in separate partitions. For example, two such programs may need to perform operations on a common file, and the operations may require actual communication between the two programs.

Through cross-partition event control macros, one partition can delay the execution of part of a program until another partition signals the completion of a critical event. This allows synchronized multiprogramming in separate partitions --thus protecting programs against inadvertent destruction of each other-- while at the same time providing for any necessary communication between them. For details about this support, see *Supervisor and I/O Macros*.

Chapter 2: Summary of DOS/VS Features

Standard Features of DOS/VS

These features are automatically included during system generation:

- Support for one virtual storage of user-specified size (up to 16M bytes).
- Batched-job mode of job initiation in a single-partition environment.
- Execution of programs in real mode and virtual mode.
- Symbolic I/O device assignment.
- Cataloged procedures.
- Storage protection.
- SAM, DAM, and ISAM.
- Command chaining for I/O retry operations.
- Tape error statistics.
- Selector channel support.
- Display operator console support (for Models 115 and 125: Video Display Keyboard Console).
- Machine check analysis and recording (MCAR), channel check handler (CCH), and recovery management support recorder routines (RMSR).
- OLTEP (optional on Models 115 and 125; can be omitted for other models).
- Job control.
- Linkage editor.
- Librarian.
- Assembler.
- System utilities (including Disk Volume Fast Copy).
- System debugging aids (SDAIDS).
- Relocating loader

Optional Features of DOS/VS

These features must be requested during system generation or added after the generation has been performed:

- Level of Multiprogramming.
- Specification of partition dispatching priority.
- Multitasking.
- POWER/VS.
- Teleprocessing support (BTAM, QTAM and VTAM).
- VSAM.

- Wait multiple support.
- Cross-partition event control support.
- Magnetic ink character reader and optical character reader support.
- Page fault handling overlap.
- Support for PFIX/PFREE macros.
- Support for GETVIS/FREEVIS macros.
- Support for RELPAG/PAGEIN/FCEPGOUT macros.
- Integrated emulators.
- Time-of-day clock support.
- Multiple timer support.
- Job accounting interface.
- Private core image libraries.
- External interruptions.
- Abnormal termination exit.
- Console buffering.
- Track hold.
- DASD file protection.
- Rotational Position Sensing (RPS).
- Seek separation.
- Channel switching for magnetic tapes.
- Burst mode operation on the byte multiplexer channel.
- Error volume analysis for magnetic tapes.
- Reliability data extractor.
- Problem determination aids (PDAIDS).
- ASCII support for tapes.
- System input and system output files on disk (SYSFIL option).
- Independent directory read-in area.
- Task timer support

DOS/VS in Various CPUs

This section shows, by way of a series of examples, how real and virtual storage could typically be employed by DOS/VS running in CPUs with different amounts of real storage. The real storage requirements of the supervisors and of the main DOS/VS features are indicated, as are the types of jobs that are processed in the partitions. In each of the examples, the real storage available to the main page pool can be obtained by subtracting the amount of real storage allocated to the supervisor and the real partitions from the CPU size. *In all cases, the figures given are approximations.*

All systems have an SVA that contains a system directory list. However, the illustrations do not explicitly show the SVA unless it must be larger than the minimum size, as for example for RPS or VSAM.

96K CPU

	Storage (K bytes)	
	Real	Virtual
Supervisor	40	
BG	10	64
F3	0	64
F2	0	64
F1	24	156
	74	

Notes:

- Batch processing operation.
- One "hot" partition for urgent, unscheduled jobs.
- POWER/VS in F1

The system described above might be typical of a DOS/VS user who formerly operated a Model 20 with programs that did not require large amounts of storage.

144K CPU

	Storage (K bytes)	
	Real	Virtual
Supervisor	42	
BG	0	600
F3	0	256
F2	0	256
F1	24	156
SVA		302
	66	

Notes:

- POWER/VS in F1
- VSAM and Access Method Services in BG
- VSAM and SDL in SVA

192K CPU

	Storage (K bytes)	
	Real	Virtual
Supervisor	52	
BG	0	600
F4	0	192
F3	0	192
F2	26	158
F1	50	192
SVA		302
	128	

Notes:

- POWER/VS in F2
- CICS/VS (an IBM program product, Customer Information Control System/Virtual Storage) in F1
- VSAM and SDL in SVA
- VSAM and Access Method Services in BG

240K CPU

DAYTIME	Storage (K bytes)	
	Real	Virtual
Supervisor	58	
BG	30	288
F3	0	288
F2	30	170
F1	60	672
	178	

Notes:

- One partition (BG) for compiling/testing
- one production partition F3
- POWER/VS in F2
- CICS/VS in F1

NIGHTTIME	Storage (K bytes)	
	Real	Virtual
Supervisor	58	
BG	50	608
F3	0	288
F2	0	288
F1	30	170
	138	

Notes:

- One batch partition (using PFIX/PFREE macros) in BG
- POWER/VS in F1
- Two batch partitions (not using PFIX/PFREE macros) in F2 and F3

384K CPU

DAYTIME	Storage (K bytes)	
	Real	Virtual
Supervisor	54	
BG	14	722
F3	28	228
F2	66	228
F1	48	176
SVA		100
	210	

Notes:

- POWER/VS RJE in F1
- CICS/VS in F2
- Two batch partitions
- RPS code in SVA

NIGHTTIME	Storage (K bytes)	
	Real	Virtual
Supervisor	54	
BG	36	600
F4	36	228
F3	36	228
F2	36	164
F1	72	512
SVA		402
	270	

Notes:

- CICS/VS in F1
- POWER/VS in F2
- Access Method Services in BG
- Three batch partitions
- VSAM and RPS code in SVA

Chapter 3: Planning the System

The IBM-shipped DOS/VS includes a number of supervisors, in the core image library, from which one or more can be selected to form the base for the system to be generated. Each of the supervisors provides a specific range of functions. Should the functions of the supervisor(s) not be in agreement with the system functions planned, the system programmer can tailor the supervisors to include the desired functions.

The assembler language source for the provided supervisors is contained in the source statement library (sublibrary A) and can be displayed by using the job stream given under *System Generation Example (on-line)* in *DOS/VS System Generation*.

After a brief description of the system generation procedure in general, this chapter describes in greater detail two major considerations during system generation, namely:

- Tailoring the supervisor (adding functions to those of the basic supervisor)
- Planning the libraries (planning the contents, the location and size of the libraries).

Because of the nature of this information, this chapter primarily addresses system programmers, who are responsible for planning the system. The section *Tailoring the Supervisor*, however, may be of interest to all DOS/VS users who wish to become more acquainted with this component of the system.

System Generation Procedure

Proper and detailed planning is essential to efficient system generation and minimizes the need to modify the system after it is generated. You may want to contact your IBM marketing representative to set up a system generation planning meeting. IBM field engineering would also attend the meeting to discuss the procedure to install the SCP (systems control programming). Generating a system includes:

- Planning the options and estimating the approximate size of the supervisor. This entails selecting from the programming services provided by IBM, those options you wish to include in the supervisor, and estimating the cost of these services in terms of bytes of storage.
- Planning the contents, organization, and size of the system and (optionally) private libraries. This entails distributing the storage space available (on the disk packs) between the libraries desired for day-to-day use. The major points you must consider are:
  a. the size of the system core image library and, other system and private libraries

b. workfile space needed to assemble the supervisor and to link-edit and catalog the components selected to the system core image library

c. standard assignments for workfiles needed for everyday operation.

You work with the IBM-supplied distribution medium, which is composed of four libraries:

The *system procedure library* initially contains procedures useful for generating DOS/VS, linking and deleting DOS/VS component, and loading the SVA with SDL entries and recommended phases.

The *system source statement library* contains macro definitions for various system components and services. Included are macro definitions (sublibrary E) from which you choose desired parameters in order to assemble your new supervisor. For your convenience, the source statement library also contains sample programs (sublibrary Z), system history model macros (sublibrary Y), and sample supervisors; they are illustrated in *DOS/VS System Generation.*

The *system relocatable library* contains assembled IBM programs and assembled macros from the source statement library. For example, logical IOCS, which performs input and output operations for IBM programs and your programs.

The *system core image library* contains all programs that are ready for execution.

The specific contents of these libraries vary from release to release and are identified in the *Program Directory,* which accompanies the system distribution medium.

Using the elements of these libraries, you

- Generate the supervisor by coding a set of supervisor generation macros, which define the system configuration and the services you wish the supervisor to contain. (These are described in detail in the section *Tailoring the Supervisor.*
- Generate POWER/VS, if desired, by coding a set of POWER/VS generation macros, which define its configuration and optional services. (These are described in detail in *DOS/VS POWER/VS Installation Guide and Reference.*)
- Delete from the libraries any components you do not require and then condense to free library space.
- Assemble (or compile) and/or link-edit programs - both your own and IBM's - and catalog them into the appropriate libraries.

After determining what elements are to be contained in the system libraries, you may wish to retain additional elements in private libraries and therefore you may want to create private core image, relocatable, or source statement libraries. These choices are discussed in the section *Planning the Libraries.*

The system libraries, together with certain system work areas, constitute the system residence file (SYSRES), which is one extent of a direct access storage volume. The SYSRES file is described in *Appendix A: System Layout on Disk.*

After establishing your SYSRES file, you should copy it onto tape or disk for backup purposes. The utility programs Backup/Restore DOS/VS System and Fast Copy Disk Volume, which are provided for this purpose, are described in *DOS/VS System Utilities.*

For complete details on how to perform a system generation procedure refer to *DOS/VS System Generation.*

Tailoring the Supervisor

This section describes the optional and required parameters that you select for the generation of the supervisor. The parameters are included in the following supervisor generation macros:

ALLOC	IOTAB
ALLOCR	PIOCS
ASSGN	SEND
CONFG	STDJC
DPD	SUPVR
DVCGEN	VSTAB
FOPT	

The parameters of these macros are discussed in a topical sequence, such that related options are presented together regardless of the macros in which they are contained. For the exact formats of these macros, refer to *DOS/VS System Generation.*

This section discusses the advantages or necessity of specifying the support for the various features in the supervisor.

In tailoring your supervisor to the requirements of your installation, you can take into consideration future plans to add hardware (main storage, I/O devices, and so on) or other functions that require supervisor options by including their requirements in your supervisor generation macros. This will allow you to upgrade your installation without having to regenerate your supervisor. You may also want to include in the libraries modules or components that will be required by planned future configuration or functional upgrades. The storage cost of additional supervisor options may be estimated by consulting the supervisor storage requirements in Module 1 of *DOS/VS System Generation.*

Storage Management Options

This section describes those supervisor options that relate to virtual and real storage. These include defining:

- The size of virtual storage (virtual address area, real address area, and the shared virtual area)
- The number and size of partitions, and their priorities
- The page data set (SYSVIS)
- The ability to fix pages in real storage
- The virtual storage access method (VSAM)

The size of virtual storage must be defined. Virtual storage is composed of the virtual address area and the real address area, and the size of each must be separately specified. You specify the size of the virtual and real address areas in the VSIZE and RSIZE operands of the VSTAB macro.

Defining the Size of the Real Address Area. Normally, you select a value for RSIZE that coincides with the amount of real storage in your CPU model. If, however, you anticipate that your system may also be used on a CPU with larger real storage, you should select the value for RSIZE such that the end of your real address area coincides with the end of real storage of the larger CPU. Otherwise, some real storage remains unused when using the larger CPU. This is illustrated in Figure 3.1. Specifying a value for RSIZE that is larger than the size of your current real storage, (see Figure 3.2) causes the start address of the virtual address area to be higher than the end address of real storage. Nevertheless, none of the virtual address area or real storage of the smaller CPU will remain unused.

Defining the Size of the Virtual Address Area. The value you specify for VSIZE is equal to the sum of the sizes of the virtual partitions and the size of the shared virtual area. Therefore, you must know these sizes before you can specify VSIZE. For selecting the size of the individual partitions, see *Defining the Size of Virtual Partitions*, later in this section. For selecting the size of the shared virtual area, see *Defining the Size of the Shared Virtual Area.*

The value specified for VSIZE cannot be changed without a new supervisor generation.

The maximum size of virtual storage is 16M (16,777,216) bytes. Because the real address area is part of virtual storage, the maximum value you can specify for VSIZE is 16M minus the size of the real address area (RSIZE).

In a single-partition system, the value you specify for VSIZE must be equal to or greater than 64K bytes (the minimum virtual background partition).

The value you specify for VSIZE is used by the system to determine the size of the page data set. Refer to *Defining the Page Data Set* later in this section.

Defining the Size of the Shared Virtual Area. The shared virtual area (SVA) can contain any program that is reenterable and relocatable. Such programs can be used concurrently by more than one partition. Having phases resident in the SVA avoids frequent fetches; the phases can be loaded into the SVA at IPL time or at the time they are cataloged into the system core image library.

As illustrated in Figure 3.3, the SVA is located in the high address space of the virtual address area. The SVA contains a system directory list (SDL), which provides fast retrieval of frequently used phases that are resident in the SVA or in the system core image library. Having SDL entries avoids searching multiple tracks of the core image directory for each FETCH or LOAD request. The SDL and the SVA always reflect the

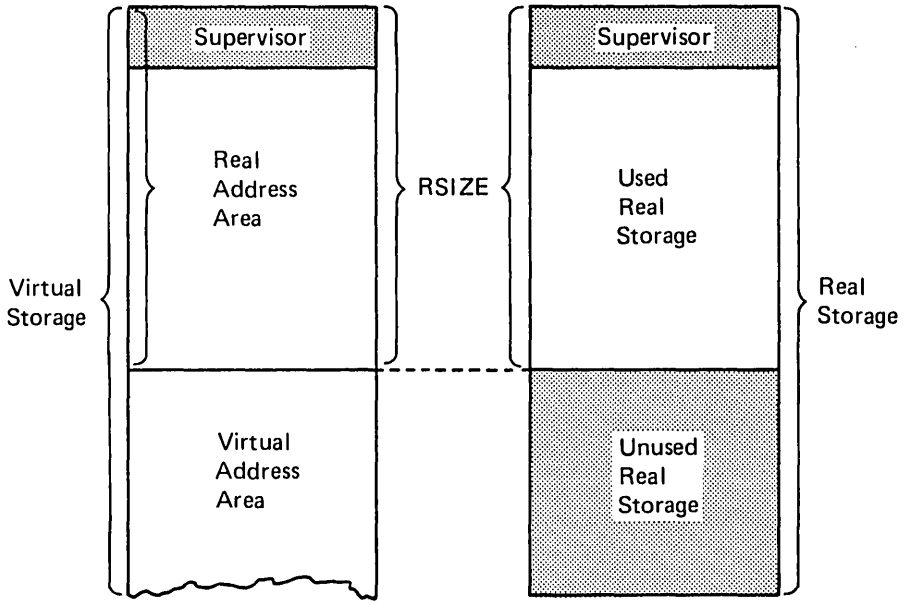


Figure 3.1. Insufficient Specification of RSIZE

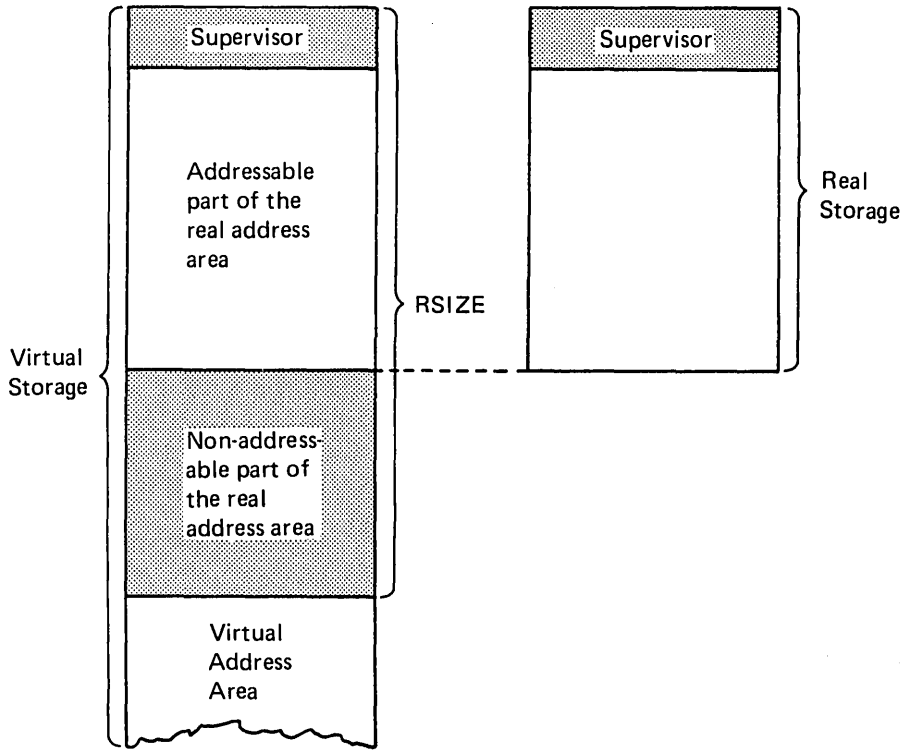


Figure 3.2. Specification of RSIZE Larger Than the Size of Real Storage

current status of the equivalent information in the system core image directory and library. In other words, the SVA will be updated when an SVA-eligible program is cataloged into the core image library.

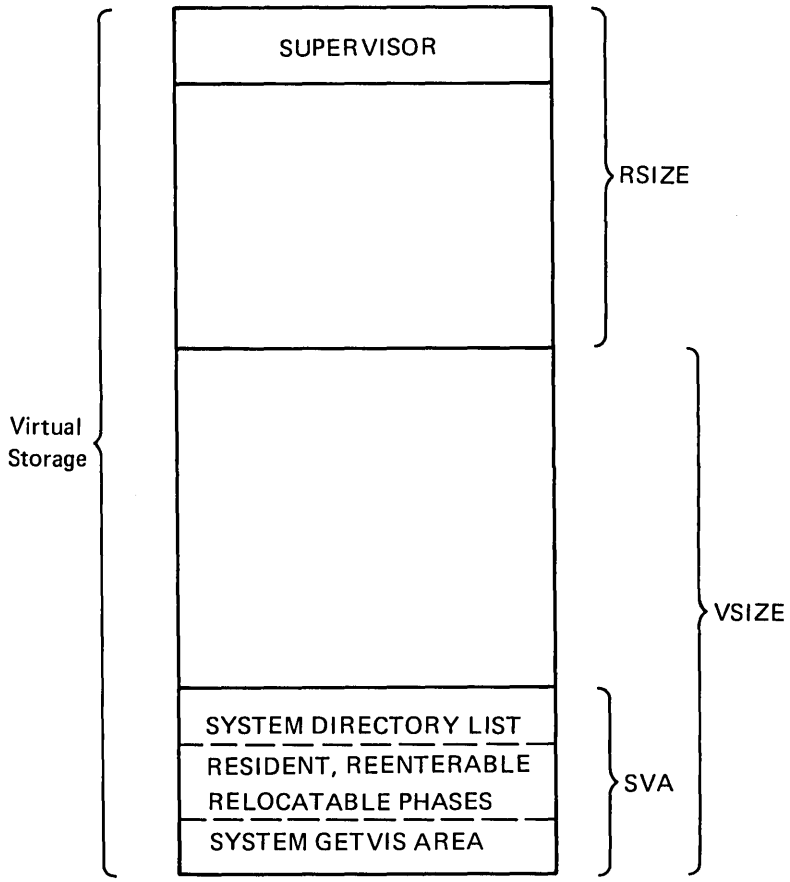


Figure 3.3. Location of the Shared Virtual Area

Note that the VSIZE specification includes the SVA specification.

You specify the size of the shared virtual area and the system GETVIS area in the SVA parameter of the VSTAB macro. If the supervisor supports RPS (rotational position sensing), 100K bytes are required for it in the SVA. Either all or part of the RPS code will be loaded into the system GETVIS area (a part of the SVA). If RPS is not preloaded at IPL time, then 100K is required in the system GETVIS area. If RPS is preloaded, then 12K is required in the system GETVIS area and 88K must be available for RPS in the SVA.

If your programs will process VSAM files, VSAM phases should be loaded into the SVA. If the IBM distributed VSAMSVA procedure is used to load VSAM phases (along with other recommended SVA eligible IBM phases) into the SVA, approximately 302K is needed in the SVA.

The SVA must be large enough to accommodate the system directory list and the programs loaded there, but it cannot be smaller than 64K. The size of the SVA that you specify during supervisor generation can be overridden by issuing SET SVA immediately after IPL. This command is discussed in the section *Building the SDL and Loading the SVA* in *Chapter 4: Starting the System.*

Defining the Number of Partitions

In the NPARTS parameter of the SUPVR macro you define the maximum number of partitions for your system.

In selecting the appropriate number of partitions for your particular installation, you should consider the type of processing you require. For example, assume you want to run concurrently the following types of programs:

- Test cases (assemble/compile, link-edit, and execute)
- Daily application programs
- POWER/VS
- Teleprocessing application program.

For this case, you should generate a system with four to five partitions, depending on the volume of application program processing. If your system includes the unlicensed version of VTAM, at least two partitions must be specified: one for VTAM and one for VTAM application programs.

For examples of typical partition usage, refer to *DOS/VS in various CPUs* in chapter 2.

Because you cannot alter the NPARTS specification unless you regenerate the supervisor, it may be advantageous to specify more partitions than you see an immediate need for.

Note: *For VTAM and QTAM at least two partitions must be specified.*

Defining the Size of Partitions

If you generate a multiple-partition system, you may explicitly define the size of each partition (except the virtual background partition). In a single-partition system the size of the virtual partition is implied by the specification of the VSIZE parameter, and the size of the real partition is implied by the specification of the RSIZE parameter minus the supervisor size.

The size of a partition is defined by specifying the amount of storage you wish to allocate to it. The ALLOC macro is used to allocate storage to virtual partitions; the ALLOCR macro is used to allocate storage to real partitions. Specification of ALLOC and ALLOCR macros during supervisor generation is optional since operator commands to allocate real and virtual storage are provided in DOS/VS. The size of both virtual and real partitions is specified as a multiple of 2K bytes.

Defining the Size of Virtual Partitions. Only the size of the virtual foreground partitions is explicitly defined (via the ALLOC macro). The virtual address area not allocated to any of the virtual foreground partitions and not allocated to the SVA is automatically allocated to the virtual background partition. At least 64K bytes must be left for the virtual background partition.

The size of an active virtual foreground partition must be at least 64K bytes. If a virtual foreground partition is defined but need not be used for a while (see *Defining the Number of Partitions* above), its size can be set to 0K, either by the ALLOC macro during system generation, or by the ALLOC command during actual operation. When using RPS, leave approximately 6K available for the partition GETVIS area, required by RPS for control blocks.

You specify the size of each virtual foreground partition by means of the ALLOC macro. The system then calculates the difference between the VSIZE specified minus the SVA value and the ALLOC value to determine the size of the virtual background partition. If this difference is less than 64K or if you omit the ALLOC macro during supervisor generation, all of virtual storage except the shared virtual area is allocated to the virtual background partition and the size of each virtual foreground partition defined in NPARTS is set to zero.

During certain periods of processing, the operator can modify the size of the individual virtual partitions by using the ALLOC command. Details on the ALLOC command are given in *DOS/VS Operating Procedures.*

Defining the Size of Real Partitions. Potentially, for each virtual partition defined in the system a corresponding real partition can be allocated. A real partition consists of a contiguous set of addresses in the real address area.

Real partitions need only be allocated to enable the following:

- Program execution in real mode
- Use of the PFIX/PFREE macros.

When a real partition is used for running a real mode program, or for fixing pages of a virtual mode program, (for example, POWER/VS), the page pool is reduced by the number of page frames required.

Because reducing the page pool in turn may reduce total system throughput, the use of real partitions should be carefully considered. When a program is running in real mode, the real storage allocated to its partition is taken from the page pool.

For each of the above cases, the virtual partition that corresponds to the real partition must be allocated (64K bytes minimum). This is because the initiation of either virtual-mode or real-mode programs is performed by the job control program, which itself runs in virtual mode. Thus, for example, the virtual F1 partition must be allocated at least 64K bytes if the real F1 partition is to be used.

When a program running in virtual mode issues a PFIX macro, the pages are fixed within the corresponding real partition. This ensures that other real partitions are available for other programs that run in real mode or that fix pages in real storage.

To allocate a real partition, specify the partition identifier and its size in the ALLOCR macro. Each real partition you require must be specified explicitly. Note, however, that ALLOCR must not be specified for a single-partition system, because all available real storage is permanently allocated to the background real partition.

A real partition may be as small as 2K bytes: the size of a given real partition is determined either by the largest program you must run in real mode, or by the maximum number of pages a virtual-mode program may fix.

The allocation of real partitions cannot exceed the size of the real address area (specified in the RSIZE parameter) minus the supervisor area.

In addition, the main page pool size must be taken into account and may be determined from the table below. The sizes shown are minimums. Also not reflected is the additional storage available to the page pool, as described in the section *Page Pool* in chapter 1.

Size of smallest real partition	PFIX=NO	PFIX=YES or SVA phases used	AP=YES (Note 2)
18K or less (including 0K)	18K minus size of smallest real partition (Note 1)	18K	+2K
larger than 18K	0K (Note 1)	18K	+2K

Note 1. If the SDL is active, the main page pool must be at least 4K.

Note 2. An additional 2K bytes must be added to the main page pool size if multitasking (AP=YES) is specified.

The system ensures (for single as well as multipartition systems) that this minimum, in which pages cannot be fixed, remains.

The supervisor indicates, by means of return codes in register 15, whether or not a PFIX macro has been executed successfully. For an example of the use of PFIX and PFREE macros and the supervisor return codes, refer to the section *Fixing Pages in Real Storage.*

Defining Partition Priorities

A priority is associated with each partition in a multiprogramming system. If you do not specify priorities during system generation, the supervisor will establish default priorities. These default priorities (from low to high) are shown in Figure 3.4.

NPARTS=2	PRTY=(BG,F1)
NPARTS=3	PRTY=(BG,F2,F1)
NPARTS=4	PRTY=(BG,F3,F2,F1)
NPARTS=5	PRTY=(BG,F4,F3,F2,F1)

Figure 3.4. Default Partition Priorities

In most cases, there will be no need to select another priority sequence; however, the PRTY parameter in the FOPT macro is provided for this purpose. In the PRTY parameter you can specify the partition identifiers in any desired sequence, and thus select another priority sequence.

The operator can display and modify the priorities established during supervisor generation at any time during processing with the PRTY command. He may want to do this in order to accelerate the execution of a given job.

Defining the Page Data Set

The page data set, a sequentially organized set of records on a direct access device, is required in DOS/VS to accommodate pages of programs that are being executed in virtual mode that have been paged out. There are as many 2K records on the page data set as there are 2K pages in the virtual address area. The size of the page data set, therefore, depends on the size of the virtual address area.

The page data set can reside on any disk device that is supported by DOS/VS as a system residence device.

You can define the page data set in the DPD macro, in which you can specify the channel and unit number of the device and the lower limit address of the extent. The upper limit address is calculated by the system according to the VSIZE parameter specified in the VSTAB macro. If you correctly specify the DPD macro, an MNOTE is issued in the supervisor assembly listing that indicates the required number of tracks for all different types of devices supported as a page data set.

You may also specify a volume serial number, which will be checked when the page data set is opened.

If you omit the DPD macro, or some of its parameters, during supervisor generation, or the information you specify is erroneous, you must define the page data set during IPL via the DPD command. (This command is discussed in the section *Initiating Page Data Set Handling* in *Chapter 4: Starting the System.*) The information specified in the DPD command overrides the information supplied during supervisor generation until the next IPL.

Fixing Pages in Real Storage

A program that runs in virtual mode is executed in page frames of the page pool. When a page frame is required by a virtual-mode program and all are currently occupied, one of the occupied page frames will be freed, if necessary by paging its contents out onto the page data set.

Some programs that run in virtual mode contain code (such as I/O appendages) that must be in real storage when needed and therefore cannot tolerate paging. The pages containing such code can be fixed temporarily via the PFIX macro instruction, and freed immediately after use via the PFREE macro instruction. POWER/VS is an example of an IBM-supplied program that uses PFIX/PFREE macros.

When pages of a program running in a given virtual partition are fixed in response to the PFIX macro, they are fixed in the corresponding real partition. Therefore, the use of the PFIX macro requires that a real partition be allocated sufficient storage to accommodate the pages to be fixed at any given time. If a PFIX macro is issued when a real partition is not allocated enough storage, the pages are not fixed, and a completion code indicating this is returned to the program.

If you anticipate the need for the PFIX/PFREE macro instructions in any of your virtual-mode programs, specify PFIX=YES in the FOPT macro during supervisor generation.

Fixing pages in real storage means that in a multiprogramming environment fewer page frames are available to other programs running in virtual mode, potentially degrading total system performance. When channel programs with large I/O areas are involved, the initial size of the page pool may be too small. Consider this effect carefully before enabling the use of the PFIX macro. Examples of the use of the PFIX/PFREE macros are provided in *Chapter 9: Designing Programs for Virtual-Mode Execution.*

Improving the Paging Mechanism

The page handling of virtual mode programs is controlled by the page management routines of the supervisor. You can, however, influence the paging mechanism in order to reduce the number of page faults, to minimize the page I/O activity, and to control the page traffic within a specific partition. You can do this by means of three macros: RELPAG, FCEPGOUT, and PAGEIN.

RELPAG (Release Page). With this macro you inform the page management routines that the contents of one or more pages is no longer required and need not be saved on the page data set when the page frames occupied by these pages are claimed for use by other pages. This saves unnecessary page I/O.

FCEPGOUT (Force Page-out). With this macro you inform the page management routines that one or more pages will not be needed until a later stage of processing, and that they should be given highest page-out priority. This prevents page-out of other pages which might be needed again immediately after being written out.

PAGEIN. With this macro you request one or more pages to be paged in in advance, so that page faults can be avoided when the specified pages are needed in real storage. If the specified pages are already in real storage when the macro is issued, they are given lowest priority for page-out.

If you anticipate the use of one or more of the above macros in any of your virtual mode programs, specify PAGEIN=n in the SUPVR macro during supervisor generation. This will generate support for the three macros. The value of n must be 1 or greater. It specifies the number of page-in requests that can be queued if more than one PAGEIN macro is issued concurrently in the system.

Virtual Storage Access Method

The virtual storage access method (VSAM) can be used for direct or sequential processing of fixed and variable-length records (including spanned records) on direct access storage devices. A significant feature of VSAM is data portability. VSAM files can be processed by DOS/VS, OS/VS1, and OS/VS2.

VSAM requires a special file, the VSAM master catalog, which contains information on file and disk characteristics. In addition, VSAM supports any number of user catalogs for alternative use. The VSAM master catalog resides on a disk extent that is contained on the logical unit SYSCAT. Catalogs are defined and maintained by the Access Method Services and used by OPEN and CLOSE. For complete information on VSAM, refer to *DOS/VS Data Management Guide* and *DOS/VS Supervisor and I/O Macros*.

Support for VSAM is generated in the supervisor, by specifying VSAM=YES in the FOPT macro. Most VSAM phases can be loaded into the shared virtual area. For details refer to the sections *Defining the Size of the Shared Virtual Area, and Reserving Storage for VSAM*.

Multiple-Partition Options

There are certain options that can be specified during supervisor generation that are particularly designed for a multiprogramming environment. The options described in this section are:

- Relocating loader
- POWER/VS
- Multitasking
- Wait multiple.
- Cross-partition event control.

Relocating Loader

The relocating loader, a standard feature of DOS/VS, causes the linkage editor to produce relocatable phases which can then be executed in any partition.

In a system supporting the relocating loader, it is not necessary

- to write an assembler-language self-relocating program, if you want the program to be executable in any partition. The high-level language programmer can thus obtain the advantages of self-relocating programs.
- to link-edit again if the size of the supervisor or the boundaries of the partitions change after a program has been cataloged into the core image library.
- to maintain multiple copies of individual programs in a core image library.

The relocating loader is also advantageous to the operator, who can execute a relocatable phase in any available partition large enough to contain it.

You can exclude the relocating loader from the supervisor by specifying RELLDR=NO in the FOPT macro. However, some DOS/VS options require the relocating loader. Therefore, if you specify OLTEP=YES, RETAIN=YES, RPS=YES, GETVIS=YES, TP=VTAM, or VSAM=YES, the relocating loader is automatically included in your supervisor.

When the supervisor contains the relocating loader and if the phase origin is not an absolute address, the linkage editor automatically produces

a relocatable phase. You can suppress this by specifying ACTION NOREL at link-edit time.

Note: *A supervisor generated without the relocating loader can still load relocatable phases. No relocation is performed, however, and the phase is loaded at the link-edited origin.*

Relocating loader applications are discussed in the section *Link-editing for Execution at Any Address* in *Chapter 6: Linking Programs.*

POWER/VS

POWER/VS provides spooling services for up to four partitions and resides in a virtual partition with a higher priority than that of the partitions it controls. Although POWER/VS runs in virtual mode, it supports programs running in virtual or real mode. POWER/VS will honor a spooling request originating from the SVA; however, the DTF (or CCB), CCWs, and data areas must reside in a partition (and not in the SVA).

Specifying POWER=YES in the SUPVR macro sets up the necessary linkages in the supervisor which are used when POWER/VS is active. The version of POWER/VS distributed in the core image library will suit the needs of many users; however, if you have special requirements, you can assemble the POWER/VS generation macros, which are distributed in the source statement library. Refer to *DOS/VS POWER/VS Installation Guide and Reference.*

Multitasking

Multitasking provides the ability to execute more than one task concurrently in a single partition. Because multitasking presupposes the multiprogramming facilities (for instance, task selection) multitasking is only available in a multiple-partition system.

A program engaged in multitasking consists of one main task, which initiates (attaches) a number of subtasks. The maximum number of subtasks in the sytem depends on the number of partitions specified in the NPARTS parameter: max. number of subtasks = 15 - number of partitions.

These subtasks may reside together in one partition or they may be distributed among the various partitions.

To generate multitasking support (also known as asynchronous processing) in the supervisor, you specify AP=YES in the SUPVR macro.

Cross-Partition Event Control

The cross-partition event control option allows tasks that execute in different partitions to wait upon completion of user-defined events and to signal event completion to each other.

Wait Multiple Option

The wait multiple option allows a task to wait on more than one event. The task regains control on the completion of any one of the events on which it was waiting.

Library Options

You can generate support for private core image libraries, for special applications in the procedure library, and for reserved supervisor space to achieve better fetching performance. These options are described below. No supervisor generation options apply to the relocatable library or to the source statement library. For full details on the type of library for your installation, refer to the section *Planning the Libraries*.

Private Core Image Libraries

Private core image libraries (PCIL) have the same format as and are supplementary to the system core image library.

To include support for private core image libraries in the supervisor, specify PCIL=YES in the FOPT macro. For more information on the creation, organization, and maintenance of private core image libraries, turn to *Chapter 7: Using the Libraries.* Refer also to the section *Second Level Directory for the Core Image Library.*

Extended Support for the Procedure Library

Normally, cataloged procedures can consist of job control statements and/or linkage editor control statements. However, with the extended support, cataloged procedures can also consist of data that is to be read from SYSIPT. Such data, for instance, may be utility control statements to be processed by a utility program.

To include the extended support for the procedure library, specify the SYSFIL parameter in the FOPT macro, which is discussed in the section *System Files on Disk* in this chapter.

More information on the procedure library is contained in the section *Planning the Libraries.*

Second Level Directory for Core Image Libraries

The directory entries for phases in the core image library are sorted by phase name in alphameric sequence. The entries are organized in 256-byte blocks, where the highest phase name in each block serves as the key. The highest key on each track of the core image directory is stored in a second level directory (SLD) in the supervisor. To help ensure good performance when a phase is fetched, the number of entries in the SLD should not be less than the number of tracks used for the core image directory.

Specify the SLD parameter in the FOPT macro if you intend to use more than five tracks for the core image directory entries. Similarly, if private core image libraries are used in the system, specify the PSLD parameter in the FOPT macro. Note that the default value for PSLD is zero, compared to five for the SLD parameter.

Independent Directory Read-in Area

If a phase must be loaded and the phase name is not found in the System Directory List (SDL) or Local Directory List, then the core image directory (in conjunction with the Second Level Directory) is searched to find the location of the phase in the core image library. Normally, the directory blocks are read into the physical transient area, which is scanned for the required entry. If a system error recovery routine is in progress, it resides in the same physical transient area. During this time, the physical transient area cannot be used for directory blocks, or for building the fetch channel programs. This effectively prevents any partition of a higher priority from fetching or loading a program phase until error recovery is complete.

By specifying IDRA=YES in the FOPT macro, an independent directory read-in area is generated in the supervisor for holding directory blocks and fetch channel programs during fetching or loading of core image of phases. IDRA=YES is available only in a multiple-partition system.

Note: *The Local Directory List is similar to the SDL, and may be defined for a partition (via assembler) to improve loading of dynamically called programs.*

Teleprocessing

DOS/VS provides facilities for teleprocessing, the interchange of data between an application in the system and terminals connected by telecommunications lines. These facilities provide the ability to define teleprocessing lines during supervisor generation and to specify one or more access methods for input/output services between an application and terminals.

Teleprocessing devices (terminals) are normally attached to the CPU through transmission control units or communications controllers. In some cases there is a direct local attachment. The control unit must be specified in a DVCGEN macro.

The access methods, defined in the TP parameter of the SUPVR macro instruction, are:

- BTAM (the Basic Telecommunications Access Method)
- QTAM (the Queued Telecommunications Access Method)
- VTAM (the Virtual Telecommunications Access Method).

Except when BTAM is specified for a single-partition system, support for any of these access methods automatically includes support for TP balancing (teleprocessing balancing).

For detailed information on generating and using a teleprocessing access method, refer to the appropriate teleprocessing publications. Teleprocessing users should also pay particular attention to the section *I/O Options* later in this chapter and the section *Balancing Teleprocessing* in *Chapter 9: Designing Programs for Virtual-Mode Execution.*

BTAM

BTAM provides READ, WRITE, and CONTROL macro instructions to control input/output. A WAIT macro instruction is used to synchronize I/O with application program processing.

Applications using BTAM can execute in either virtual or real mode. Users of previous DOS releases must reassemble and catalog BTMOD. If BTMOD and the application program were assembled together, the application program must also be reassembled and re-link edited. To execute BTAM in virtual mode, PFIX=YES must be specified in the FOPT macro. the interval timer. For more information, see the QTAM MCP publication.

QTAM

QTAM provides a way to write one or several application programs using GET and PUT macro instructions to request input/output from a Message Control Program (MCP). This MCP, which you generate using QTAM macro instructions, frees the application (called a Message Processing Program) from I/O processing details required by a BTAM application.

The QTAM MCP and its application programs (MPP) can execute only in real mode, and require two separate partitions. Users of previous DOS releases must reassemble the QTAM MCP.

When support for QTAM is generated in the supervisor, BTAM is also supported.

QTAM requires a special disk extent for messages and, in some cases, the interval timer. For more information, see the QTAM MCP publication.

VTAM

VTAM directs transmission of data between application programs and local or remote terminals, and controls the terminals in a telecommunications network. Because VTAM operates with the IBM 3704 and 3705 Communications Controllers, communications lines and communications controllers need not be considered in coding application programs.

Basic services performed by VTAM include:

- Establishing, terminating, and controlling access between application programs and terminals.
- Moving data between application programs and terminals.

- Permitting application programs to share communications lines, communications controllers, and terminals.

VTAM requires that multitasking support be specified during supervisor generation. Other options automatically generated when VTAM is specified include:

- Support for the use of the STXIT macro instructions (all options) by problem programs.
- Storage management support for the GETVIS and FREEVIS macro instructions.
- Use of the PFIX and PFREE macro instructions for fixing and freeing pages.
- Inclusion of the relocating loader.
- Support for the time-of-day clock.
- Support for the multiple wait function.
- Support for the use of the EXCP macro instruction with the REAL parameter.

Both real and virtual storage must be allocated for the partition in which VTAM is to run. A second partition is required for VTAM application programs. For information on calculating storage requirements for both the VTAM partition and the application program partition, refer to *DOS/VS System Generation.* Other installation details are contained in the *DOS/VS VTAM System Programmmer's Guide.*

ASCII

In addition to processing EBCDIC files, DOS/VS can process magnetic tape files written in ASCII (American National Standard Code for Information Interchange), a 128-character, 7-bit code. The high-order bit in the System/370 8-bit environment is zero. ASCII tape files may be either unlabeled or labeled according to the specifications of the American National Standards Institute, Inc., (ANSI).

ASCII tape files may be processed in any partition. Because internal processing of ASCII files is performed in EBCDIC, the data is translated at I/O time. No user translation tables or instructions are required.

Input files containing ASCII data are translated to EBCDIC as soon as the record is read into the I/O area. Output files described as ASCII are translated from EBCDIC to ASCII just prior to writing the record.

If your system is required to process ASCII files, specify ASCII=YES in the SUPVR macro. This generates the two translation tables needed for the conversion from ASCII to EBCDIC and from EBCDIC to ASCII, in the supervisor.

Job Accounting

The job accounting interface facility provides job and job step information that can be used for charging system use, supervising system operation, planning new applications, etc.

When this option is selected (JA=YES in the FOPT macro), job accounting tables are built in the supervisor to accumulate accounting information. One DOS/VS job accounting table is maintained per partition. The format of these tables is shown in *Chapter 10: Using the Facilities and Options of the Supervisor.*

To utilize this information, you must write a routine to store or print the desired portions of the table. This routine must be cataloged in the core image library under the name $JOBACCT.

If the user I/O routine ($JOBACCT) is written using LIOCS with label processing, the JALIOCS parameter of the FOPT macro must be specified in addition to the JA parameter. JALIOCS indicates that a user save area and a label area in the supervisor are to be reserved. The label area replaces the one normally used by LIOCS label processing routines.

Information on how to write a DOS/VS job accounting routine can be found in *Chapter 10: Using the Facilities and Options of the Supervisor.*

If POWER/VS job accounting is desired, support for the job accounting interface is required. Job accounting interface information and POWER/VS job accounting information are combined in the POWER/VS account file for each partition running under POWER/VS. No user-written data collection routine is necessary. Refer to *DOS/VS POWER/VS Installation Guide and Reference.*

Timer Services

The following timer services are available to DOS/VS users:

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

Both the time-of-day clock and the interval timer are standard hardware features; while the task timer requires other hardware features (the clock comparator and the cpu timer) which are standard on all System/370 models except the 135 and the 145. Utilization of these timer services in DOS/VS also requires software support, for which supervisor generation parameters are provided.

Time-of-Day Clock

The time-of-day (TOD) clock provides a consistent measure of elapsed time suitable for time-of-day indication. You can use the TOD clock to *time-stamp* programs. Regardless of whether or not DOS/VS programming support for the TOD clock is included in the supervisor, programs can inspect the contents of the TOD clock by means of a store clock (STCK) instruction. For more information on the use of this instruction, refer to *IBM System/370 Principles of Operation.*

To include support for the time-of-day clock in the supervisor, specify TOD=YES in the FOPT macro. The time-of-day and the date are then

automatically included with each // JOB and / & job control statement that is printed on SYSLST and/or SYSLOG.

The ZONE parameter in the FOPT macro is associated with the TOD=YES specification. In the ZONE parameter you indicate the difference between Greenwich Mean Time (GMT) and local time in hours and minutes. If the local time to be specified is GMT, the ZONE parameter can be omitted.

During the IPL procedure, if IPL is performed from SYSLOG, a message is printed on the operator console to inform the operator of the status of the date, clock, and zone. If necessary, the operator can correct this information in the SET command.

The TOD clock support also enables programs to issue the GETIME macro instruction, which causes the exact time-of-day to be stored in general register 1. When a GETIME macro instruction is issued, the date fields in the supervisor communications region are updated, if necessary. A description of the use of the GETIME macro instruction is included in the section *Using the Time-of-Day Clock* in *Chapter 10: Using the Facilities and Options of the Supervisor.*

Interval Timer

The interval timer can be used by programs (main tasks and/or subtasks) that need to schedule certain processing based on discrete time intervals. If support for the interval timer is included in the supervisor, and a problem program is written with the appropriate macro instructions and routines, the interval timer causes an external interrupt when the time limit established by the program has elapsed.

To include support for the interval timer in the supervisor, specify the IT parameter in the FOPT macro.

Seven problem program macro instructions relate to interval timer support. These are described in other parts of this manual, as indicated below:

- The section *User Exit Routines* which follows describes the STXIT and EXIT macros in general, and the section *Interval Time Exit* describes their specific use in relation to the SETIME macro.
- *Chapter 10: Using the Facilities and Options of the Supervisor* describes how to implement the SETIME, STXIT, EXIT and TTIMER macros.

Task Timer

The task timer can be used by the main task of the partition owning the task timer to escape from processing and enter an exit routine after a specified period of time. This discrete time interval is decremented only when the main task is executing. If support for the task timer is included in the supervisor, and the owning partition's main task is written with the appropriate macro instructions and routines, the specified task timer routine is entered when the time interval has elapsed.

To include support for the task timer in the supervisor, specify the TTIME parameter in the FOPT macro.

If an exit routine is not specified in the STXIT TT macro, the interrupt is ignored. The SETT macro is used to set the time interval, and that interval can be tested or cancelled by means of the TESTT macro. The EXIT TT macro is used to return control from a task timer exit routine.

Console Buffering

Since there is only one console typewriter in the system and it is a relatively slow device, the entire system can be held up while messages are being issued to the operator. Console buffering support builds a queue of output messages and returns control immediately to the partition requesting the output. The messages are then written as soon as the console becomes available.

Support for console buffering is indicated by the CBF=n parameter in the FOPT macro (where n is the number of I/O requests to be buffered.) At least one buffer should be specified for each partition or task issuing messages so that buffers are available and the task can continue processing while the message is being printed. Five per batched-job partition is recommended. The console buffering is not split per partition, but used by the whole system.

Console buffering is not supported:

- when Models 115, 125, 138, or 148 are used and DOC=125D or DOC=3277 is specified.

User Exit Routines

If required, the supervisor can permit user routines to gain control when any of the following types of events occurs:

- Interval Timer Interrupt (IT)
- Program Check Interrupt (PC)
- Abnormal Termination (AB)
- Operator Communication Interrupt (OC)
- Task Timer Interrupt (TT)
- Page Fault Handling Overlap (PHO)

Both the supervisor and the problem program that contain the user routine must have the proper code to establish an interface. The supervisor part of this interface is specified during system generation with the first five options being specified in the FOPT macro, and the last option in the SUPVR macro.

The problem program that wants to utilize the options must contain code to set up the interface. For the first five events, code can be generated by the STXIT macro. For the last event, code is generated by the SETPFA macro. This code is assembled in the main line of a problem program.

The first operand of the STXIT macro indicates the type of event to be handled. It must have an equivalent in the supervisor. The second and third

operands indicate the addresses of the user routine and its save area. If the second and third operands are missing, this means that an existing interface has to be discontinued. Once the linkage has been established and one of the events occurs, control is passed to the user routine, which takes appropriate action and returns control to the supervisor, either directly or via a termination macro. The direct return can be handled by including the EXIT macro in the user routine. The job termination return can be handled by the CANCEL, EOJ, JDUMP, or DUMP macro; one of these must be used for the abnormal termination condition.

Short descriptions of the support for each of the types of user exit routines follow. For more details refer to *Chapter 10: Using the Facilities and Options of the Supervisor.* For information on how multitasking affects this support and what happens if multiple events coincide, refer to the *DOS/VS Supervisor and I/O Macros*. Some high-level languages offer similar facilities, for details of which see the appropriate programmer's guide.

Interval Timer Exit

Interval timer support is indicated by the IT=parameter of the FOPT macro. If IT=YES is specified, all tasks in all partitions may refer to the interval timer.

Example of how to use the Interval Timer: Suppose you want to take a checkpoint on a job at a certain time after it has started. Include the STXIT and the SETIME macros in your program. The first macro will set up the interface with the supervisor; the second will enable you to set a time interval. When that interval elapses, an interval timer interrupt occurs and control is given to the user routine. Please note that the user routine need not be entered immediately. For instance, if the user routine is in a background partition, and a foreground partition is active, the user routine will not be entered until the background partition becomes active. Chapter 10 contains coded examples of this option.

To find out the time remaining in an interval, a program can issue the TTIMER macro instruction. The supervisor then loads this value in general register 0. This macro can also be used to cancel the remaining time in the interval.

Program Check Exit

If PC=YES is specified in the FOPT macro, programs can establish linkage from the supervisor to a user routine by executing a STXIT macro. If a program check occurs within the program, the supervisor gives control to the user routine instead of discontinuing the program. The user routine can analyze the program check and choose to ignore, to correct, or to accept it. If the check is ignored, control can be given back to the supervisor by executing an EXIT PC macro.

In some cases it may be possible to correct the error condition. For example, if a data exception occurs on an add pack (AP) instruction, the user routine can be written to correct the sign and arrange for the

instruction to be processed again. The user routine can request that processing of the main line program continue via the EXIT macro.

In case the problem cannot be resolved the program check is accepted as valid. The user routine can then terminate further processing of the program by issuing a CANCEL, DUMP, JDUMP, or EOJ macro.

The ability to include a user routine to process program checks can be especially advantageous when using LIOCS. In that case I/O housekeeping such as closing files and freeing tracks can be performed before termination of the job or task.

Abnormal Termination Exit

If AB=YES is specified in the FOPT macro, any program can issue a STXIT AB macro. This instruction allows a user routine to get control from the supervisor before an abnormal end-of-job condition discontinues the processing of the program. The user routine normally ends with one of the termination macros (CANCEL, DUMP, JDUMP or EOJ), to terminate the problem program and to return control to the supervisor, rather than by initiating the continuation of the problem program.

Operator Communications Exit

OC=YES in the FOPT macro supports the use of user routines for handling external interrupts from the operator. This support is useful in a number of applications, for example:

- A change in the environment is needed. A message is then issued by the program. For example: MOUNT TAPE XXX on unit xxx and press the interrupt key.
- In teleprocessing, the OC exit allows the operator to start and stop activities on certain lines or terminals, or to invoke diagnostic procedures. In this case, program run books with explicit instructions may be required to ensure understanding between programmer and operator.

The external interrupt that links to an OC user exit routine can be caused in one of two ways:

- If the program with the OC exit routine is being executed in the background partition, the operator can press the interrupt key on the system console.
- If the program with the OC exit routine is being executed in a foreground partition, the operator can press the request key on the console typewriter. When the attention routine identifier AR appears, he should reply *MSG F1* (or give the appropriate partition identifier).

Task Timer Exit

Task timer support is included by the TTIME= parameter of the FOPT macro. This parameter also identifies the partition owning the task timer.

Only the main task in the owning partition can utilize the task timer.

The time interval is specified in the SETT macro and is decremented only when the task is executing. The exit routine specified in the STXIT TT macro is entered when the interval has elapsed, provided that routine has already been supplied to the supervisor.

To find out the time remaining in an interval, the task can issue a TESTT macro. This causes the time remaining in the interval to be returned in register 0. The task can also issue a TESTT CANCEL to cancel the remaining interval time. In this case the exit routine is not entered.

Page Fault Handling Overlap Exit

If PHO=YES is specified in the SUPVR macro, a user routine can continue processing during the time a page fault is being handled by the system, if this page fault occurs in the same task and not in a supervisor routine invoked by this task. This support is of interest only for programs executed in a virtual partition that make use of user-developed subtasking rather than IBM-supplied multitasking.

Such programs may issue the SETPFA macro instruction to establish linkage from the page management routines in the supervisor to a user routine, called the page fault appendage routine. The SETPFA macro instruction is described in *DOS/VS Supervisor and I/O Macros.*

Disk Options

Options are provided for some DASD devices. These are:

- System files on disk (or diskette)
- DASD file protection
- Track hold option
- Seek separation
- Rotational position sensing
- Block multiplexer channel support.

System Files on Disk or Diskette

The system logical units SYSRDR, SYSIPT, SYSLST, and SYSPCH can be assigned to the following devices: card reader, printer, card punch, magnetic tape, disk, or diskette. When a system logical unit is assigned to a disk, it must have only one extent.

For example, you may want to catalog the output from a language translator to the relocatable library. During the language translation step, SYSPCH could be assigned to a disk extent. The resultant object module would then be cataloged via MAINT by assigning SYSIPT to the same disk extent.

Support for system files on disk or diskette is specified in the SYSFIL parameter of the FOPT macro.

The SYSFIL option also implies extended support for the procedure library. This means that cataloged procedures may contain in-line SYSIPT data. The sets of control statements that can be cataloged into the procedure library are, therefore, not limited to job control or linkage editor control statements. (See also *Extended Support for the Procedure Library.*)

For systems without magnetic tapes, the SYSFIL option is required in order to apply IBM programs and program maintenance, which, in this case, must be distributed on disk packs in SYSIN format.

DASD File Protection

This feature is provided to prevent user programs, which utilize DAM or user-written channel programs for writing onto DASD, from writing data outside of the limits of the DASD file currently being accessed. This might happen if, for example, a randomizing algorithm produces an unexpected DASD address which is outside the file limits.

DASD file protection support is indicated in the DASDFP parameter of the FOPT macro. The parameters indicate that protection is given to channels and device types. If used, DASDFP should be provided for the entire channel range, for instance, DASDFP=(1, 3, 3330).

DASDFP gives protection on the basis of programmer logical units. If two files in the same partition are assigned to the same programmer logical unit, the DASDFP option gives no protection.

Protection begins and ends on a disk cylinder boundary or a data cell strip boundary. Files to be protected should, therefore, begin and end on such boundaries. No protection is given to partially allocated cylinders or strips.

If you are using physical IOCS, you must use the DTFPH macro to define the file. The file must be opened using the OPEN or OPENR macro, and each channel program must commence with a long seek (X'07') command, and contain no chained long seeks.

If you specify DASDFP, the SYSRES file must reside on a protected channel: otherwise, it will not be possible to IPL the system.

DASDFP does not prevent file contention between partitions, or within partitions if the same symbolic unit is used. Thus, more than one partition may access the same file at the same time, and may even attempt to update the same record simultaneously. The track hold option (TRKHLD) is provided to solve this problem. Note, however, that all DASD writes (DAM and otherwise) within the DASDFP range will be checked.

Track Hold Option

The track hold option is used to ensure that if a DASD track is being modified by one task, no other task in the system can access that track provided that they also use track hold. The facility is available for all VSAM, ISAM and ISAM interface program functions (except LOAD), all DAM functions, all SAM work file functions and other SAM update functions. The facility is a combination of supervisor (PIOCS) and LIOCS functions.

The track hold option can be selected by specifying the TRKHLD parameter in the FOPT macro. For non-VSAM files, the DTF must specify HOLD=YES.

For VSAM files, if SHAREOPTION 4 is specified at the time a VSAM file is defined, VSAM uses the track hold facility to ensure file write integrity. Note: Performance may be affected.

If you write your own channel programs, each program must begin with a long seek (X'07') command. If multiple track search channel programs are used, only the first track will be held, which is not necessarily the track on which the record is located.

Deadlock occurs if one task is waiting for a track held by a second task and the second task is waiting for a track held by the first. This can easily be prevented by establishing the convention that every task must be programmed so that it will not hold more than one track at a time. Deadlock may also occur if the maximum number of tracks demanded to be held by all tasks combined exceeds the maximum specified in the TRKHLD parameter.

Seek Separation

A channel program for a DASD device usually consists of a number of functions to perform the I/O operation as follows:

1. A *long seek* to position the access arm over the required cylinder.
2. A search to find the required record on a track on that cylinder.
3. A *transfer in channel* to branch back to the search until the search is completed successfully or unsuccessfully.
4. The actual read or write which transfers the data.

Since the channel is monopolized once the channel program has been initiated, no other device on this channel can be accessed until the data has been transferred. This is inefficient, particularly since most of the time utilized during the execution of a DASD channel program is taken up by the *long seek* (1). With seek separation support, the supervisor handles this by separating the long seek from the rest of its channel program and initiating the seek command separately. The channel is then free while the disk access arm is being moved and other devices on the channel and control unit can be accessed.

Once the access arm has been positioned over the correct cylinder, the rest of the entire channel program is executed. By performing this function in the supervisor, contention is avoided between two tasks trying to move the same disk access arm.

For 3330's and 3340's attached to block multiplexer channels seek separation is handled by the channel. See the section *Block Multiplexer Channel Support* in this chapter.

Specifying SKSEP=YES in the FOPT macro creates seek separation support for each DASD device specified in a DVCGEN macro at supervisor generation time.

Specifying SKSEP=n indicates the number of DASDs to be supported and must not be less than the number of DASDs you specify in DVCGEN macros. Specifying n adds flexibility to your installation by allowing for expansion: seek separation support then also applies to the DASDs added at IPL time.

Rotational Position Sensing

Rotational Position Sensing (RPS) is a feature on all IBM disk storage devices except 2311, 2314, and 2319 (optional feature on IBM 3340, Models A2, B1, and B2). It provides the ability to overlap positioning operations on one device with service requests for other devices on a block multiplexer channel (or its equivalent on Model 3115/3125 CPUs).

Better channel utilization can increase system throughput, especially in large multiprogramming systems with heavy concurrent I/O activity. Because a selector channel is monopolized once a channel program has been initiated, no other device on this channel can be accessed until the data has been transferred. With block multiplexer channels and the RPS feature of DASD devices, however, the device can disconnect from the channel during positioning operations. The channel is then available for other requests so that other devices on the channel can be accessed.

Overlap of positioning to a record on a track requires adding RPS CCWs to the direct access storage device channel programs. DOS/VS system control and service programs that support RPS create these CCWs at execution time provided that the supervisor is generated with RPS=YES and that the direct access storage device has the feature.

RPS support for DOS/VS is provided in all access methods which support RPS DASD devices and in the DOS/VS system control and service programs where the implementation benefits total system performance. Implementation of RPS support in DOS/VS utilizes virtual storage to enable you to use RPS without recompiling or relink-editing your problem programs. The partition GETVIS area is used to generate an extension to the DTF, and the shared virtual area is used to hold the RPS version of the logic modules. Since this implementation requires a partition GETVIS area, programs executing in real mode do not have RPS support for DASD LIOCS functions. If you have specified RPS=YES in the FOPT macro at supervisor generation time, all programs using DASD LIOCS should define a GETVIS area within the partition to enable the access methods to construct RPS channel programs.

The effective use of RPS depends on each channel program's ability to free that channel so that it can service requests for other devices. Programs using DOS/VS DASD LIOCS access methods will have RPS channel programs constructed by the access method provided a GETVIS area is defined in the partition (by using the SIZE parameter of the *EXEC job control statement*) and that sufficient virtual storage is available in the SVA for loading RPS versions of the logic modules. Programs using PIOCS for DASD access have to be recoded to include Set Sector CCWs and to establish arguments for the CCWs. If this is not done, these programs will destroy the effectiveness of RPS by monopolizing the channel.

Specification of RPS=YES forces generation of block multiplexer channel support, which is a prerequisite for RPS support. See section *Block Multiplexer Channel Support* for a further description.

For a more effective use of RPS with SAM, DAM, or ISAM, you should preload frequently used RPS logic modules into the SVA during IPL by specifying them in your System Directory List (SDL). You may determine frequently used modules by using the Fetch/Load Trace facility of PDAIDS. When using Checkpoint/Restart, modules used must be preloaded. Each access method has RPS versions of the logic modules associated with it. These modules reside in the core image library and are not assembled or link-edited with the user's program. However, any user coded logic modules, coded with the RPS=SVA parameter, must be link-edited to the core image library. the RPS modules are then loaded in the SVA either during IPL or dynamically as needed when the file is opened.

Figure 3.5 shows the organization of a user's program running in virtual storage without RPS support.

Figure 3.6 shows how, with RPS support, this organization will be modified at OPEN time to put the DTF extension in the partition GETVIS area. The pointers to the RPS version of the logic module and channel program will be put into the DTF while the non-RPS logic module and channel program addresses will be saved in the DTF extension. The DTF extension will be freed and the pointers restored at CLOSE time.

Figure 3.7 shows that the RPS version of the logic modules can be either in the SVA or in the SVA GETVIS area, or in some combination of both.

SVA storage requirements for RPS are discussed in *Defining the Size of the Shared Virtual Area*, and *Reserving Storage for RPS*.

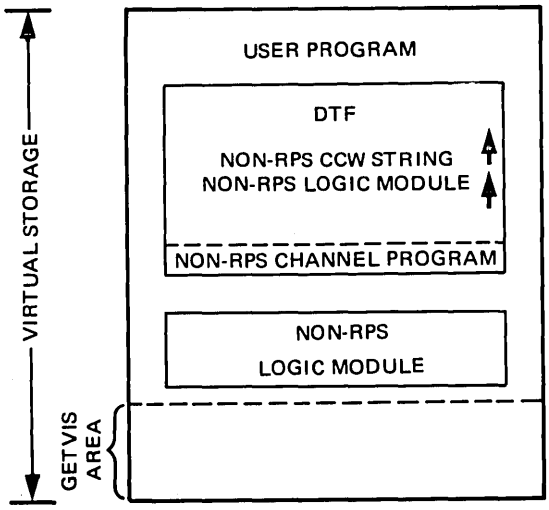


Figure 3.5. User Program Running in Virtual Storage without RPS Support

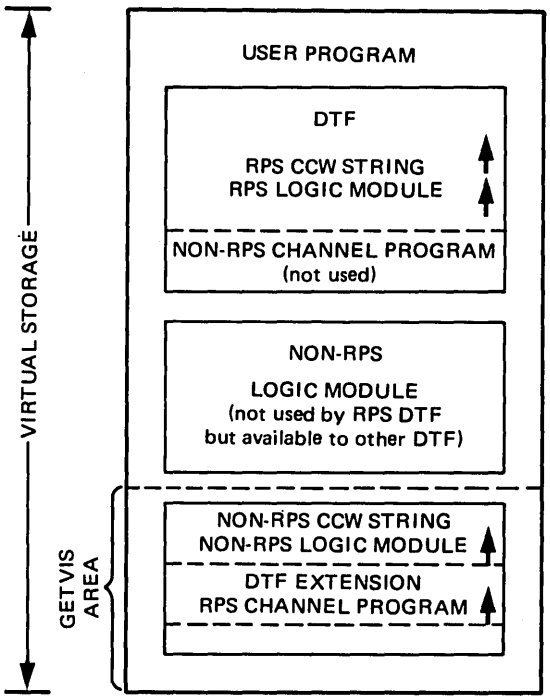


Figure 3.6. User Program Running in Virtual Storage using RPS Versions of Logic Module and Channel Program

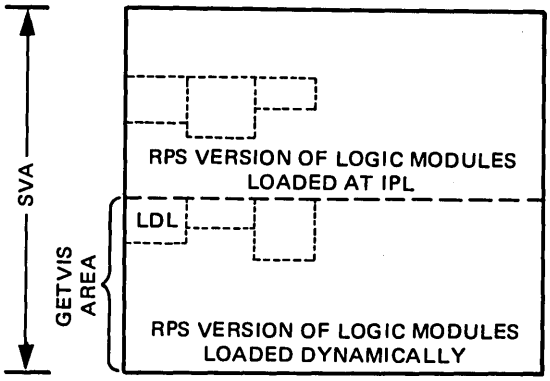


Figure 3.7. Location of RPS Version of Logic Modules

Sequential DASD File Support for 3330-11 and 3350

Sequential or direct-access files can be created and accessed on these devices in the same way as on other IBM DASDs if the pertinent program has so been written.

However, to use these devices also with programs that were written to create and access sequential and direct-access files on other IBM DASD types (such as 2314 or 3330-1), specify for supervisor generation RPS=YES. This enables DOS/VS to write on and read from a 3330-11 or 3350 also for programs that reference different IBM DASD types without a need for recompiling or reassembling these programs.

Block Multiplexer Channel Support

Block multiplexer channel support is useful in configurations with 3330, 3340, and 3350 DASD devices that are attached to block multiplexer channels. To obtain block multiplexer support, specify BLKMPX=YES in the PIOCS macro during supervisor generation.

In a DASD configuration that consists only of 3330, 3340, and 3350 devices, there is no need to request seek separation support since the block multiplexer support provides channel overlap during seeks in a more efficient way. Furthermore, the code generated by a specification of SKSEP=YES is bypassed for these devices if BLKMPX=YES is specified. You cannot have block multiplexer channel support if you are planning to use the 2311 or 2314 compatibility features and your CPU is a Model 115 or Model 125. If your CPU is a Model 135 or 138 block multiplexer support may be specified. This support will be inoperative for files being handled by the Emulator, but it will work properly for files being addressed in native mode.

I/O Options

Defining the Number of CCW Translation Buffers

Because all addresses associated with instructions and data are virtual, they are translated to real addresses before they are actually used. All addresses except those in channel programs are translated by the DAT facility; addresses of I/O areas and channel programs, including channel command blocks (CCBs) and channel command words (CCWs) are translated by DOS/VS. The translation for channel programs is done using buffers (copy blocks) in the supervisor area. Because this software address translation may be time consuming for repetitive I/O requests, the fast translation option may be specified (FASTTR=YES in FOPT macro). This option, if selected, causes DOS/VS to retain and reuse the translated channel programs in the copy blocks and the associated fixed I/O areas. However, when there are no available copy blocks and/or the paging rate reaches the threshold (page pool is too small), those saved copy blocks and fixed I/O areas are released (least recently used first).

Specification of FASTTR=YES may cause degradation of performance when CICS accesses SAM, ISAM, and DAM files.

The required number of CCW translation buffers (specified with the BUFSIZE operand of the VSTAB macro) generally depends on the number of channel queue entries and on the number of CCWs in the channel program. If the number of buffers is too small, overall performance degradation will occur because tasks are put into the wait state until buffer space is available. On the other hand, too large a value for BUFSIZE wastes storage.

If you expect that most of the I/O requests will be made from virtual-mode programs, the number of buffers specified in the BUFSIZE operand should be three times the number of entries in the channel queue for normal CCW translation. Fast CCW translation needs more buffers, and you should

specify six buffers for each channel queue entry. If you expect to do much I/O from real-mode programs, the number of buffers should be reduced proportionally. If ISAM is the predominant access method, about 20% more buffers should be specified. If RPS is specified, about 20% more buffers should also be specified. At least 40 additional buffers should be specified when VSAM is used. If teleprocessing terminals are supported under BTAM or if the fast CCW translation option (FASTTR) is specified, read the description of the BUFSIZE parameter of the VSTAB macro in DOS/VS System Generation.

In order to determine if the number of copy blocks are sufficient, refer to *Channel Queue* for a similar procedure.

Bypassing System CCW Translation

In most instances, double buffering techniques and an increase in block size can significantly reduce the system overhead associated with channel program translation. However, in extreme cases, you may wish to perform your own translation of channel programs and thereby avoid system CCW translation overhead. Programs that might require this are EXCP programs that have very high start I/O rates and that repeatedly use the same channel programs.

By specifying ECPREAL=YES during supervisor generation you obtain support that assists in the translation of channel programs. This support allows you to use the VIRTAD and REALAD macros as well as the REAL parameter of the EXCP macro. You must obtain real storage by means of the PFIX macro and then translate the channel program. The CCB must have the REAL operand. For detailed information see *DOS/VS Supervisor and I/O Macros.*

Channel Queue

The channel queue (CHANQ) is used by the supervisor to schedule I/O operations. An entry is made in the channel queue whenever a request is made for an I/O operation and the entry remains until the operation has completed. Thus, at any point in time, the queue will consist of entries for I/O operations in progress and I/O operations waiting for initiation. Whenever an I/O event completes, the queue is examined cyclically to see if another entry exists for the channel, and if so, the operation is initiated.

The number of channel queue entries to be reserved in the supervisor can be specified in the CHANQ parameter of the IOTAB macro.

The number of occupied entries in the channel queue depends on the activity in the system. No accurate formulas for determining the optimum size can be given though.

The thing to bear in mind is that specifying too small a channel queue will cause performance degradation, too large a CHANQ value will waste storage space (8 bytes per entry).

Real-mode tasks or programs that request an I/O operation when the channel queue is full will be set to reissue the request until an entry

becomes free. Virtual-mode tasks or programs that request an I/O operation when the channel queue is full will be set in the wait state until an entry becomes free.

To avoid performance degradation it is better initially to specify ample channel queue space, and reduce the allotted space later, if desired. The *rule-of-thumb* to be followed is:

- At least one queue entry should be available for each I/O request that can be issued concurrently (open files per jobstep per partition).
- Specify one entry for the SYSRES file and one for the page data set (SYSVIS).
- Specify one entry for each task or partition in the system.
- Specify one entry for each console buffer in the system.
- If multiple volume files are used on the system, specify one entry for each file being accessed at the same time.
- Add two entries per tape drive.
- One entry should be specified for each teleprocessing line that could solicit input. If IBM 2260 local or 3270 local video display units are to be supported by BTAM one CHANQ entry should be specified for each display.
- Add five entries to the total for contingencies.

When the system has been generated, run the programs that make the heaviest use of logical I/O units in the system. If a multiple-partition system, run as many programs as represent the heaviest work load; in particular, run any teleprocessing programs. Then, before the next IPL, obtain a dump of the channel queue (by using the DUMP command or the standalone program generated by DUMPGEN). The channel queue location and format, as well as the use of the DUMP command and DUMPGEN are fully described in *DOS/VS Serviceability Aids and Debugging Procedures.*

An analysis of the channel queue should show that entries near the beginning of the table have been used, whereas those near the end are unused. Although the unused entries are normally redundant, a few surplus entries should be retained to allow for exceptional cases. If all the entries have been used, then the channel queue was almost certainly too small, and a process of experimentation will show the correct size.

Error Queue

The error queue option is of value to installations employing large numbers of I/O devices, for instance, teleprocessing systems. The ERRQ parameter allows you to specify the number the error queue entries within the error recovery block of the supervisor. These entries are used to record information on I/O device errors, and is used by the ERP and RMSR routines. The normal default value is five entries for a multiprogramming system, but in ERRQ you can specify up to 25. Each entry is 44 bytes.

Reliability/Availability/Serviceability

IBM provides software routines that analyze and record CPU, channel, and device errors and attempt to recover from them. The data is stored on the system recorder file (SYSREC). The information obtained from this file serves not only as an aid in diagnosing machine errors, but also helps IBM customer engineers to increase reliability, availability and serviceability (RAS) of your system.

If on-line recovery is impossible, the system may be placed in a hard wait state. A message is then issued to the system operator to run either the SEREP or EREP program to obtain the diagnostic data.

On the IBM System/370 Models 115 and 125, errors in the CPU and natively attached input/output devices (for example, card reader/punch, disks and printer) are recorded on the service diskette (see note).This hardware error recording is independent of the software routines. The recorded hardware statistics may be obtained on the video display unit (DOC), on advice of the IBM CE, through the LOG ANALYSIS displays. Error recovery for channel-attached input/output devices for these CPU models requires the use of software routines with error recording on SYSREC. The information covered here introduces RMS, OLTEP and PDAIDS. Since SDAIDS and OLTEP do not require supervisor generation macros, these topics are covered in detail in *DOS/VS Serviceability Aids and Debugging Procedures*, which contains extensive information about the various RAS features discussed below.

Note: *IBM System/370 Model 158 has a similar hardware error recording feature in addition to software error recording facility.*

Recovery Management Support

These routines, referred to as Recovery Management Support (RMS), are standard for all System/370 models, except for the Models 115 and 125. For these models, specify the RMS, MCH, or CHAN parameters to obtain the RMS support of your choice.

If full RMS support is included (RMS=YES is specified or forced for models 135 and above), the following RAS facilities are provided:

- Machine Check Analysis and Recovery (MCAR)
- Channel Check Handler (CCH)
- I/O device Error Recovery Procedures (ERP)
- Recovery Management Support Recorder (RMSR)

Device ERP routines are standard for all CPU models. The first three facilities provide hardware error analysis and attempt recovery, while RMSR provides for recording of error and operational statistics on SYSREC as follows:

- Machine Check (CPU)
- Channel Check
- Unit check
- Tape/disk error statistics by volume
- MDR (Miscellaneous Data Recorder)
- IPL information
- End-of-Day statistics held in main storage

For models 115 and 125, if full RMS support is not desired, RMSR support for channel attached devices, tape units, and TP devices must be included by specifying CHAN=YES and RMS=NO. RMSR support for MCAR and CCH is provided by specifying MCH=YES and RMS=NO. Specification of RMS=NO, CHAN=NO, and MCH=NO will cause the system to enter a hard wait on the occurrence of a hardware failure with no recording on SYSREC. However, the service diskette will contain error recordings for the CPU and natively attached devices. If your installation plans to use DASD switching, RMS=YES must be specified.

RMS has several options that must be specified, in addition, during supervisor generation if they are desired. These options involve the reliability data extractor, tape error statistics and error volume analysis.

Reliability Data Extractor. If included in addition to RMSR support in the supervisor, the reliability data extractor (RDE) enables data about the IPL procedure to be recorded on the system recorder file (SYSREC). This option requests the operator, when he performs an IPL, to enter the reason for the IPL. This data alerts IBM and installation management to recurring machine errors or other operational problems.

If RDE support is desired, specify ERRLOG=RDE in the SUPVR macro. More information on RDE is included in this manual in the section *Entering RDE Data* in *Chapter 4: Starting the System.*

Tape Error Statistics. As a standard feature the DOS/VS system gathers tape error statistics. This information is accumulated in the PUB2 table for each tape unit and stored in the system recorder file SYSREC (if RMSR support is included in the supervisor). For tapes with standard labels the information is accumulated and stored per volume. When error statistics are required to enable the monitoring of nonstandard or unlabeled tapes, the TEBV parameter of the FOPT macro gives you two options: the parameter can be specified as IR (individual recording) or as CR (combined recording). IR refers to the accumulation of error statistics between two consecutive OPENs on the same tape unit. CR refers to the accumulation of error statistics on the same tape unit until a standard labeled tape is opened on that unit or until a ROD-command is issued. When error statistics are required to monitor the IBM 2495 cartridge reader, the TEB parameter in the FOPT macro must be specified.

Error Volume Analysis. This option of RMS enables you to specify the number of temporary read/write errors that occur on a tape volume to be specified before an informatory message is printed on SYSLOG. The threshold value of temporary read/write errors is specified in the EVA parameter of the FOPT macro. This option is not applicable if RMSR support is not included in the supervisor.

OLTEP

The On-line Test Executive Program (OLTEP) gives the IBM customer engineer the opportunity to test whether the I/O devices attached to the CPU are in working order. OLTEP runs in real mode in the background partition and can run concurrently with user jobs in other partitions.

OLTEP=YES is the default value in the FOPT macro. If you do not want support for OLTEP, specify OLTEP=NO.

The RETAIN function of OLTEP enables the IBM customer engineer to execute OLTEP from a location remote from the CPU. The RETAIN function is available only in the United States of America and Canada. RETAIN is provided only with Models 145, 145-3, 148, 155-II, and 158 and requires that the 2955 Data Adapter Unit be attached to the CPU.

To generate support for RETAIN in the supervisor specify RETAIN=YES in the FOPT macro.

Problem Determination Aids

Problem determination aids (PDAIDS) can be used to assist the programmer in debugging his program. Six trace routines and a dump routine are included in the PDAIDS:

- Input/Output trace
- FETCH/LOAD trace
- Generalized supervisor call (SVC) trace
- QTAM trace
- VTAM trace
- VTAM buffer pool trace
- Transient dump.

Because these routines are executed within the supervisor, the PD parameter in the FOPT macro must be specified. The PD parameter reserves an area in the supervisor for the use of the trace routines.

In addition to the trace and dump routines, PDAIDS contains a program to display and modify object code in a core image library, thereby facilitating the application of quick fixes until a permanent fix can be made by modifying and recompiling source statements. The PD parameter in the FOPT macro need not be specified to use this program.

Defining the System/370 Configuration

During supervisor generation you must specify various macros that relate to the central processing unit, whether programs written for execution on another system may be run on this model, the I/O devices installed (or planned to be installed), and other macros that indicate the standard job control settings for the installation.

Central Processing Unit

In the MODEL parameter of the CONFG macro, you must specify which model of the System/370 line of central processing units is to be used. If you plan to run your generated system on more than one CPU model, you should specify the larger model.

If you specify MODEL=115 or MODEL=125 in the CONFG macro, support for the video display keyboard console (DOC=125D) is always included.

For reasons of system portability, you may wish to specify DOC=125D for larger models. The larger model will then operate in 3210/3215 mode. If, on a CPU model other than 115 or 125, a 3277 is used as operator

console, specify DOC=3277. When you specify MODEL=138 or MODEL 148, the default used for system generation is DOC=3277.

I/O Devices

The supervisor generation macros that relate to the I/O devices attached to the CPU that are described below are: PIOCS, IOTAB, and DVCGEN.

The PIOCS macro defines the configuration requirements to be supported by IOCS. The associated parameters involve the channel switching, specific tape and disk device support, and the use of burst mode devices on the byte multiplexer channel. No distinction is made between 7- and 9-track tapes.

The IOTAB macro, in general, defines the area for the necessary device tables for the system. The parameters involved refer to:

- The number of programmer logical units for each partition defined by the NPARTS parameter in the SUPVR macro.
- The number of job information blocks for the system. (One is required whenever a temporary assignment is made, see *Chapter 5: Controlling Jobs.* Extra JIBs are required if DASDFP is specified.)
- The number of DASD devices (2311, 2314, 2321, 3330, 3333, 3340, and 3350).
- The number of tape devices (2400-series, 3410, and 3420).
- The number of TP devices.
- The estimated number of physical I/O devices.

The DVCGEN macro defines each physical input and output unit in terms of their channel and unit address, device type, whether channel is switchable, and (if applicable) their mode. One DVCGEN macro instruction must be used for each unit on the system. Each individual drive of a 2314/2319, 3333/3330, 3340, or 3350 needs a DVCGEN macro. The total number of DVCGEN macros must not exceed the total number of devices specified in the IODEV parameter of the IOTAB macro. Note that if one physical spindle contains two or more logical spindles, DVCGEN macro instructions must be issued for each of these logical spindles. Device generation by the DVCGEN macro can be changed with ADD and/or DEL commands at IPL time. Refer to the section *Changing I/O Device Assignments* in *Chapter 4: Starting the System.*

Emulators

Through emulation, a program can be run on a machine series other than that for which it was designed. The emulator program, serving as the interface between the user program and the DOS/VS supervisor, runs together with the user program in the same partition, in either a single-partition or multiple-partition environment. In a multiple-partition environment, several emulators can be executed concurrently. One exception, however, is the Model 125, which cannot execute two 1400-series emulator jobs concurrently. For both a Model 20 and a 14xx emulator on a Model 125, RPQ SU002 is required.

Tape reading and writing on 1400-series machines can operate with odd or even parity checking. If you will be using a 1400-series emulator and mixed-parity tape processing, you must specify EU=YES in the SUPVR macro. If you do not use mixed-parity tape processing, you need not specify EU=YES.

Prior to executing emulator jobs, you must generate the emulator program and catalog it into the core image library. This can be done when the system is generated or at a later time.

Further information on the emulator programs is contained in the following publications:

- *1401/1440/1460 DOS/VS Emulator on System/370.*
- *1410/7010 DOS/VS Emulator on System/370.*
- *Model 20 DOS/VS Emulator on System/370.*

Standard Job Control Settings

Each time a programmer submits a job to be executed, he includes job control statements that define the beginning and end of his job and all the physical or logical requirements or options associated with the job. If certain job control settings are agreed upon within an installation and made *standard* during supervisor generation, the programmer need not provide a lengthy OPTION job control statement for each job submitted. If a given job requires different settings from those that are standard, the // OPTION card can be used to override the standard settings for the duration of that job.

The job control settings that can be defined as standard include: whether a dump is desired if an abnormal termination occurs, whether language translators are to list source module diagnostics or to produce an object deck, and whether a symbolic cross-reference list is desired from the assembler or ANS COBOL, etc.

These job control settings are specified in individual parameters of the STDJC macro.

Another macro that deals with standard job control settings is ASSGN. This macro establishes standard job control associations between symbolic device names and physical I/O devices. If multiple assignments within one job stream are made for a single logical unit, only the last assignment for that logical unit is valid: the rest are ignored. These standard assignments can be overridden for the duration of a job via the // ASSGN job control statement or for the duration until the next IPL via the ASSGN job control command (no //).

Standard assignments may be established for all programmer logical units and all of the system logical units, except the following: SYSRES (which is established during the IPL procedure), SYSVIS (which is established via the DPD macro during supervisor generation or the DPD command during IPL), SYSIN, SYSOUT, and SYSCLB (the latter three during job control execution).

These standard assignments are supplemented in the system by cataloging disk and tape labels to the various system and partition standard

label tracks. This relieves the programmer of having to supply this label information for regular jobs such as compilations and linkage editor functions. (Refer to *Chapter 5: Controlling Jobs* for the details on how this is done.)

End of Supervisor

The last macro instruction supplied during supervisor generation must be the SEND macro, which may indicate the address of the end of the supervisor (or more accurately, the requested starting address of the real storage to be used by problem programs).

Regardless of your particular supervisor configuration, the SEND address can be calculated internally. If you have previously assembled a DOS supervisor (previous to DOS/VS), you may still of course calculate the size of the supervisor and round the value up to the nearest 2048 bytes (2K). However, keep in mind that storage protection is a standard feature on all models of the System/370, and therefore:

- The SEND address is always a multiple of 2K bytes.
- The address you specify in the SEND macro is compared with the actual size of your generated supervisor, so that the calculated address never overlaps any part of the supervisor.
- If no address is specified in the SEND macro, the default is the lowest address possible (that is, the minimum space to contain the generated supervisor plus 1, and rounded up to the nearest 2K bytes, if necessary).

Planning the Libraries

The components of the DOS/VS system are shipped in four system libraries: the core image library, the relocatable library, the source statement library, and the procedure library. Most programs and procedures developed and used by your installation will also be stored in these libraries. In addition to the system libraries, DOS/VS supports private libraries which you can use to either substitute for or supplement the corresponding system libraries.

Planning the size, contents, and location of these libraries according to the needs of your installation is an essential part of the system generation procedure. Such detailed planning will ensure that:

- No disk space is wasted by components not required in your installation.
- The libraries are large enough to allow for future additions.
- The libraries are accessed by the system with maximum efficiency.

Following a brief description of the purpose and contents of the individual libraries, this section discusses the three major considerations involved in tailoring the libraries to the needs of your installation. These considerations are:

1. Which libraries are required.

2. How many disk drives are available and where on these devices should the individual libraries be placed.
3. How large should each of the libraries be and what should they contain.

Note that this section is intended to give only general guidance for planning the libraries. More details are contained in *DOS/VS System Generation* and *DOS/VS System Control Statements*. How to change the size of a library, how to insert elements into or delete elements from a library, and how to create private libraries is described in *Chapter 7: Using the Libraries.*

Purpose and Contents of the Libraries

The following is a brief summary of the purpose and contents of the DOS/VS system and private libraries.

The Core Image Library

The core image library contains system and user programs (phases) ready for execution. Each program phase must first be placed in a core image library by the linkage editor program. (The structure of a program in the core image library is described in *Chapter 6: Linking Programs.*)

The Relocatable Library

The relocatable library contains object modules in relocatable form. These object modules are the output of the language translator programs (assemblers and compilers).

The purpose of the relocatable library is to allow you to maintain frequently-used object modules in the library and combine them with other modules without requiring recompilation. The modules from the relocatable library must be processed by the linkage editor and stored in the core image library before they can be executed.

The Source Statement Library

The elements in the source statement library are called books. A book is either a sequence of source statements or a macro definition.

You can catalog into the source statement library sets of source statements that are used by more than one program, and then include these statements in your source program by specifying a COPY (assembler and COBOL) or %INCLUDE (PL/I) statement.

The macro definitions in the source statement library include those macros supplied by IBM as well as any others which you have written and cataloged yourself. When you issue a macro instruction in your program, the corresponding macro definition is retrieved from the source statement library and included in your program according to the parameters you specified.

Each book in the source statement library is classified as belonging to a specific sublibrary; for example, an assembler, a PL/I, or a COBOL sublibrary. Sublibraries are identified by a one-letter prefix added to the book name. Letters A through I and the letter Z are reserved for sublibraries containing system components. You can use the letters J through Y, the digits 0 through 9, and the special characters $, & , and #, to define your own sublibraries.

Classifying books by a sublibrary prefix allows a program, for example written in COBOL, to have the same name as a program written in assembler language, or for two COBOL programs to have the same name. A book is defined to belong to a certain sublibrary at the time it is cataloged into the source statement library.

The Procedure Library

Frequently-used sets of control statements can be cataloged into the procedure library. The elements of the procedure library, called cataloged procedures, can consist of job control statements and/or SYSIPT data. Included POWER/VS JECL statements will be treated as DOS/VS comment statements. If extended procedure support was included during supervisor generation (by specifying the SYSFIL option) you can also catalog procedures containing data that is to be read from SYSIPT under control of the device-independent sequential IOCS, by your program or by IBM-supplied service programs and language translators. SYSIPT in-line data can be, for example, the control statements processed by the librarian or the sort/merge program. Cataloged procedures are retrieved from the procedure library by a special form of the EXEC job control statement.

Private Libraries

Private libraries can be defined for the core image, relocatable, and source statement libraries. The procedure library is supported as a system library only. You can use private libraries to either replace or supplement the corresponding system libraries.

Private core image libraries (PCIL) have the same format as and are supplementary to the system core image library. A private core image library can be used:

- During maintenance or development of operational programs. You can catalog the copy of the program that you are altering to a PCIL with the same name as the operational version in the system core image library.
- To preserve security of operational programs, they may be cataloged into PCIL which is controlled exclusively by the operations department.
- In a multiple-partition system, allocation of PCILs on separate volumes can relieve disk arm contention on the SYSRES volume.
- If the linkage editor is to be used to catalog into a PCIL in a foreground partition. Then that PCIL must be exclusively assigned to that partition.

A private core image library is created by the librarian program CORGZ and is not located on the system residence (SYSRES) extent. The private core image library extent (associated with the logical name SYSCLB) can reside on any disk volume that is supported by DOS/VS. Multiple private core image libraries can reside on one volume or they can be created on separate volumes. They can be created on the same volume as SYSRES, but this is not recommended unless the access level is low. SYSCLB can only be assigned permanently (not temporarily) and is not acceptable as a standard assignment during supervisor generation.

Choosing the Libraries for an Installation

In as well as executable user programs an operational DOS/VS system all system components (supervisor, job control program, linkage editor, etc.) must reside in the system core image library. Therefore, a system core image library must be present in every DOS/VS installation. Which of the other libraries you need depends largely on the type and amount of work to be done and the resources available at your installation. The following discussion of the advantages and possible applications of the individual libraries is intended to assist you in selecting a set of libraries that will help guarantee optimum performance of your system.

Relocatable and Source Statement Libraries

Although these libraries are optional, few installations can operate efficiently without them. If, for example, you work with a PL/I compiler and you need to have the PL/I resident library routines on-line at all times, these routines must be in the relocatable library. (The only -- and very inefficient -- alternative would be to include the physical card decks for such modules in-line with the linkage editor input.) Similarly, when you assemble programs that use IBM-supplied macros the corresponding macro definitions *must* be present in the source statement library.

The same advantages as those gained by having IBM-supplied modules in a library can of course be obtained if you store your own object modules or source statement books in a relocatable or source statement library. The more information you have on-line in a library the less card handling is required and the more efficient your system will operate. Because the disk space available to the libraries is limited, you may prefer to reduce the contents of the relocatable and source statement libraries to a minimum to allow for sufficient space for the core image library. If additional disk drives are available, the space problem can be solved by creating private libraries (see *Private Libraries,* later in this section.)

Procedure Library

In most data processing installations there are a number of programs that are frequently executed. An inventory control program, for instance, may have to be run daily or weekly. Or a payroll program may have to be executed weekly or monthly. These programs are probably used for a long period of time without being changed.

For each of these programs, there would be one or more sets of job control statements, which the programmer prepared and tested when the program was first run. These sets of job control statements can be cataloged in a procedure library and then, to retrieve a set, only one statement is required.

This minimizes repetitive operator handling (which often includes the replacement of defective cards and reinsertion of diskettes), and reduces machine time and errors.

A cataloged procedure is exactly the same as what is described above as a fixed set of job control statements. But the individual procedures are no longer collected by the operator and selected manually for use; instead, they are cataloged in card image format in the procedure library, from where they can be retrieved through a special form of the EXEC job control statement or operator command. Cataloged procedures can be modified as they are retrieved from the library.

Refer to *Chapter 7: Using the Libraries* for information on how to create and maintain (catalog, delete, etc.) a procedure library. The use of cataloged procedures (retrieving and modifying) is discussed in *Chapter 5: Controlling Jobs.*

Private Libraries

You can establish private relocatable or source statement libraries either to supplement or to replace the system libraries on the SYSRES file, thereby extending the space available to the system core image library. Conversely, you can reduce the size of the system core image library by cataloging selected programs in a private core image library.

Private libraries are also useful in a testing environment where you can keep working copies of your programs intact on a system library while you test modifications of the same programs on a private library. Private libraries can thus add a great deal of flexibility to your system.

You may define as many private core image, relocatable, and source statement libraries as desired, each serving a particular purpose. For instance, having a separate core image library for each partition, each on a separate disk drive, would reduce the disk arm movements on the SYSRES volume, which means faster access to the libraries. Be careful, however, not to have too many private libraries in your installation because of the additional maintenance required. Also, if each programmer were allowed to have his own private library, the total time spent by the operator in mounting and dismounting disks might exceed the execution time of the program.

To be able to use a private core image library the PCIL option must have been specified when the supervisor was generated. The PCIL option, and other special considerations concerning the planning of private core image libraries are discussed under *Tailoring the Supervisor,* earlier in this chapter.

Note: *Private relocatable and private source statement library are restricted to the same device type as the SYSRES device; the private core image can be on a different device type than the SYSRES.*

Determining the Location of the Libraries

Having decided which libraries you want in your system, you must determine where on the available devices these libraries are to be placed. All system libraries must reside in the SYSRES extent of the system disk pack in a predefined sequence (see Figure 3.8). Although it is theoretically possible to have private libraries on the system pack (outside the SYSRES extent), this is not recommended because it involves increased movement of the disk arm.

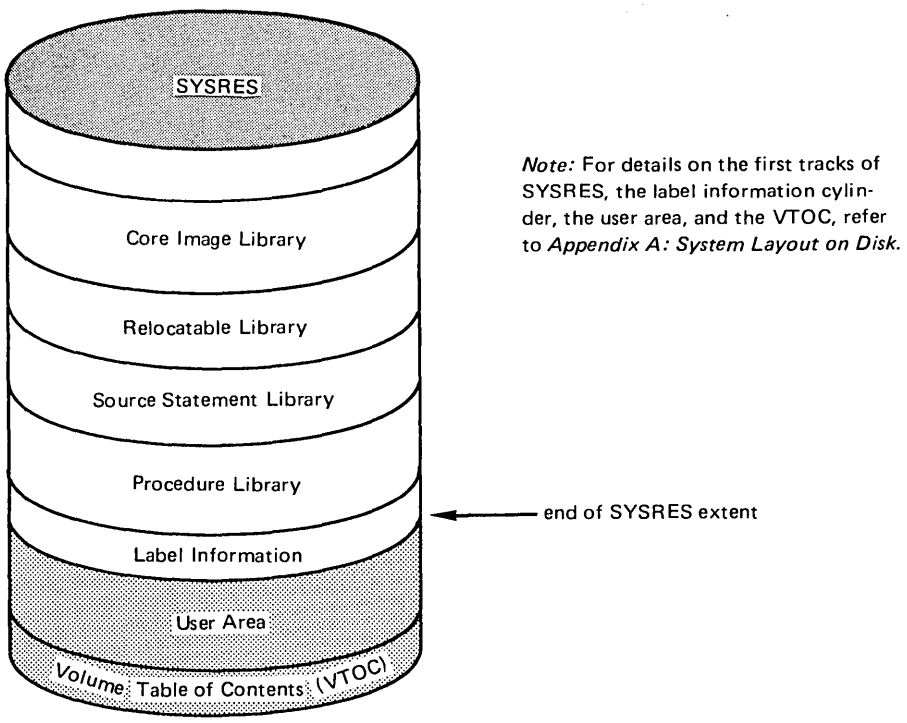


Figure 3.8. The Relative Location of the Four System Libraries

The directory area for each library is not shown in the figure. By definition, all system libraries reside on the system residence file (SYSRES). If you have additional disk drives, you can define private core image, relocatable, and/or source statement libraries on the extra volumes. Private relocatable and private source statement library volumes must be of the same type as the SYSRES pack. Private core image libraries can be on any disk device type supported by DOS/VS. The system relocatable and system source statement libraries can be removed from SYSRES and established as private libraries; the system core image library, however, must always be present on SYSRES. It can be supplemented but not replaced by a private core image library. The procedure library is supported only as a system library; you cannot create a private procedure library.

Figure 3.9 shows two examples of how you can organize the libraries in a system with three disk drives. Any other combination of libraries on the available devices in possible.

The examples in Figure 3.9 are to demonstrate that you can distribute your private libraries among the available devices as desired. A more practical example of how you can organize your libraries is given in Figure 3.10. The example assumes a system with three disk drives, but it is also applicable if you have only two or more than three drives. The organization of the libraries in this example is especially useful when you need large amounts of data on-line during execution.

Planning the Size and Contents of the Libraries

When planning the libraries for an operational system, you must decide on their precise contents and size for daily use. Although you can change the size of your system libraries at any time after system generation (by means of the librarian programs), you should try to anticipate future space requirements and, if possible, provide this space. Such detailed planning can eliminate the need for a complete reorganization of the libraries which would be necessary if the extension of a library results in an overflow on the disk pack. Careful planning of the private libraries will save you additional work because you cannot redefine the extents of a private library once it has been created. To change the size of a private library you must create a new private library and copy the contents of the old library into it. Consider the following factors before deciding on the contents and size of the libraries:

- The average size of a program in your installation.
- The number of programs you want on-line.
- The amount of space available.

The core image library, for example, is the library in which you will keep most of your programs. (Otherwise, each program must be submitted to the linkage editor and placed in the core image library temporarily before it can be executed.) Therefore, ensure that your core image library is large enough to accommodate all programs that must be resident and on line; this includes your own programs as well as IBM-supplied components.

Special considerations apply when you work with an on-line private core image library:

- Program phases starting with $ could be in a private core image library, but it is more efficient to keep them in the system core image library. When a $ phase is required, the system first searches the system core image library and, if it does not find the phase, it then searches the assigned private core image library.
- For all other phases (not beginning with $), first the private and then the system core image library is searched; thus, if you work with a private core image library, search time is reduced for these phases cataloged in the private core image library.

To plan the contents and size of the relocatable library, determine which of the IBM-supplied modules can be deleted and how much space you need to store your own object modules on-line. For any modules you wish to retain in relocatable form, you can copy them onto a backup disk and delete them from the operational pack.

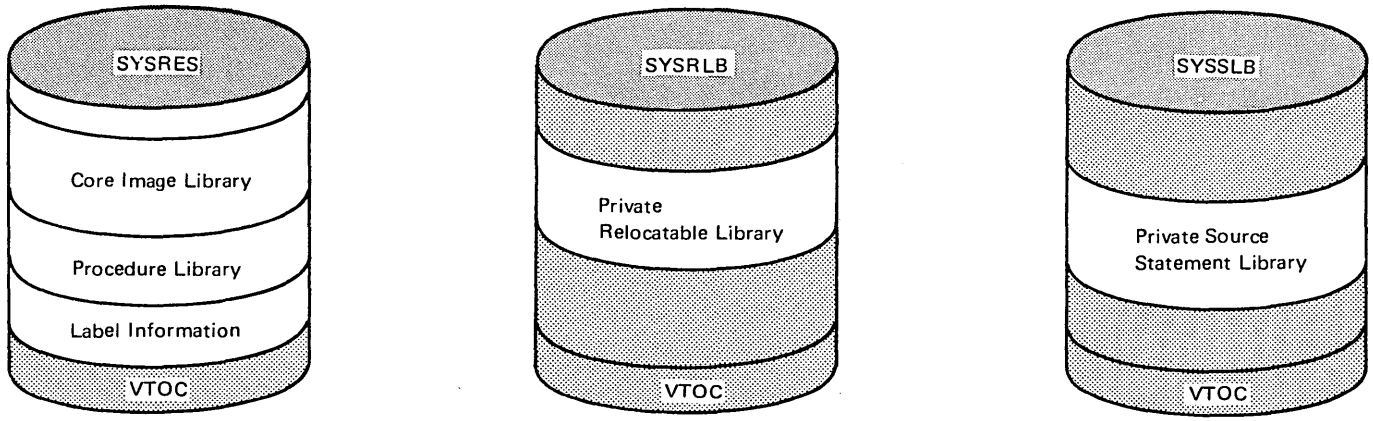
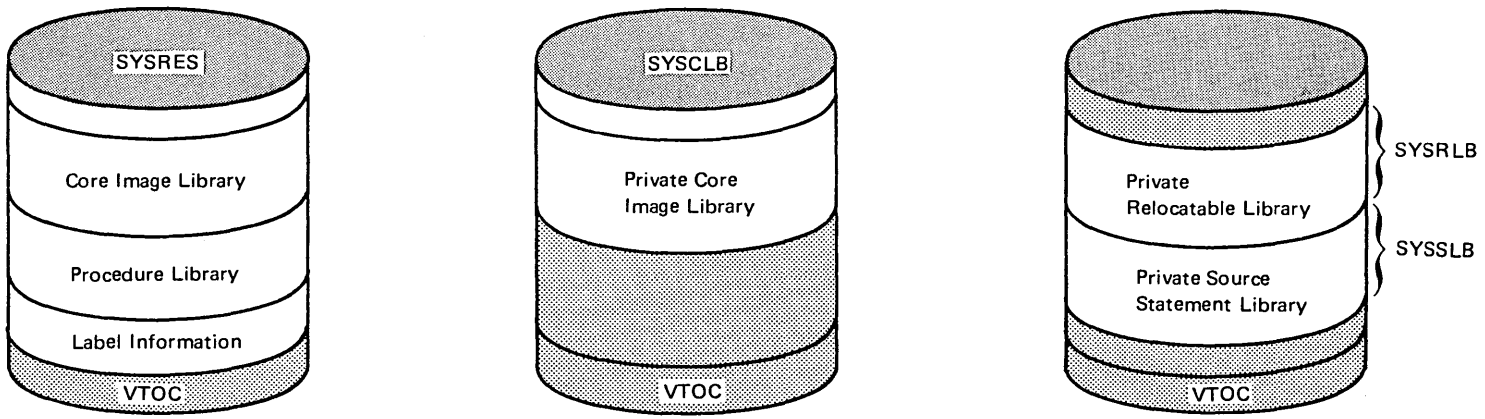


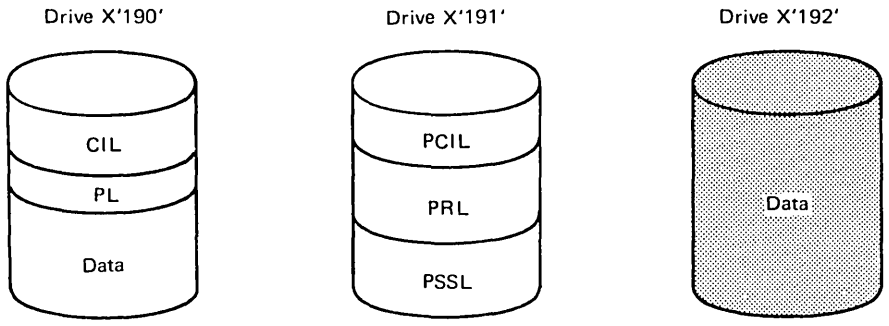
Figure 3.9. Alternative Locations of the Libraries

If a private relocatable library and a private source statement library are to *replace* the corresponding system library, the core image library directly precedes the procedure library. These private libraries can also be used to supplement the system relocatable and source statement libraries, in which case the SYSRES file would appear exactly as shown in Figure 3.6.



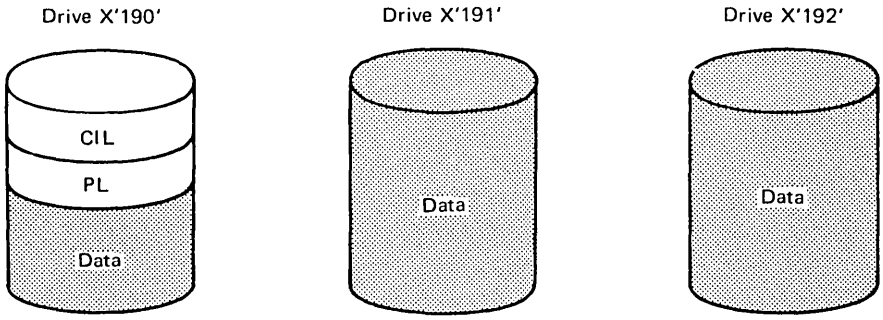
A private core image library can only be used to *supplement* the system core image library, which must always be present on SYSRES. Several private libraries may reside on the same disk as illustrated.

1 Compiling – Assembling – Link-Editing



The system core image library (CIL) contains only those programs required for execition-time processing. The compilers, assemblers, and the linkage editor are kept in the private core image library (PCIL).

2 Processing



For execution-time processing, the private libraries are no longer required and can be replaced by a data volume. Thus, maximum possible space is allowed for processing data.

CIL = system core image library
PL = procedure library
PCIL = private core image library
PRL = private relocatable library
PSSL = private source statement library

Figure 3.10. Example of Library Organization

With one disk drive you may prefer to maintain only enough free space in the relocatable library of the operational pack to contain the modules for the largest component in the system. This small relocatable library permits temporary insertion of any component in relocatable form. This component can then be immediately link-edited into the core image library and deleted from the relocatable library.

Similar considerations apply for the source statement library. Determine which of the IBM-supplied components you need on-line, which should be transferred to a backup volume for future extensions of your system, and which can be deleted entirely.

If you intend to use a procedure library, you should allocate sufficient space for it on the SYSRES file during system generation. In estimating the amount of space required, consider the number of job control statements and SYSIPT data records (source modules, utility control statements, etc.) you expect to store in the procedure library.

After you have determined the space requirements for your libraries in terms of number and size of programs, you must define and allocate the amount of disk space needed to accommodate these programs. A set of formulas is available to calculate the number of tracks and cylinders required for each library. These formulas are contained in *DOS/VS System Generation*. Refer to *Chapter 7: Using the Libraries* for information on how disk space is allocated to a library.

The contents of the libraries are identified in the *Memorandum to Users* (shipped with the distributed DOS/VS system). The storage requirements (sizes) for these components and macro definitions are identified in the section for each component.

Part II: Using the System

This section is provided especially for applications programmers and operators. It is a guide to the day-to-day use of the system. The chapters it contains are:

Chapter 4: Starting the System describes how the operator performs the initial program load (IPL) procedure. It also describes how to create the file required for recording error information.

Chapter 5: Controlling Jobs describes how the applications programmer or operator supplies input to the job control program, which controls the execution of a job.

Chapter 6: Linking Programs describes how the applications programmer prepares input to the linkage editor program, which links the modules produced by language translators and produces executable programs that are placed in the core image library.

Chapter 7: Using the Libraries provides applications programmers and operators with the information on how to alter, copy, and inspect the contents of the libraries. It also describes how to allocate space to the libraries and how to create private libraries.

Chapter 8: Using POWER/VS has been deleted. See *DOS/VS POWER/VS Installation Guide and Reference.*

Chapter 4: Starting the System

Before a job can be entered into the system for execution, the supervisor must be read into the supervisor area of real storage and the job control program must be loaded into the virtual background partition. To do this, the operator starts the system by following the initial program load (IPL) procedure.

This chapter describes the use of the IPL commands. The exact formats of these commands are contained in *DOS/VS System Control Statements*, and *DOS/VS Operating Procedures*. This chapter also provides a summary of the automatic functions of IPL; descriptions of how to modify the shared virtual area, and how to create the system record file (SYSREC) and the hard copy file; a section on the optional user exit routine for security checking after IPL; and a section on entering SYSREC if the reliability data extractor (RDE) option was generated in the supervisor.

You must perform the IPL procedure each time you have to:

- Load a new supervisor (for normal system start-up, for different supervisor options, or to recover from a system malfunction. For the last, refer to *DOS/VS Serviceability Aids and Debugging Procedures*).
- Change the channel and unit assignment of the system residence (SYSRES), the VSAM master catalog (SYSCAT), or the page data set (SYSVIS) due to hardware problems with the channel or disk drive.
- Modify the shared virtual area (to change allocation or to create the system directory list).
- Create SYSREC (for the first time or because the file was damaged).
- Replace SYSRES or SYSVIS because of a hardware problem with the pack.
- Add devices to or delete them from the system configuration.
- Set or change the time-of-day clock value.
- Set or change the system's time zone value (if TOD=YES was specified in the FOPT macro during supervisor generation).

Initial Program Loading (IPL)

To invoke the IPL routines, you place the system residence disk pack on a drive, set the address of that drive in the load unit switches, and press LOAD (on the video display/keyboard console, type in the address on the drive and press ENTER). This causes the first record on track 0 to be read into storage bytes 0-23. The information read in consists of an IPL PSW (program status word) and two CCWs (channel command words), which in turn cause the reading and loading of the IPL routines.

Next, IPL enters the wait state. You now must indicate to IPL the device that is to be used as the operator console. To do so, press the Request key (or END/ENTER) on the selected device. This causes an interrupt and automatically

transmits the address of this device to IPL. (If you have installed an IPL communication device list, the interrupt will only be accepted if the address of the device is contained in the list). IPL assigns the device to SYSLOG. The assigment remains valid until the next IPL; it overrides any SYSLOG assigments made during supervisor generation.

After SYSLOG assignment, IPL responds with an information type message requesting you to enter the supervisor name. If you wish to use the default supervisor ($$A$SUP1) simply press the request key or enter key; otherwise type in the name of the supervisor and press ENTER.

Operating in the supervisor state, IPL reads the supervisor nucleus into low real storage from the core image library. If an unrecoverable error is sensed while reading the supervisor nucleus, an error message is displayed on SYSLOG; the hard wait status is entered and an error code is set in the first four bytes of real storage. The IPL procedure must then be restarted. For more information on wait states and error codes, refer to the *DOS/VS Serviceability Aids and Debugging Procedures.*

After successfully reading in the supervisor nucleus, IPL assigns the current physical unit address of the system residence disk pack to the SYSRES file (in response to your dialing this address in the load unit switches).

Establishing the Communications Device for IPL

Next, the IPL routine places the central processing unit in the wait state with all interrupts enabled (see Note 1). At this time you must indicate which device is to be used to communicate the IPL commands to the system (see Note 2). The specific manual operation you must perform depends on the device desired:

- If you wish to use the console (SYSLOG), press the request key on the console. (On the video display/keyboard console, you can either press the enter key, the request key, or the cancel key.)
- If you wish to use a card reader that was not assigned as SYSRDR in the ASSGN macro during supervisor generation, ready this card reader. IPL then assigns the SYSRDR file to this device for the duration of this procedure.
- If you wish to use the card reader that is assigned as SYSRDR, press the interrupt key. (This card reader must have been readied before you pressed LOAD to invoke the IPL routines as described above.)
- If you wish to use the IBM 3540 Diskette I/O Unit, ready it. IPL assumes that the file IJIPL is part of the diskette and that it contains the IPL commands in card image format (unblocked 80 byte records).

Note 1: *Because any interrupt will (on a first-come basis) establish the issuing device as the IPL communication device, it is advisable that TP installations and terminal-oriented installations with locally attached terminals, (for example, IBM 3277) install the IPL-phase $$A$CDL0. (See IPL communication Device List.) A 3277 terminal which is to be used as an operator console or as an IPL communication device must not be attached to a selector channel.*

Note 2: *When you submit IPL commands, enter them via the selected communication device.*

Changing I/O Device Assignments

If the physical addresses of any I/O devices are different from those established by DVCGEN macros during supervisor generation, you have to change the system configuration. (To determine which devices are supported in the system configuration, check the supervisor assembly listing.) You can change the configuration by adding or deleting devices. IPL changes the physical unit configuration accordingly. The modified system configuration remains in effect until the next IPL.

If you want to change any symbolic unit assignments (except SYSRES, SYSCAT, and SYSVIS), you must use ASSGN statements or commands. These are processed by job control as described in the section *Symbolic I/O Assignment* in *Chapter 5: Controlling Jobs*.

Adding Devices

Use the ADD command to include an I/O device and physical unit address that were not included in the system configuration during supervisor generation. The following requirements should be kept in mind:

- You can add a device only if sufficient device table space was provided via the IOTAB macro during supervisor generation.
- If you add a tape cartridge unit, there must be enough space for an associated Tape Error Block (TEB) if TEBs were specified during supervisor generation.
- If DASD file protection was generated in the supervisor and you add a DASD, the DASD must conform to the channel range and DASD types specified in the DASDFP parameter.
- If the seek separation option was generated in the supervisor and you add a DASD, the system must be able to accommodate an additional seek address block (SAB).
- To add TP devices, TP support must have been specified during supervisor generation.

If any of these requirements is not satisfied, you will get an appropriate error message. You must then provide space in the control blocks for the additional device by:

- re-assembling the supervisor, or
- deleting unnecessary devices of the type you want to add. You must then re-issue the ADD command.

Deleting Devices

Use the DEL command to drop an I/O device from the existing system configuration. Because all references to the device are removed, any subsequent ASSGN job control statement that refers to a deleted device will not be accepted.

Setting System Values

The SET command is required because it indicates to IPL that the ADD and DEL commands (if any) are to be checked. The channel and unit assignment for SYSRES is also checked at this time.

You can use the SET command to set the system date in the communications region, the time-of-day clock, and the system time zone. If you specify a time-of-day clock setting, you must depress the time-of-day clock switch to the "enable set" position at the exact time specified in the SET command.

Assigning the VSAM Master Catalog

If VSAM is to be used, the CAT command may be used during IPL to assign the VSAM master catalog to the SYSCAT file. This is only necessary if you wish to override the SYSCAT assignment made during system generation, or if you failed to assign SYSCAT during system generation. The CAT command (if used) must be submitted after the SET command and before the DPD command (described below). In the CAT command, you indicate the channel and unit number to be associated with the SYSCAT file. If the VSAM master catalog resides on a 3340 disk storage device, this device must be ready before starting the IPL procedure.

Initiating Page Data Set Handling

You must follow the SET command (or the CAT command) by the DPD command to indicate that IPL is to handle the page data set, which is necessary for the virtual address area. The DPD command is required, with or without operands. If submitted without operands, IPL will use the information specified in the DPD macro during supervisor generation to perform page data set handling. This includes opening the page data set, checking its extent limits, and creating label information in the volume table of contents (VTOC). IPL assigns the symbolic name SYSVIS to the page data set.

The operands of the DPD command indicate whether the page data set is to be formatted, its location, extent, and (optional) volume identification. Because formatting the page data set is time-consuming, you should only request it if the pack was damaged. The first time you use the page data set, it will be formatted automatically.

The page data set can reside on any DASD supported by DOS/VS as a system residence device. To help ensure better performance, the page data set should not reside on a pack that is subject to heavy I/O requests.

Automatic Functions of IPL

IPL performs the following operations automatically:

- Sets storage protection keys to coincide with the partition allocations determined during supervisor generation.

- Checks that the CPU model specified during supervisor generation is the same as the model being used.
- Informs the operator about the status of the time-of-day clock.
- Checks that all DASDs included in the configuration conform to the channel range and DASD types specified in the DASDFP parameter (if specified during supervisor generation).
- Checks that 3340 disk storage devices that are on line contain data modules of a size as described by the pertinent PUB and, if they do not, updates the PUB accordingly.
- Unassigns any DASD assignments for devices that are not operational at this time (so as to prevent the error recovery routines from trying to establish error recording statistics for these devices).
- Fetches the buffer loader transients to load the printer-control buffers of pertinent printers.
- Builds an address list in the supervisor for all RAS transients cataloged in the system core image library. (The first RAS transient is also loaded during IPL.)

After IPL completes these operations, the system loader loads the job control program into the virtual background partition and places the system in the problem program state. The message "READY FOR COMMUNICATIONS" appears on the console immediately after IPL is complete unless a warm start copy of the SVA is found (in which case the message appears directly thereafter).

IPL Communication Device List

For teleprocessing installations and for installations with locally attached terminals (such as the IBM 3277), devices allowed to present an interrupt to IPL should be restricted; in this way, a device outside the operator's control cannot establish itself as the device for submitting IPL commands.

To build a restrictive pool of allowable IPL communication devices you can create an IPL communication device list (CDL) and catalog the list under the phasename $$A$CDL0 in the system core image library (SCIL). IPL automatically checks for the presence of this phase and, if it is present, reads the CDL into real storage. (Installation of the phase is optional.) When IPL enters the wait state and an interrupt occurs, the CDL is searched for the address of the device issuing the interrupt. If the address is listed, the interrupting device is accepted as an IPL communication device and processing continues. If the address is not found, IPL again enters the wait state.

Once phase $$A$CDL0 has been cataloged, the CDL addresses remain effective for subsequent IPLs. However, you can delete or make changes in the list as follows:

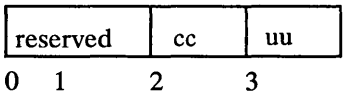
- Delete or rename the phase, using the MAINT program.
- Override any CDL entry by manual intervention.

You cannot add or delete addresses in the CDL list singly; instead, you must reassemble and catalog any updates you can find necessary. You can, however, create a temporary CDL by using the ALTER MEMORY

function – the temporary CDL is effective for only a single run (see the following section on RESTART/ALTER MEMORY facilities).

For IPL to be successful, once $$A$CDL0 is installed, the SYSLOG device address must be present in the CDL (it usually ranges from 009 to 01F). If you intend to submit IPL commands (such as ADD, DEL, etc.) from card reader or diskette, you must enter these addresses in the CDL as well. To ensure backup in case of hardware errors during IPL, consider stand-by devices, such as another card reader, diskette, or even an additional SYSLOG device; list the addresses of these alternatives in the CDL.

The CDL may have up to eight entries each of which is four bytes in length:



where: cc = channel number
uu = unit number

You create the CDL by submitting a job that catalogs $$A$CDL0 into the system core image library. The following example creates a CDL with five entries:

```
// JOB CATALOG CDL
// OPTION CATAL,NODECK
   PHASE $$A$CDL0,+0
// EXEC ASSEMBLY
$$A$CDL0 CSECT
          DC  XL4'00C'      card reader
          DC  XL4'009'      1052
          DC  XL4'01F'      SYSLOG (DOC)
          DC  XL4'4BD'      3277
          DC  XL4'240'      diskette
          END
/*
// EXEC LNKEDT
/&
```

RESTART/ALTER MEMORY Facilities

If your CDL has been created incorrectly (for example, the SYSLOG device address was omitted, or the CDL contains invalid code), IPL will enter an endless loop because the address of the device used by the operator cannot be found in the CDL. You can recover from this situation by manual intervention using the RESTART/ALTER MEMORY facilities.

- Activate a RESTART interrupt when IPL is waiting for an interrupt from the SYSLOG device. IPL then enters the disabled wait state.
- Use the ALTER MEMORY function and
  - key in the device address of SYSLOG into real storage locations X'10' to X'13' (for example, 0000001F).

- If your IPL communication device is a card reader or diskette, key in the address of the device into real storage locations X'14' to X'17' (for example, 000004BD).

- Activate RESTART again. This signals IPL to take a new CDL from storage locations X'10' to X'17'. IPL enters the enabled wait state.

- Now press the Request key (or END/ENTER) on the SYSLOG device to continue normal IPL processing.

Note: *The above RESTART/ALTER MEMORY facilities can also be used to enter a CDL manually at IPL wait time. It allows you to override an existing CDL entry or to provide CDL functions in the case that a CDL was not created. In both instances, the address of the SYSLOG device must be given. All assignments are of temporary nature, they do not apply to subsequent IPLs.*

Building the SDL and Loading the SVA

After IPL when job control is first invoked, it will attempt to find a warm start copy of the shared virtual area (SVA). If a warm start copy is found, you can either accept it or reject it. You should reject it if you want to reallocate the SVA, load other phases into the SVA and system directory list (SDL), or add phase names to the (SDL).

If the warm start copy is rejected or not available, you can change (if desired) the allocation of the SVA specified during supervisor generation by means of the SET SVA job control command.

Next, you must submit SET SDL=CREATE, which enables job control to build the system directory list and to load the SVA. (**Note:** *The procedure library initially contains suggested statements for loading the system directory list.)* Immediately following these statements, enter the phase names to be included in the system directory list via SYSRDR or SYSLOG (depending on the device from which job control is reading). These statements can be entered via the IPL communications device. Figure 4.1 illustrates such a job stream.

These statements can also be entered via a cataloged procedure. The procedure library, as distributed with the system, contains two procedures for loading the SVA, for which refer to *DOS/VS System Generation.* You can also create your own procedure to load your own phases into the SVA. Execute this procedure immediately after IPL.

The phases need not be currently cataloged in the core image library, and, if they are not, the system issues a message on SYSLST (or SYSLOG if SYSLST is not available). If you subsequently catalog a phase into the system core image library under a name listed as uncataloged, the entry in the SDL is activated. In this case, if the phase is also identified in the SDL as eligible for the SVA, it is loaded there immediately after it has been link-edited. Thus, under the circumstances described above, you do not have to re-IPL when you want to load additional phases in the SVA.

Note: *The SDL is not searched for a phase that is loaded by an attention routine.*

Replacing Phases Stored in the SVA

Occasionally, a phase stored in the SVA needs to be changed; that is, it must be replaced by an updated version. To replace a phase in the SVA, linkedit the updated version of the phase to the system core-image library. Immediately after this linkedit operation, DOS/VS loads the updated phase into the SVA. The old version of the phase remains in the SVA, but is made inaddressable. Linkediting for inclusion of a phase in the SVA is further discussed in *Chapter 6: Linking Programs*.

Creating the System Recorder File

The DOS/VS Recovery Management Support Recorder (RMSR) requires a disk extent on which to record statistical information about machine errors and environmental information. This disk extent is called the system recorder file and is identified by the symbolic name SYSREC. The SYSREC file must be created before job control encounters the first JOB card following an IPL procedure. Usually, you create the SYSREC file only after the first IPL (not after each IPL). If the SYSREC file has been damaged, however, you must re-IPL and re-create SYSREC.

The SYSREC file requires a minimum of ten tracks (not including an alternate track) and cannot be a split cylinder file. You must define SYSREC as an extent of a permanently online disk device that DOS/VS supports as a system residence device.

The SYSREC file label information must be included in the standard label portion of the label cylinder on the SYSRES file. You must, therefore, submit the // OPTION STDLABEL statement when creating the SYSREC file. (Since the label information you submit is written at the beginning of the standard label track, which overwrites the information that was present there, you must resubmit all the necessary information. A more detailed description of preparing standard label information is contained in *Chapter 5: Controlling Jobs.*)

Figure 4.1 illustrates a job stream to create the system recorder file. The IPL commands are included in the figure to emphasize the proper placement of the statements that create the SYSREC file. Do not include a // JOB statement until you have supplied all the information applicable to SYSREC. This is because the SYSREC file is opened when the first // JOB statement is encountered. Note that the file name IJSYSRC is required in the DLBL job control statement.

When the system is to be shut down, you should issue the Record On Demand (ROD) command to ensure that no statistical data is lost. For the IBM System/370 Models 115 and 125, the U command of the mode select display, should also be issued to save disk usage statistics on the service DISKETTE. These commands are not valid for recording teleprocessing statistical data. Refer to the appropriate teleprocessing guides for more information.

To obtain a listing of the SYSREC file, run the EREP program as described in *DOS/VS Serviceability Aids and Debugging Procedures.*

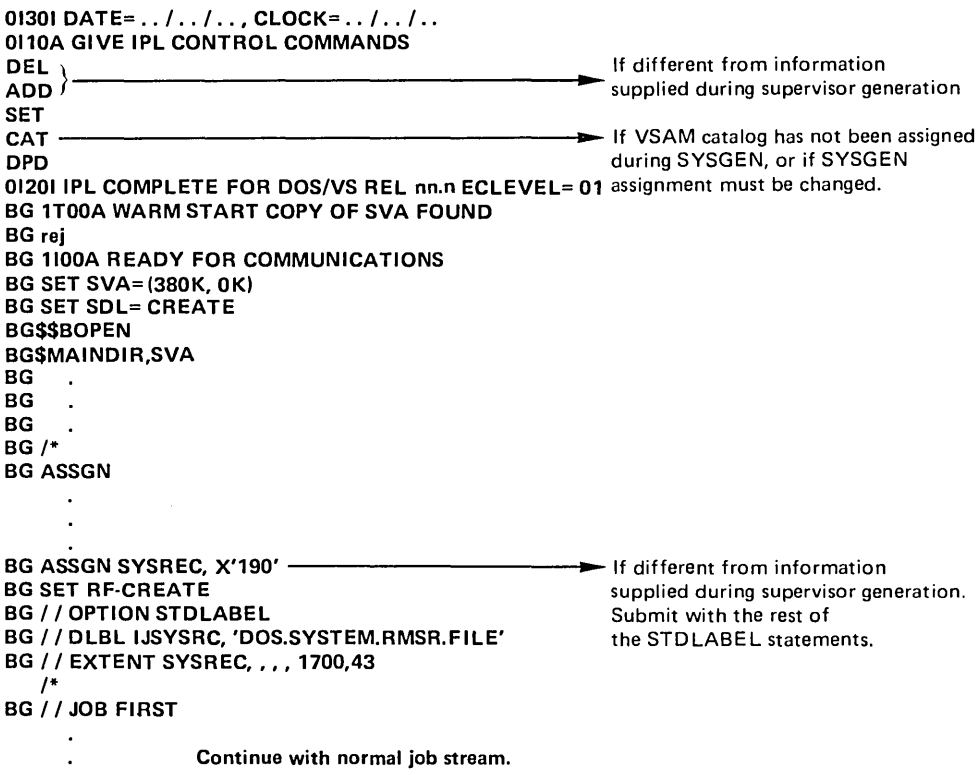


Figure 4.1. Example of Creation of the Shared Virtual Area and the SYSREC File

During execution of the EREP program, recording on SYSREC is suppressed.

Creating the Hard Copy File

On a system that supports a video display/keyboard console, all messages displayed on the screen and all information typed in by the operator are saved in a file on the device assigned to SYSREC. This file is called the hard copy file because you can obtain printed copies of the file whenever required.

You must create the hard copy file after the first IPL procedure and before you submit the first // JOB statement to the job control program.

The control statements and commands needed to create the hard copy file are the same as those shown in Figure 4.1 for the SYSREC file with the exception that you specify HC=CREATE in the SET command, and the filename IJSYSCN in the DLBL job control statement. More information about creating and printing the hard copy file is given in *DOS/VS Operating Procedures*, and *DOS/VS System Utilities*.

Security Checking after IPL

In the larger DOS/VS systems it is often desirable to perform certain security checks at the end of an IPL procedure. It may, for instance, be

important to know who performed the procedure, whether the right system pack was mounted, and whether the correct date was entered for the new work session. Moreover, if you work with labeled data files it is important that they bear the correct creation date, so as to guarantee that data files are protected until their expiration date.

After the IPL procedure has been completed, control can be passed to a user exit routine (phase name=$SYSOPEN) that checks system security and integrity. This routine is entered once after every IPL procedure. The DOS/VS distribution volume contains a dummy phase $SYSOPEN in the system core image library. If you do not use the facility it has no effect on your system. Conventions for writing this kind of user exit routine, together with an example, are contained in the section *Writing an IPL User Exit Routine* in *Chapter 10: Using the Facilities and Options of the Supervisor.*

Entering RDE Data

If the supervisor was generated to support the reliability data extractor (RDE), the system will ask you to provide additional information about the system when the first // JOB statement after IPL is processed. A message (1I89A IPL REASON CODE=) is issued on the device assigned to SYSLOG. You should respond with a reason code (two characters), which indicates why the system was restarted. The system may have been started as the beginning of normal operation or restarted because of a machine error, a program error, an operator error, etc. Another message (1I91A SUB-SYSTEM ID=) is issued and you should respond with a code identifying the device type or program type that failed. On the basis of these replies job control will build a record for SYSREC.

Before shutting down at the end of the day (or processing period), you must ensure that no environmental data is lost, by issuing the ROD command. This command also causes the RDE end-of-day record to be written on the disk assigned to SYSREC. To obtain a listing of this file, run the EREP program as described in *DOS/VS Serviceability Aids and Debugging Procedures.*

This information will be very valuable to your operations management. By replying with the exact reason code that applies in each case, you are in fact ensuring a permanent record of the reason why you had to re-IPL.

Refer to the *DOS/VS Operating Procedures*, for more extensive information on the RDE messages and the valid replies to them. *DOS/VS Messages* also contains this information for use at the console.

Chapter 5: Controlling Jobs

After the system has been successfully started by means of the IPL program it is ready to accept input for execution.

The unit of work that is submitted to the system for execution is called a *job*. A job, and the environment in which it is to run, must be defined to the system through job control statements and commands. These job control statements and commands are processed by the job control program. The job control program is invoked by the supervisor

- after initial program loading, to process the first job after an IPL procedure, or
- at the normal or abnormal end of a job or job step.

The job control program runs in any virtual partition of at least 64K bytes. It performs its functions only between jobs and job steps, and, therefore, it is not present in the partition while a problem program is being executed.

This chapter describes how to supply information to the job control program to enable it to prepare a job for execution. It shows how to define jobs and job steps, how to associate files on auxiliary storage with problem programs and how to execute programs in virtual or real mode. Moreover, it describes how standard sets of job control statements, called cataloged procedures, can be retrieved from the procedure library, and how cataloged statements can be modified.

After each job control statement is read, control can be given to a user exit routine for examining and altering job control statements prior to their being processed by the system. For a comprehensive description of this facility refer to the section *Checking and Altering Job Control Statements* later in this chapter.

The differences between job control statements and commands are not spelled out in detail because a clear-cut distinction is not required in the context of this chapter. Whenever applicable, it is simply stated whether the function can be performed using statements, commands, or both. The description of the job control statements and commands in this chapter is limited to their use and functions; formats and characteristics of statements and commands are detailed in *DOS/VS System Control Statements*.

The information in this chapter is intended for use by system programmers, application programmers, and system operators.

Defining a Job

The beginning and end of a job are defined by the JOB and / & (end-of-job) statements:

```
// JOB jobname
   .
   .
 additional job control statements and program input
   .
   .
/&
```

The program to be executed in a job is invoked through the EXEC statement. In the following example, the program PROGA is fetched from the core image library and executed:

```
// JOB jobname
   .
   .
// EXEC PROGA
   .
   .
/&
```

One or more programs can be executed within a job; the execution of a single program is a *job step*. Therefore, each job can consist of one or more job steps. The following job comprises two job steps.

```
// JOB jobname
   .
   .
// EXEC PROGA
   .
   .
// EXEC PROGB
   .
   .
/&
```

You are free to include as many job steps in a job as you wish. It is, however, not advisable to execute, in one job, several programs that are completely independent of one another. This is because, if one step terminates abnormally, the job control program will ignore the remaining job steps up to the next / & statement.

Thus, although perfectly in order, the programs following the one that failed will not be executed. A typical example of related job steps that should form a single job are assembling, link-editing, and executing a program, where correct execution of one job step depends on successful completion of the preceding one.

For POWER/VS job setup considerations and examples refer to *DOS/VS POWER/VS Installation Guide and Reference.*

Setting Up Job Streams

The job control program provides automatic job-to-job transition. This means that an unlimited number of jobs can be submitted to the system in one batch, and that job control processes one job after the other without

requiring intervention by the operator. The job or jobs submitted are referred to as a *job stream* (see Figure 5.1 for an example of a payroll jobstream).

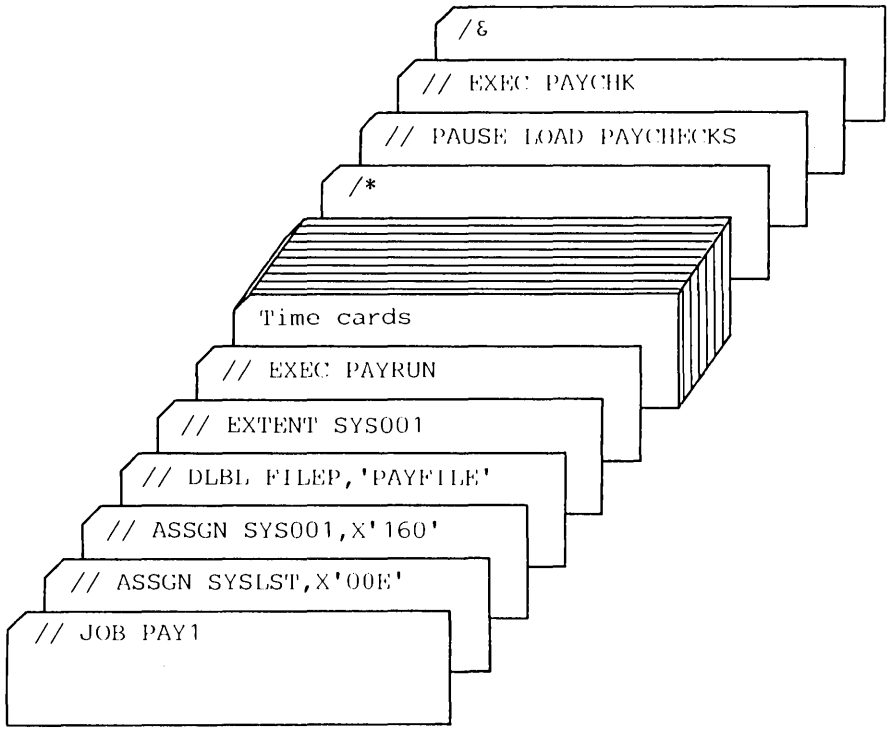


Figure 5.1. Example of a Job Stream

The operator can interrupt the processing of a job stream in any partition to make last-minute changes to one of the jobs or to squeeze in a special rush job. He does this by pressing the request key on the operator console and entering a PAUSE job control command. This causes processing to halt at the end of the current job step, or, if the EOJ operand is specified in the PAUSE command, at the end of the current job.

When setting up a job stream for a partition, you should bear in mind that all jobs will get the priority of that partition. The selection of the jobs for a particular partition in a multiprogramming system can help to improve the efficiency of your installation. For example, jobs which have a relatively low CPU usage and a relatively high rate of I/O activity, and which therefore spend most of their time waiting for the completion of I/O operations, should run in a high priority partition. Conversely, CPU-bound jobs should be in a partition with a lower priority. More information about partition priorities is given in the section *Multiprogramming* in *Chapter 1: Understanding the System*.

Summary of Job Control Statements and Commands

The following describes the JOB, end-of-job (/ &), DATE, and PAUSE statements/commands. The EXEC statement is discussed under *Executing a Program*, later in this chapter. The description of the statements will

touch upon a number of subjects (for example, job control options, logical unit assignments, UPSI byte, label information cylinder, etc.), which will be discussed later in this chapter.

JOB

The JOB statement indicates the beginning of control information for a job. The specified job name is stored in the communications region of the corresponding partition and is used by job accounting and to identify listings produced during execution of the job.

The JOB statement may be omitted, in which case the job name NONAME is stored in the communications region. If the JOB statement is present, it must contain a job name; otherwise, an error condition occurs.

The JOB statement is always printed in positions 1 through 72 on SYSLST and SYSLOG. If the time-of-day clock is supported, the time of day is also printed. The JOB statement causes a skip to a new page before printing is started on SYSLST.

When a JOB statement is encountered, the job control program stores the job name from the JOB statement into the communications region. If the / & statement was omitted, the JOB statement will cause control to be transferred to the end-of-job routine to simulate the / & statement. Refer to the following section for the operations that are performed.

End-of-Job (/ &)

This statement is the last one for each job (not job step). It signals the end of the input stream for the job. When job control encounters / & on SYSRDR during normal operation, the standard assignment for SYSIPT becomes effective and SYSIPT is checked for an end-of-file condition.

If the standard assignments for SYSRDR and SYSIPT are not to the same device, SYSIPT is advanced to the next / & statement. In the event of an abnormal termination, job control advances SYSRDR and SYSIPT to the next / & and proceeds, only if a JOB statement is provided. Therefore if SYSRDR and SYSIPT are assigned to different devices, the / & statement should be present on both devices.

If the / & statement is omitted, the next JOB statement will cause control to be transferred to the end-of-job routine to simulate the / & statement.

When a / & statement is encountered, the job control program performs such operations as the following:

- Resets all job control options for the partition to standard, as established at system generation, resets the LINK and CATAL options to zero.
- Resets all system and programmer logical unit assignments for the partition to the permanent assignment established by job control commands, or (if no permanent assignments have been made) to the standard assignment established during supervisor generation.
- Modifies the communications region as follows:
  1. Resets the date from the DATE statement to the one specified in the SET command during IPL, or (if the time-of-day clock is supported) to the date currently valid.
  2. Stores the job name NONAME.
  3. Sets the user area and the UPSI byte to zero.

- Displays the EOJ message on SYSLST and SYSLOG with the time and duration of the job if the time-of-day clock is supported.
- Lists all tape error statistics (TEBs) for the IBM 2495 tape cartridge reader.
- Ensures that end-of-file has been reached on SYSIPT.
- Deletes the temporary labels in the label information cylinder on SYSRES and restores the USRLABEL mode. (See *Editing and Storing Label Information*, later in this chapter.)
- Checks whether the automatic condense limits of any of the libraries have been reached (if maintenance has been done in the job).

PAUSE

The PAUSE statement or command can be used to allow for operator intervention between jobs or job steps.

The PAUSE statement can be included anywhere among the job control statements of a job stream. It becomes effective at the point where it was inserted; processing is suspended in the affected partition, and the operator console is unlocked for input. The PAUSE statement can contain instructions to the operator and is always listed on SYSLOG.

The PAUSE statement may also be helpful when SYSIN is assigned to a 5425 card reader (which does not have an end-of-file button). Place the // PAUSE card after the last / & card; this will force control to be given to the console-keyboard, which enables the console operator to control subsequent system operation.

The PAUSE command may be entered either through the operator console (after pressing the request key), or as a job control card; if entered through the console to the attention routine, the command must specify the partition that is to pause (if the background partition is intended, however, no operand is required). After encountering a PAUSE command, the system passes control to the operator (through the console) the next time that the job control program is fetched into the specified partition, that is, at the end of the current job step (which may also be the end of the job). If the PAUSE command that is entered through the console specifies the EOJ operand, however, control will pass to the operator only at the end of the current job, regardless of the number of steps needed to reach that point.

DATE

The DATE statement can be used to override the date specified in the SET command during IPL. The new date is stored in the communications region for the duration of one job only, unless it is overridden by another DATE statement.

Note: *The date is not incremented if the job runs past midnight.*

You can use the DATE statement, for example, when your program's output is to indicate yesterday's date. The DATE statement can be submitted with the rest of the job control statements.

Relating Files to your Program

Programs always perform some kind of input/output operation, that is they process files on auxiliary storage devices. Before such files can be processed, certain information about the files must be provided to the system. This information includes:

- The generic device name and volume serial number or the physical address of the I/O device on which each of the files resides. (Relating a file to an actual I/O device is called symbolic I/O assignment).
- For files on direct access storage devices (DASD), the exact location of the file on the storage medium.
- For files on DASD, on diskettes, or on labeled magnetic tape, a description of the file, called a label, which is used for checking and protection purposes.

The above information, specified in job control statements, is stored in the system by the job control program for use by the DOS/VS data management routines. How this is done is described below.

Symbolic I/O Assignment

Whenever a processing program needs access to a file on auxiliary storage, the system must be informed of the address of the I/O device involved. The program need not specify an actual device address, but only a symbolic name which refers to a logical, rather than physical, unit. Before the program is executed the logical unit must be associated with an actual device. This is done by either the system, the programmer, or the operator, by means of the ASSGN job control statement or command which specifies the symbolic name of the logical unit and one of the following:

- A general device class or specific device type, with or without volume serial number.
- The physical address (channel and unit number) of the I/O device.
- A list of physical addresses.
- Another logical unit.

See Figure 5.2 for an illustration of some of these combinations.

Logical Units and Symbolic Device Names

There are two types of logical units: *system logical units,* primarily used by the system control and service programs, and *programmer logical units,* primarily used by the processing programs. The following list shows the symbolic names that refer to a logical unit and the I/O devices that each unit can represent. In the case of disk devices, the logical unit is not assigned to the entire volume mounted on the device but only to the referenced extent(s). Refer to the section *Files on Direct Access Devices* for more information on disk files.

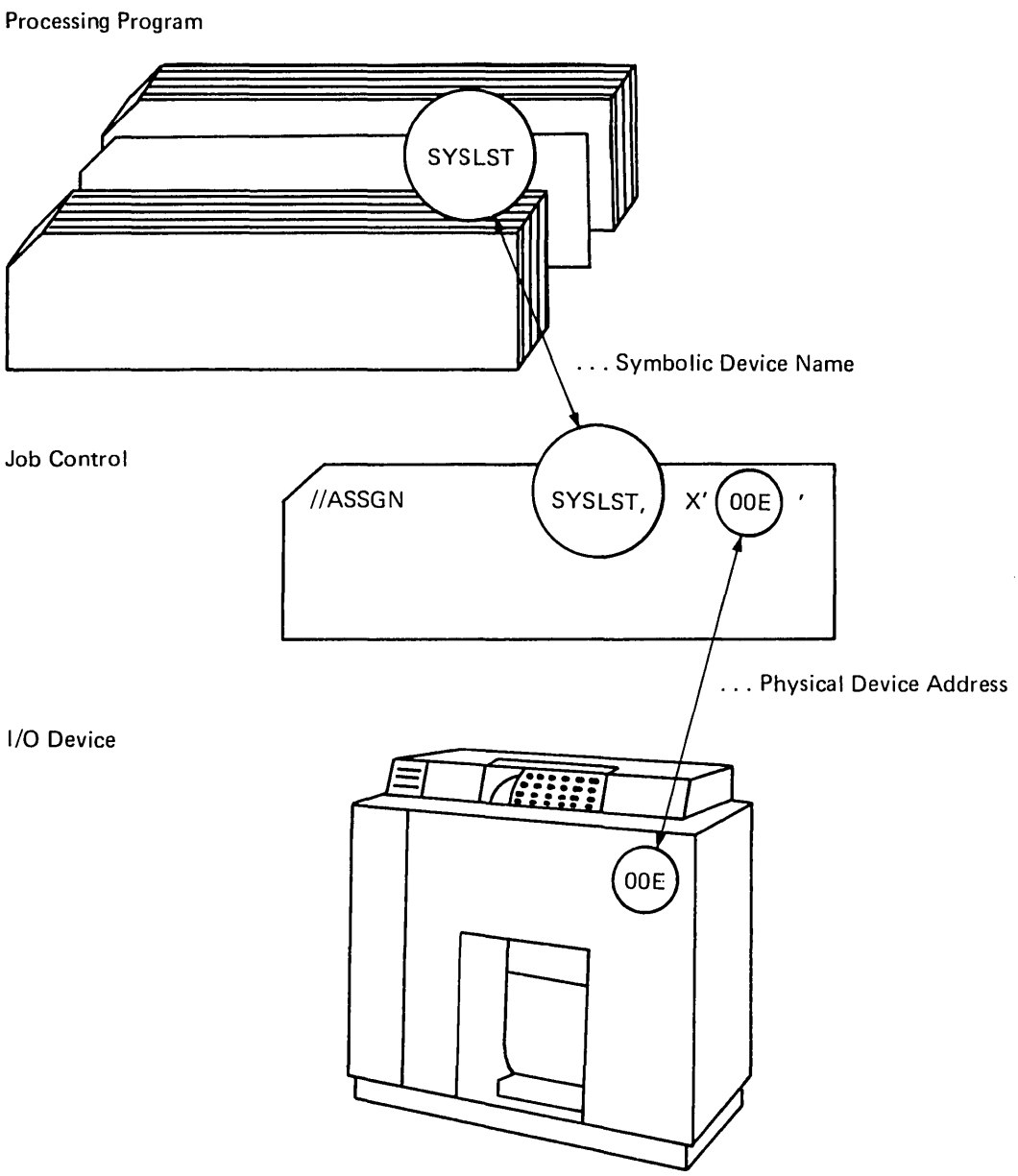


Figure 5.2. Example of Symbolic I/O Assignment (Part 1 of 2)

1. The logical unit specified in the processing program (via DTF or CCB) is a print file referred to by the symbolic device name SYSLST.
2. An ASSGN statement is used to associate SYSLST with the physical address 00E of a printer. This information is stored in the system by job control and can be accessed when a program is executed.

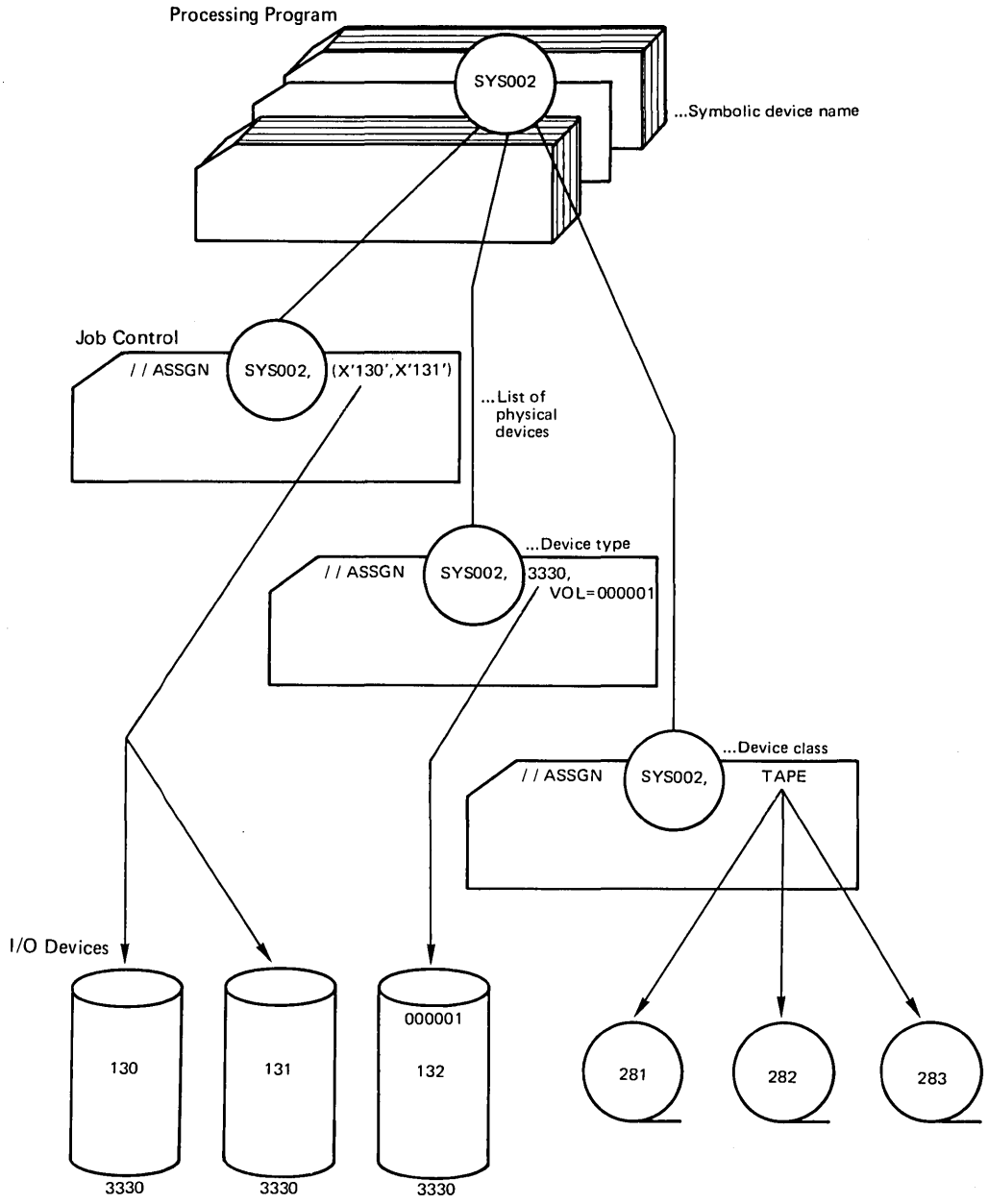


Figure 5.2. Example of Symbolic I/O Assignment (Part 2 of 2)

If you use the DISK device class option, or device type option use volume serial numbers, and be sure that they are unique.

SYSRDR	Card reader, magnetic tape unit, disk device, or diskette used as input unit for job control statements or commands.
SYSIPT	Card reader, magnetic tape unit (single volume), disk device, or diskette used as input unit for programs.
SYSPCH	Card punch, magnetic tape unit, disk device, or diskette used as the unit for punched output.
SYSLST	Printer, magnetic tape unit, disk device, or diskette used as the unit for printed output.
SYSLOG	Operator console used for communication between the system and the operator and for logging job control statements.
SYSLNK	Disk device used as input to the linkage editor.
SYSRES	System residence extent on a disk pack.
SYSCLB	Disk device used for a private core image library.
SYSSLB	Disk device used for a private source statement library.
SYSRLB	Disk device used for a private relocatable library.
SYSREC	Disk device used to store error records collected by the recovery management support recorder (RMSR) function. For the Models 115 and 125, messages to or from the operator are stored on another file on SYSREC so that a hard copy listing of these messages can be produced.
SYSVIS	Disk device used to hold the virtual storage page data set.
SYSCAT	Disk device used to hold the VSAM master catalog.
SYSCTL	Used by DOS/VS at IPL time to load the buffer(s) of FCB-type printers.

Of these system logical units, user programs may also use SYSIPT and SYSRDR for input, SYSLST and SYSPCH for output, and SYSLOG for communication with the operator. However, other system logical units must not be used in place of programmer logical units (within user programs or EXTENT statements).

Two additional symbolic names, SYSIN and SYSOUT, are used under certain conditions:

SYSIN	*Can* be used if you want to assign SYSRDR and SYSIPT to the same card reader or magnetic tape unit. You cannot assign SYSRDR and SYSIPT to the same disk or diskette extent, you must instead assign SYSIN to that extent.
SYSOUT	*Must* be used if you want to assign SYSPCH and SYSLST to the same magnetic tape unit. It *cannot* be used to assign SYSPCH and SYSLST to disk or diskette because these two units must refer to separate extents.

SYSIN and SYSOUT are valid only to job control and cannot be referenced in a user program. Examples for the use of SYSIN and SYSOUT are given in the section *System Files on Tape, Disk, or Diskette* later in this chapter.

SYS000 - SYSmax: Any devices in the system used for processing program (including user program) input/output.

Note: *The linkage editor uses SYS001 and the assembler uses SYS001, SYS002, and SYS003. Some IBM language translators also use SYS004 and DOS/VS system utilities use SYS005 (refer to the appropriate programmer's guides).*

You can assign each of these programmer logical units to any of the existing partitions without a prescribed sequence. The maximum number of programmer logical units for the system and for each partition as well as the minimum per partition can be determined as follows:

- The background partition requires a minimum of ten programmer logical units.
- Each foreground partition requires a minimum of five programmer logical units.
- The maximum number of programmer logical units in the system depends on the partitions generated. The maximum value that you can specify as SYSmax is as follows:

NPARTS	Maximum number of programmer logical units for	
	F1	Total of all other partitions (including BG)
1	-	241
2	241	226
3	241	212
4	241	198
5	241	184

The maximum number of programmer logical units for the foreground partition F1, independent of the number of other partitions, is always 241.

- The maximum number of programmer logical units for a specific partition is determined by the formula:

  max. number- sum of all programmer logical units assigned to all other partitions except F1.

  As an example, assume that your system has five partitions. The maximum number of programmer logical units for a five partition system is 184. Assume further that 15 programmer logical units have been assigned to the partition F1, 13 to F2, 19 to F3, and 11 to F4. The maximum number of programmer logical units for the background partition would then be

  $$184 - (13 + 19 + 11) = 141$$

  (The 15 programmer logical units (LUBs) for F1 are not included).

When you specify a programmer logical unit in the form SYSnnn, the range for nnn is 000 up to but not including the maximum number of programmer logical units.

Types of Device Assignments

Device assignments are either standard, permanent, or temporary, depending on the time of the assignment and the type of ASSGN statement or command used.

Standard Device Assignments. Standard device assignments are established during supervisor generation in the ASSGN macro. These assignments are valid until the next supervisor generation.

Once the supervisor is loaded, and after IPL, modifications to the existing standard assignments can be introduced. These assignments can be either permanent or temporary.

Permanent Device Assignments. A permanent assignment is set up between jobs or job steps any time after IPL by the ASSGN job control command (no //) or the ASSGN job control statement with the PERM operand. It is valid until the next IPL procedure unless superseded by another ASSGN job control command. A permanent assignment can be changed for the duration of a job or job step by a // ASSGN statement or by an ASSGN command with the TEMP option.

Temporary Device Assignments. A temporary assignment is established either by a // ASSGN statement or by an ASSGN command with the TEMP option. It is valid for a single job only, unless superseded by another temporary or permanent assignment. Temporary assignments are reset to standard or permanent by

- a / & or JOB statement, whichever occurs first, or by
- a RESET job control statement or command.

Restrictions: The type of device assignment is restricted under certain conditions:

1. If one of the system logical units SYSRDR, SYSIPT, SYSLST, or SYSPCH is assigned to a disk device or diskette the assignment must be permanent or standard.
2. If SYSRDR and SYSIPT are to be assigned to the same disk device or diskette SYSIN must instead be assigned and this assignment must be permanent.
3. SYSOUT, if used, must always be a permanent assignment.
4. SYSIN and SYSOUT cannot be specified in the ASSGN macro during supervisor generation, that is, they cannot be standard assignments.
5. The SYSLOG assignment is restricted when SYSLOG was previously assigned during IPL or by an ASSGN statement/command. The following table shows the restrictions to the SYSLOG assignments:

new ASSGN / old ASSGN	PRINTER	1052	125D	3277
PRINTER	YES	YES	If IPL from 125D YES	If IPL from 3277 YES
			Else NO	Else NO
1052	YES	YES	If IPL from 125D YES	If IPL from 3277 YES
			Else NO	Else NO
125D	YES	YES	YES	NO
3277	YES	YES	NO	YES
YES = New ASSGN is allowed NO = New ASSGN is not allowed				

Device Assignments in a Multiprogramming System

During supervisor generation you can establish the standard assignments for the system and programmer logical units for each partition. The same logical unit can be defined for all partitions referring either to the same or to different physical devices. Also, different logical units can refer to the same physical device. This is illustrated in Figure 5.3.

At any other time, however, it is not possible to share a physical device (except DASD) between partitions. If the physical device in cases (2) and (3) in Figure 5.3 is not DASD and, for example, no program is in the F2 partition when you want to initiate the F1 partition, you must first unassign this physical device in the background partition.

With direct access devices this problem does not exist because each extent of a disk or data cell can be thought of as a separate device. It is not possible, however, to share a diskette between partitions.

When assigning a DASD, it is advantageous to specify a volume serial number in the EXTENT statement, especially for a scratch pack.

Device Assignments Required for an Assembly

Figure 5.4 shows the logical units that must be assigned to assemble a program. Note that the ASSGN statements must always precede the EXEC statement of the job step for which they are to be effective.

The device assignments for compilers are similar to the device assignments shown in this assembler example; any variations are documented in the applicable programmer's guides.

Files on Diskette Devices

After you have informed the system, via the ASSGN statement or command, on which physical device the file is to reside, you must supply the following information to allow the creation and checking of diskette labels:

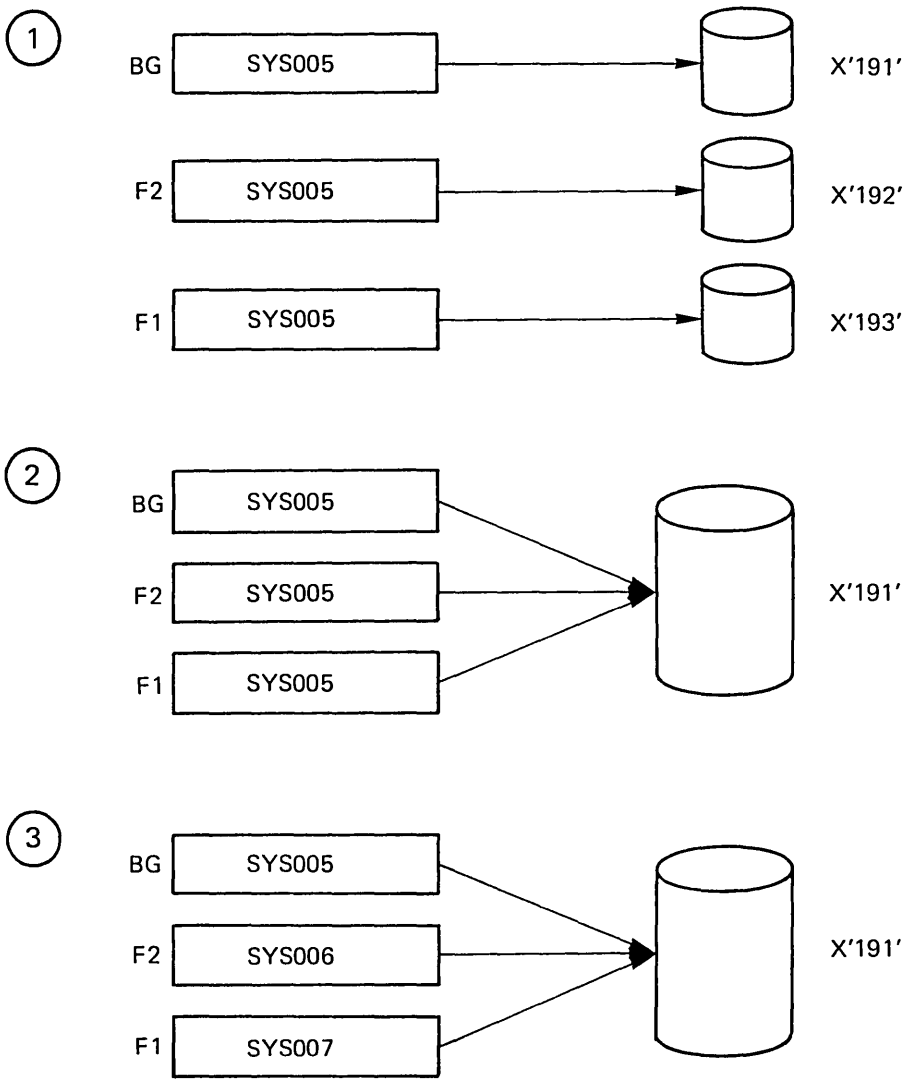


Figure 5.3. Possible Device Assignments Set at Supervisor Generation

1. A description of the characteristics of the file. You specify this in the DLBL job control statement.
2. The volume(s) the file is contained on. You specify this in one or more EXTENT job control statements.

The label information you supply in the DLBL job control statement may include the following:

- The name of the file. This name must be identical to the corresponding file name specified in your program. For programs written in assembler language, this would be the name of the DTF (**D**efine **T**he **F**ile).
- An identification of the file. This name is the one contained in the volume table of contents (VTOC) on the diskette. It is associated with the file name via a DLBL statement for the duration of a specific job or job step to make programs independent of physical files.
- The expiration date of the file.

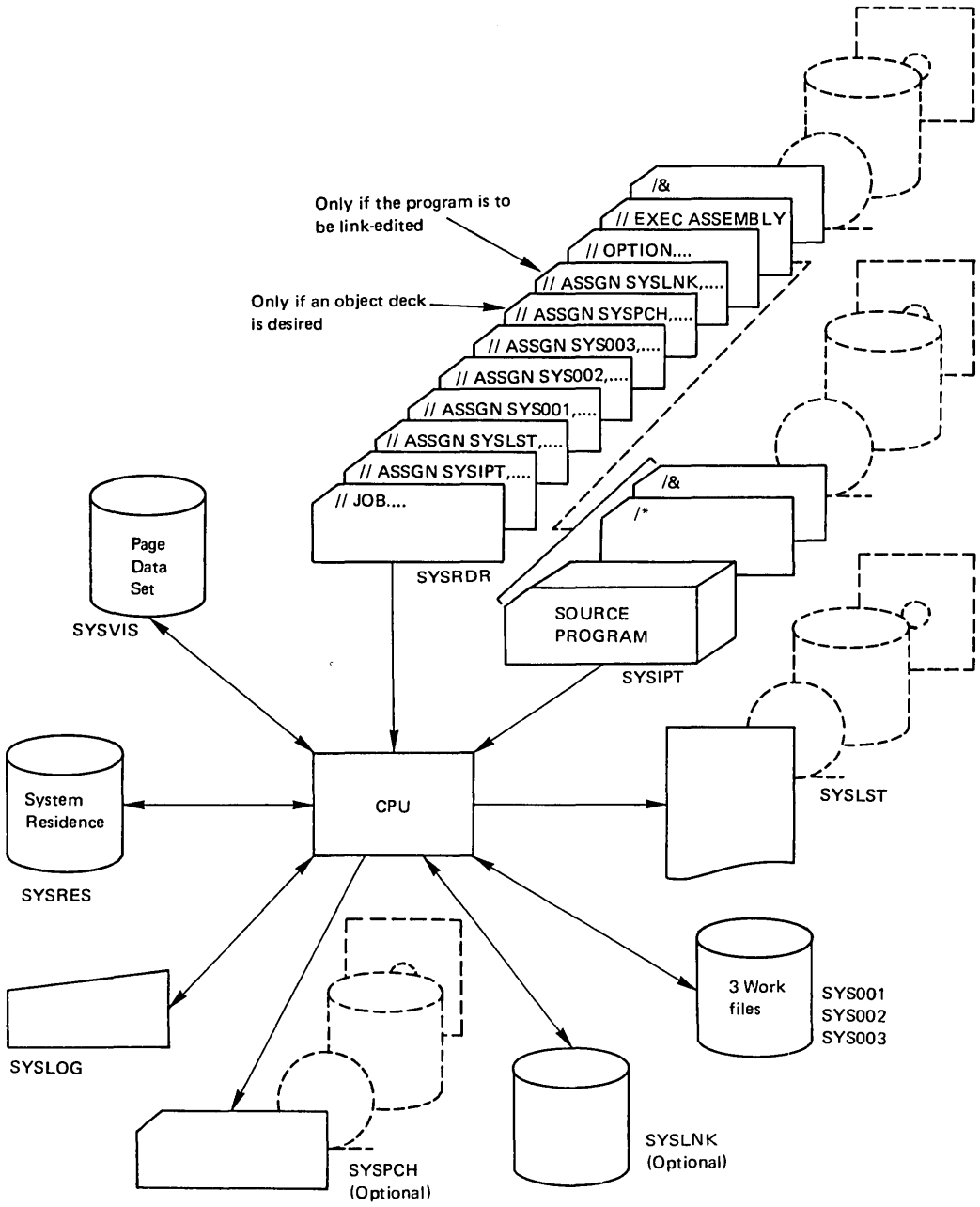


Figure 5.4. Device Assignments Required for an Assembly

1. These assignments will usually be standard, established during supervisor generation.
2. If SYSRDR and SYSIPT are assigned to the same device, the source input must be placed after the // EXEC ASSEMBLY card.

- The type of access method used to process the file; always coded as DU.

A diskette file can consist of a data area on one or more volumes; each volume can contain only one data area for a particular file. For each of these data areas, called extents, you must supply the following information on an EXTENT job control statement:

- The symbolic name of the device on which the volume containing the file is mounted.
- The serial number of the volume.
- The type of extent; always coded as 1.

In the following example, the program CREATE creates a diskette (DU) file named SALES that is to be retained for 30 days. The file comprises up to three diskettes. The diskettes have the volume serial numbers 111111, 111112, and 111113, and are mounted on the drive assigned to the symbolic device named SYS005.

```
// JOB EXAMPLE
// ASSGN SYS005,X'060'
// DLBL SALES,'MONTHLY',30,DU
// EXTENT SYS005,111111,1
// EXTENT SYS005,111112,1
// EXTENT SYS005,111113,1
// EXEC CREATE
/&
```

The job control program checks the DLBL and EXTENT statements for correctness and stores the supplied information in the label cylinder on SYSRES for the duration of the job (see the section *Editing and Storing Label Information,* later in this chapter).

Example for Submitting Label Information

Here is an example of how to code the job control statements required to create or access the labels for a diskette file. It is helpful if you are familiar with the formats of the DLBL and EXTENT job control statements as described in *DOS/VS System Control Statements.*

Assume that a program PROG100 needs a diskette file. The file consists of four extents; one extent is the diskette with serial number 000020, one is diskette 000030, one is diskette 000040, and one is diskette 000050. The following job stream shows the label statements required:

```
   // JOB SAMLABEL
   // ASSGN SYS005,X'060'
1  // DLBL FILNAME,'FILE ID',99/365,DU
   // EXTENT SYS005,000020,1
   // EXTENT SYS005,000030,1
   // EXTENT SYS005,000040,1
   // EXTENT SYS005,000050,1
2  // EXEC PROG100
3  /&
```

1 Only one DLBL statement is required. For each extent, one EXTENT statement must be supplied in the sequence in which the extents are processed.

2 Logical IOCS in PROG100 opens the first extent using the file name and file ID in

the DLBL statement, and the logical unit and volume serial number in the first EXTENT statement to locate the actual label on the disk pack. After PROG100 has processed the first extent, logical IOCS, based on the extent sequence number, opens the second extent.

Processing is identical for the third and fourth extents.

3 The /& statement causes the label information stored in the label information cylinder to be cleared. Thus, if the next job requires the same file, the label statements must be resubmitted (see *Types of Label Information*, later in this chapter and *Figure 5.6*).

Files on Direct Access Devices

After you have informed the system, via the ASSGN job control statement or command, which volume or physical device you want, you must supply the following information to allow the creation and checking of DASD labels:

1. A description of the characteristics of the file. You specify this in the DLBL job control statement.
2. The exact location of the file on the storage medium. You specify this in one or more EXTENT job control statements.
3. For non-sequential DASD files the amount of storage in the partition to be reserved for label processing. You specify this in the LBLTYP job control statement. Since this information is needed by the linkage editor, the LBLTYP statement is discussed in *Chapter 6: Linking Programs.*

The label information you supply in the DLBL job control statement may include the following:

- The name of the file. This name must be identical to the corresponding file name specified in your program. For programs written in assembler language this would be the name of the DTF (**D**efine **T**he **F**ile).
- An identification of the file which may include generation and version numbers of the file. This name is the one contained in the volume table of contents (VTOC) on the storage device. It is associated with the file name via a DLBL statement for the duration of a specific job or job step to make programs independent of physical files.
- The expiration date of the file.
- The type of access method used to process the file.
- An indication of whether or not a data secured file is to be created.
- The name of the catalog owning this VSAM file (valid only for VSAM).
- The buffer space to be allocated for this file (VSAM only).
- The blocksize to be used for this file on an IBM 3330-11 or 3350 device.

A DASD file can consist of one or more data areas on one or more volumes. For each of these data areas, called extents, you must supply the following information on an EXTENT job control statement:

- The symbolic name of the device on which the volume containing the file extent is mounted.

- The serial number of this volume.
- The type of the extent. An indexed sequential file, for instance, can consist of data areas, index areas, and overflow areas. For each of these areas an extent must be defined, and its type (data, index, or overflow) must be specified.
- The sequence number of the extent within the file.
- The number of the track (relative to zero) on which the file extent begins.
- The amount of space (in tracks) the file occupies.

In the following example, the program CREATE creates a sequential disk (SD) file named SALES that is to be retained until the end of 1975. The file comprises one extent of 190 tracks, starting on track number 1320. The disk pack has the volume serial number 111111 and is mounted on the drive assigned to the symbolic device name SYS005:

```
// JOB EXAMPLE
// ASSGN SYS005,DISK,VOL=111111,SHR
// DLBL SALES,'ANNUAL SALES RECORDS',75/365,SD
// EXTENT SYS005,111111,1,0,1320,190
// EXEC CREATE
/&
```

The job control program checks the DLBL and EXTENT statements for correctness and stores the supplied information in the label cylinders on SYSRES for the duration of the job or job step (see the section *Editing and Storing Label Information,* later in this chapter).

Examples for Submitting Label Information

Here are a number of examples of how to code the job control statements required to create or access the labels for the various types and organizations of DASD files. It is helpful if you are familiar with the formats of the DLBL and EXTENT job control statements as described in *DOS/VS System Control Statements.* Detailed information on the possible organizations and access methods for DASD files is given in *DOS/VS Data Management Guide.*

Sequentially Organized Disk Files (Single Drive). Assume that a program PROG100 needs a sequential disk file located on three different disk packs that are to be mounted successively on the same device (SYS005). The file consists of four extents: two on the pack with serial number 000020, one on pack 000100, and one on pack 000006. The following job stream shows the label statements required:

```
   // JOB SAMLABEL
   // ASSGN SYS005,DISK,VOL=000020
1  // DLBL FILNAME,'FILE ID',99/365,SD
   // EXTENT SYS005,000020,1,0,1320,190
   // EXTENT SYS005,000020,1,1,8,740
   // EXTENT SYS005,000100,1,2,1275,64
   // EXTENT SYS005,000006,8,3,50,636,6
2  // EXEC PROG100
3  /&
```

1 Only one DLBL statement is required. For each extent one EXTENT statement must be supplied in the sequence in which the extents are processed. The last extent occupies a split cylinder to illustrate that this is acceptable for sequential files.

2 Logical IOCS in PROG100 opens the first extent using the file name and file ID in the DLBL statement, and the logical unit and volume serial number in the first EXTENT statement to locate the actual label on the disk pack. After PROG100 has processed the first extent, logical IOCS opens the second extent, based on the extent sequence number.

For the third extent, volume serial number 000100 is specified while the volume currently mounted on SYS005 has the number 000020. The OPEN routine of LIOCS notifies the operator of this discrepancy, and the operator can mount the correct volume, at which time the OPEN routine regains control.

3 The /& statement causes the label information stored in the label information cylinder to be cleared. Thus, if the next job requires the same file, the label statements must be resubmitted (see *Types of Label Information* later in this chapter and *Figure 5.6*).

Sequentially Organized Disk Files (Multiple Drives). This example has the same requirements as the preceding 'Single Drive' example except that the three volumes are mounted on three different drives. The required job control statements are as follows:

```
   // JOB SAMLABEL
   // ASSGN SYS005,DISK,VOL=000020
   // ASSGN SYS006,DISK,VOL=000100
   // ASSGN SYS007,DISK,VOL=000006
1  // DLBL FILNAME,'FILE ID',99/365,SD
   // EXTENT SYS005,000020,1,0,1320,190
   // EXTENT SYS005,000020,1,1,8,740
   // EXTENT SYS006,000100,1,2,1275,64
   // EXTENT SYS007,000006,8,3,50,636,6
2  // EXEC PROG100
   /&
```

1 All label statements submitted are identical to the 'Single Drive' example except for SYSnnn in the EXTENT statements.

2 Logical IOCS opens each extent in the same way as described in the 'Single Drive' example except that processing does not stop for removal and mounting of packs, because enough devices are online to contain the file. A combination of this and the 'Single Drive' example could be used to reduce handling time without excessively increasing the total drive requirements.

DA Files. The program PROG101 processes a direct access file consisting of four extents contained on three disk packs. The three packs must be ready at the same time. The following job shows the label statements required to process the file:

```
   // JOB DALABEL
   // ASSGN SYS005,DISK,VOL=000065
   // ASSGN SYS006,DISK,VOL=000025
   // ASSGN SYS007,DISK,VOL=000002
1  // DLBL FILNAME,'FILE ID',99/365,DA
   // EXTENT SYS005,000065,1,0,1320,190
   // EXTENT SYS005,000065,1,1,80,740
   // EXTENT SYS006,000025,1,2,50,906
   // EXTENT SYS007,000002,1,3,1275,64
   // EXEC PROG101
   /&
```

1 The label statements follow the same pattern as for sequential files (described in the preceding examples) except that (1) the DLBL statement must specify DA to indicate direct access, and (2) split cylinder mode cannot be used for direct access files.

Note: *If program PROG101 is a prior DOS self-relocating program, a // LBLTYP NSD(4) statement must be included immediately preceding the EXEC PROG101 statement.*

Files on Magnetic Tape

Files on magnetic tape can be processed with or without labels. For tape files with IBM standard labels, the label information must be submitted through the TLBL job control statement. (A tape file can also have standard-user or non-standard labels; for these labels no job control statements are required. More information on tape labels is given in *DOS/VS Data Management Guide.*)

The standard label information submitted in the TLBL statement may include the following:

- The name of the file. This name must be identical to the corresponding filename (DTF name) specified in your program.
- An identification of the file.
- Creation date for input and expiration date (or retention period) for output files.
- The volume serial number of the tape reel that contains the file.
- For files that extend over more than one volume, the sequence number of the volume.
- For volumes that contain more than one file, sequence number of the file.
- The version and modification number of the file.

When a program that processes tape files with standard labels is to be link-edited, you must supply a LBLTYP job control statement to define the amount of storage required in the partition for label processing (see also *Chapter 6: Linking Programs).*

As with DASD files, the label information you supply in the TLBL job control statement is checked and stored in the label information cylinders on SYSRES for the duration of the job or job step (see *Editing and Storing Label Information* later in this chapter).

Controlling Magnetic Tape Operation

The MTC job control statement or command controls certain magnetic tape operations, for example, file positioning. Files on magnetic tape are almost invariably processed sequentially. This means, for example, that if you have five files on one tape reel and you want to process the last one, you have to read four files before you can access the one you need. Since this is time consuming, however, you can instruct the job control program to position the tape at any particular file.

The MTC job control statement or command controls operations such as:

- Spacing the tape backward or forward to the required file.
- Spacing the tape backward or forward a specified number of records.
- Rewinding the tape to the beginning.
- Writing a tapemark to indicate the end of a file.

In the following example, program PROGA creates a labeled tape file named RATES on tape volume 222222. At the end of the first job step, an MTC job control statement is used to rewind (REW) the tape to the beginning so that the newly created file can be processed by PROGB.

```
// JOB TAPE
// ASSGN SYS004,TAPE,VOL=222222
// TLBL RATES,'MASTER',75/365,222222
// EXEC PROGA
// MTC REW,SYS004
// EXEC PROGB
/&
```

Controlling Printed Output

Most of the DOS/VS supported printers use a forms control buffer (FCB) to control the length of forms skips. In addition, printers may be equipped with the universal character set feature, which is controlled by a universal character set buffer (UCB). Examples of printers equipped with these buffers are the 3203 and 3211 printers.

The buffers of these printers must be loaded during, or immediately after, IPL and they may have to be reloaded later between job steps or, occasionally, while a job step using the printer is being executed.

The following methods for loading the buffers are available:
To load the FCB

- Automatic loading during IPL
- Using the SYSBUFLD program between job steps or immediately after IPL
- Using the LFCB command
- Using the LFCB macro in the problem program.

To load the UCB

- Automatic loading during IPL (applies to 3203, 3211, and 5203U printers)
- Using the SYSBUFLD program between job steps or immediately after IPL
- Using the LUCB command
- Using the UCS command (only applies to a 1403 UCS printer).
- Using the FCB parameter in the POWER/VS * $$ LST statement.

The method of loading the buffers by using the SYSBUFLD program offers the advantage that hardly any operator activity is involved; however,

loading the buffers by using the LFCB or LUCB command does not require the operator to wait for a partition to finish processing.

When the contents of an FCB or a UCB are replaced by a new buffer load, the system uses this new buffer load to control printed output until the buffer is reloaded (or until the next IPL). None of the above methods provides automatic resetting of the buffer load to the original contents. It may be necessary to reset the buffer load to the original contents before taking a storage dump, to ensure that the dump is printed in the correct format, without any part of it being left out.

Details on how to load the FCB and UCB are contained in *DOS/VS System Control Statements.*

Editing and Storing Label Information

The job control program checks the DLBL, EXTENT, and TLBL statements for correctness and stores the supplied label information in the label information cylinders on SYSRES. Label information (DLBL and EXTENT) for a sequential disk file is written after each EXTENT statement is checked; however, all EXTENT statements for a non-sequential disk file are processed prior to storing on the label information cylinders. When the program that processes the file is executed, the data management routines access the label data in the label information cylinders

1. to write the appropriate labels onto the storage volume, if the file is to be created, or

2. if an existing file is to be processed, to check the contents of the label information cylinders against the label(s) of the file to ensure that the correct volume is mounted, that no unexpired files are overwritten, etc.

Detailed information on labels and label processing is given in *DOS/VS Data Management Guide, DOS/VS DASD Labels*, and *DOS/VS Tape Labels.*

Types of Label Information

Label information can be stored in the label cylinder either temporarily (for the duration of one job or job step) or permanently (until the next IPL). In addition, label information can either be dedicated to a single partition or it can be accessed by all partitions. For the 3340, label information can also be stored permanently on a second, adjacent cylinder which can be accessed by all partitions.

The various types of label information are controlled by the following three options of the OPTION job control statement:

USRLABEL causes all DASD, diskette, and tape label information to be stored temporarily for one job or job step. The label information is accessible only by the partition in which it was submitted. User label information submitted at the beginning of one job step can be used in subsequent job steps, unless it is overwritten by label information

submitted for an intermediate job step. When label information is submitted in an intermediate job step, the USRLABEL area for that partition is cleared and only label information submitted by the intermediate job step is written in the USRLABEL area. Therefore, it is a good idea to inculde all TLBL, DLBL, and EXTENT statements in the first step of a job (preceding the // EXEC statement). If no option is specified, or if the OPTION statement is omitted, USRLABEL is assumed.

PARSTD causes all DASD, diskette, and tape label information to be stored permanently for all subsequent jobs. The label information is accessible only by the partition in which it was submitted.

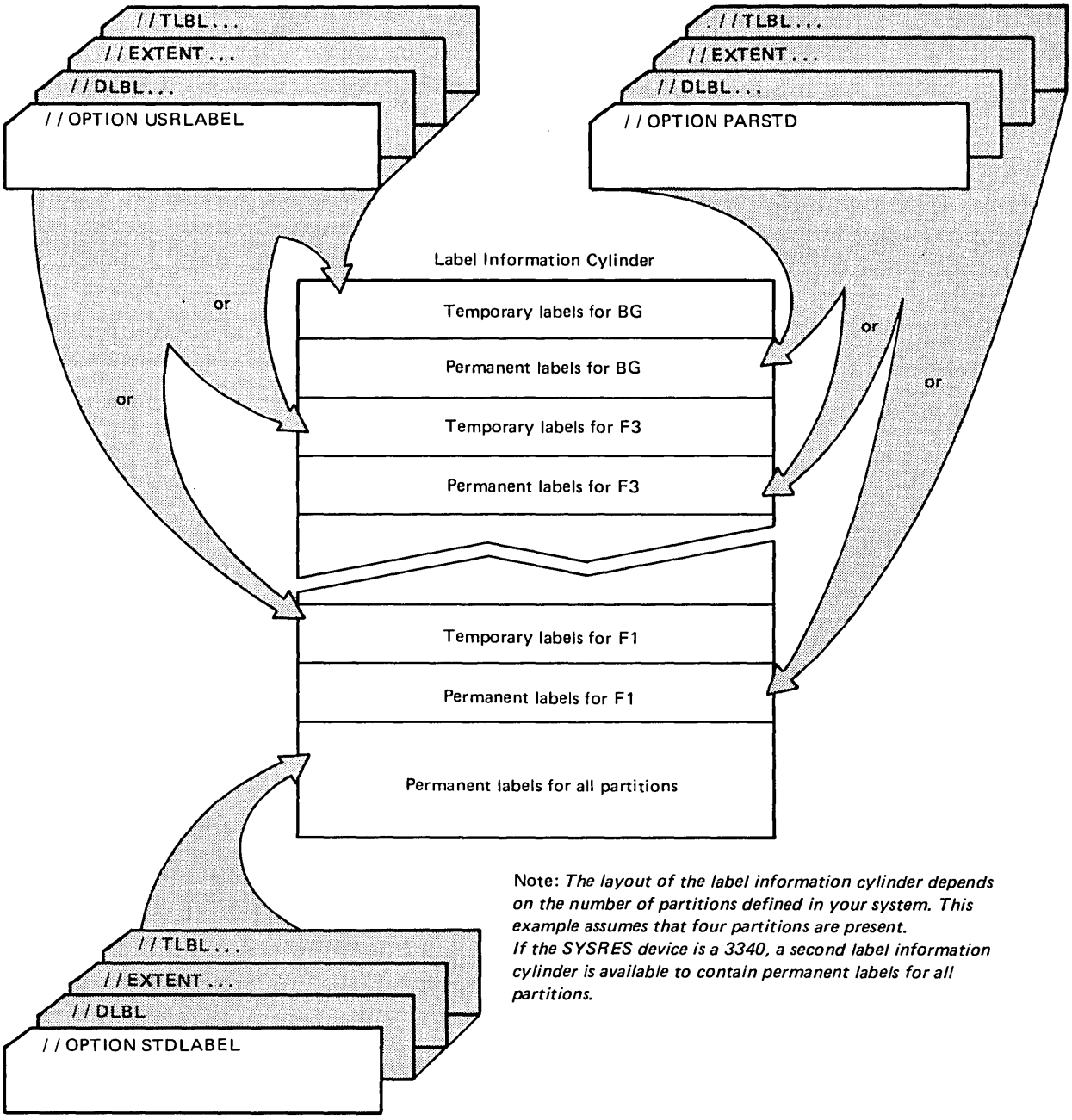
STDLABEL causes all DASD, diskette, and tape label information to be stored permanently for all subsequent jobs. The label information is accessible by all partitions but can only be submitted in the background partition. This ensures that the label information cylinder(s) is/are not updated simultaneously by two partitions. Symbolic logical units contained in the submitted label information must not be greater then the highest symbolic logical unit specified for background at system generation.

Each type of label information is stored in a separate area of the label information cylinder(s) depending on the specified option. This is illustrated in Figure 5.5. The system searches the label information cylinder(s) in the following sequence:

(1) user label information,
(2) partition label information, and
(3) standard label information.

It is important to distinguish between (1) the period of time for which a label option is in effect and (2) the period of time for which the label information is retained on the label information cylinder(s). For example, the label data submitted following an OPTION statement with the PARSTD option is retained for all subsequent jobs until overwritten by another PARSTD option, but the PARSTD option is canceled at the end of the job or job step in which it was specified. This is shown more clearly in the summary of label options in Figure 5.6.

By storing the label information for a disk file on the label cylinder(s), DOS/VS relates that file to the type of the device which is assigned to the pertinent logical unit when this file is processed for the first time. A later attempt to use this label information for the same file (and extent) on a different device type causes DOS/VS to cancel the job. If a different device type has to be used for a file whose label information is stored on the label cylinder(s), DOS/VS requires that the original label statements be resubmitted with the pertinent logical unit assigned to an extent on a device of the new type.



Note: *The layout of the label information cylinder depends on the number of partitions defined in your system. This example assumes that four partitions are present. If the SYSRES device is a 3340, a second label information cylinder is available to contain permanent labels for all partitions.*

Figure 5.5. Storing Label Information in the Label Information Cylinder(s)

Option[1]	Type of label information	Option in effect until	Label information retained	For
USRLABEL[2]	temporary	STDLABEL or PARSTD is specified.	for one job. The / & statement causes the temporary label area to be cleared.[5]	the partition in which the option was specified.
PARSTD	permanent	a) end of job step b) end of job c) USRLABEL or STDLABEL is specified.	for all subsequent jobs until another PARSTD option is used.[3]	the partition in which the option was specified.
STDLABEL	permanent	a) end of job step b) end of job c) USRLABEL or PARSTD is specified.	for all subsequent jobs until another STDLABEL option is used.[3]	all partitions.[4]

[1] Search sequence is USRLABEL, PARSTD, and STDLABEL.

[2] If no option is given or if the OPTION statement is omitted, USRLABEL is assumed.

[3] All label information submitted following a PARSTD or STDLABEL option is written at the beginning of the label area thus destroying any previously stored information. Therefore, if you want to add label data for another file, all previously stored label information that is to be kept must be resubmitted.

[4] Label information stored with the STDLABEL option is available to all partitions but can be submitted only through background programs.

[5] Additional label information from a subsequent job step will overlay previous label information.

Figure 5.6. Summary of Label Option Functions

Summary of Job Control Statements and Commands

The following summarizes the functions of those job control statements and commands needed to handle I/O devices and files, as discussed in the preceding section. Also included are a number of commands that can be used by the operator to manipulate I/O devices. **Note:** *The previous forms of label information statements (DLAB, VOL, XTENT, TPLAB) are still supported, except when you use 3330, 3340 or 3350 disk drives. However, when new statements are prepared, DLBL, EXTENT, and TLBL should be used.*

ASSGN

The ASSGN statement or command is used to connect a logical I/O unit to a general device class, a specific device type, a physical device or a list of physical devices, or another logical unit. An ASSGN statement or command can also be used:

- to specify a temporary or permanent assignment.
- to specify a volume serial number for a tape, disk, or diskette.
- to specify that a disk is shareable by more then one partition or logical unit.
- to unassign a logical unit to free it for assignment to another partition.
- to ignore the assignment of a logical unit, that is, program references to the logical unit are ignored (useful in testing and certain rerun situations).

- to specify an alternate tape unit to be used when the capacity of the original is reached.

The assignment routines check the operands of the ASSGN statement/command for the relationship between the physical device, the logical unit, the type of assignment (permanent or temporary), etc. The following list summarizes the most pertinent items to remember when making assignments:

1. Assignments are effective only for the partition in which they are issued.
2. Apart from the operator console, no physical device except DASD can be assigned to more than one active partition or logical unit at the same time.
3. All system input and output file assignments to disk or diskette must be permanent.
4. SYSIN must be assigned if both SYSRDR and SYSIPT are to be assigned to the same extent.
5. SYSOUT cannot be assigned to disk or diskette; it must be a permanent assignment if assigned to tape.
6. SYSLNK must be assigned before issuing the LINK or CATAL option in the OPTION statement; otherwise, the option is ignored and the message 'PLEASE ASSIGN SYSLNK' is issued to the operator.
7. If SYSRDR, SYSIPT, SYSLST, or SYSPCH is assigned to tape, diskette, or disk when the system is generated, it will be unassigned by IPL. Such assignments can be made effective only with the job control ASSGN statement or command, because ASSGN also opens the file.
8. Before a tape unit is assigned to SYSLST, SYSPCH, or SYSOUT, all previous assignments to this tape unit must be permanently unassigned. This may be done by using a DVCDN command instead.
9. The assignment of SYSLOG cannot be changed while a foreground partition is active.
10. SYSRES, SYSCAT, and SYSVIS can never be assigned by an ASSGN statement or command. An IPL is required to change these assignments.

RESET

The RESET statement or command can be used to reset temporary assignments to standard or permanent. With one RESET statement or command you can reset

- all logical units, or
- all system logical units, or
- all programmer logical units, or
- one specific system or programmer logical unit.

The RESET statement is effective only for the partition in which it is issued.

LISTIO

With the LISTIO statement or command you can obtain a listing of the current status of all I/O assignments in your system.

DVCDN — The DVCDN (device down) command informs the system that a device is no longer physically available for system operations.

When the device becomes available again for system operations a DVCUP (device up) command must be given before new assignments can be made.

DVCUP — The DVCUP (device up) command informs the system that a device is available for system operations after it has been down.

DLBL — One DLBL statement is required for each DASD or diskette file to be processed. This statement and its associated EXTENT statement(s) are used for checking or creating DASD and diskette file labels.

EXTENT — One extent statement must be supplied for each area (extent) of a DASD file or each volume of a diskette file. The EXTENT statement(s) must directly follow the associated DLBL statement.

TLBL — For tape files with standard labels, a TLBL statement must be supplied for checking or creating the standard label.

MTC — The MTC statement or command can be used to control magnetic tape operation. For example, a tape can be rewound to the beginning or it can be positioned to a certain file or record.

LFCB — The LFCB command causes the system to load the specified FCB image from the core image library into the FCB of the printer for which the command was issued.

LUCB — The LUCB command causes the system to load the specified UCB image from the core image library into the UCB of the printer for which the command was issued.

Executing a Program

After you have properly defined the I/O requirements of your program to the system you can instruct job control to prepare your program for execution. How this is done and how the supplied information is processed is described in the following section.

Assembling, Link-Editing, and Executing a Program

In DOS/VS, three processing steps are necessary to obtain results from a problem program once the source program has been written:

1. Assembly or compiling of the source program into an object module. (Object modules are discussed in *Chapter 6: Linking Programs.*)

2. Link-editing of the object module to form an executable program phase (see *Chapter 6: Linking Programs*).
3. Execution of the program phase.

Each of these steps is initiated by the job control program in response to an EXEC job control statement. The EXEC statement must be the last of the job control statements submitted for any one job step. Figure 5.7 shows an example of the job control statements needed to assemble, link-edit, and execute a source program.

```
   // JOB EXECUTE
1  // OPTION LINK
2  // EXEC ASSEMBLY
3  // LBLTYP TAPE
4  // EXEC LNKEDT
5  // EXEC
   /&
```

1 To link-edit and execute a program in the same job, the LINK option must be specified in the OPTION job control statement.

2 The assembler is fetched from the core image library and starts execution.

3 Required to reserve a partition area for processing tape labels at execution time.

4 The linkage editor is fetched from the core image library and starts execution.

5 If an EXEC statement without a program name is encountered, the program last stored (if stored within the same job) in the core image library by the linkage editor is fetched for execution (see also *Preparing Programs for Execution*).

Figure 5.7. Job Control Statements to Assemble, Link-edit, and Execute a Program in one Job

If SYSRDR and SYSIPT are assigned to the same device, and you wish to submit data to your program via SYSIPT, the data cards must follow the corresponding EXEC job control statement. For example, the data processed by the assembler is your source program which must follow the // EXEC ASSEMBLY statement. The end of the input data submitted for one program must be indicated by a /* (end-of-data) statement. The /* statement is not processed by job control but is read by the processing program. (Note: For an input file on an IBM 5424 MFCU, the /* card must be followed by a blank card.) The placement of input data and the /* statement is shown in Figure 5.8.

```
// JOB INPUT
// OPTION LINK
// EXEC ASSEMBLY
    .
    .
source program
    .
    .
/*
// LBLTYP
// EXEC LNKEDT
// EXEC
    .
    .
input data for user program
    .
    .
/*
/&
```

Figure 5.8. Submitting Input Data on SYSIPT

How the job shown in Figure 5.8 is processed by the system is illustrated in Figure 5.9. The inclusion of SYSIPT data in job streams in the procedure library is described in the section *SYSIPT Data in Cataloged Procedures.*

1 Job control reads the JOB statement and stores the job name in the communications region in the supervisor. Other functions of the JOB statement are described under *Defining a Job,* earlier in this chapter.

2 Job control reads the OPTION statement with the LINK option and sets the LINK bit in the supervisor. This indicates

   a) to the assembler, that the assembled object module is to be written onto SYSLNK,

   b) to the linkage editor, that the executable program is to be stored in the core image library only temporarily for execution in the same job.

3 On encountering the // EXEC ASSEMBLY statement, job control transfers control to the supervisor passing it the name of the assembler program.

4 The supervisor loads the assembler into the partition, overlaying job control.

5 The assembler reads the source program, assembles it, and stores the object module on SYSLNK (not shown).

6 The assembler transfers control to the supervisor.

7 The supervisor loads job control into storage, overlaying the assembler.

8 Job control reads the // EXEC LNKEDT statement, as well as any preceding linkage editor statements, and transfers control to the supervisor, passing it the name of the linkage editor.

9 The supervisor loads the linkage editor into storage, overlaying job control.

10 The linkage editor reads the object module from SYSLNK and link-edits it.

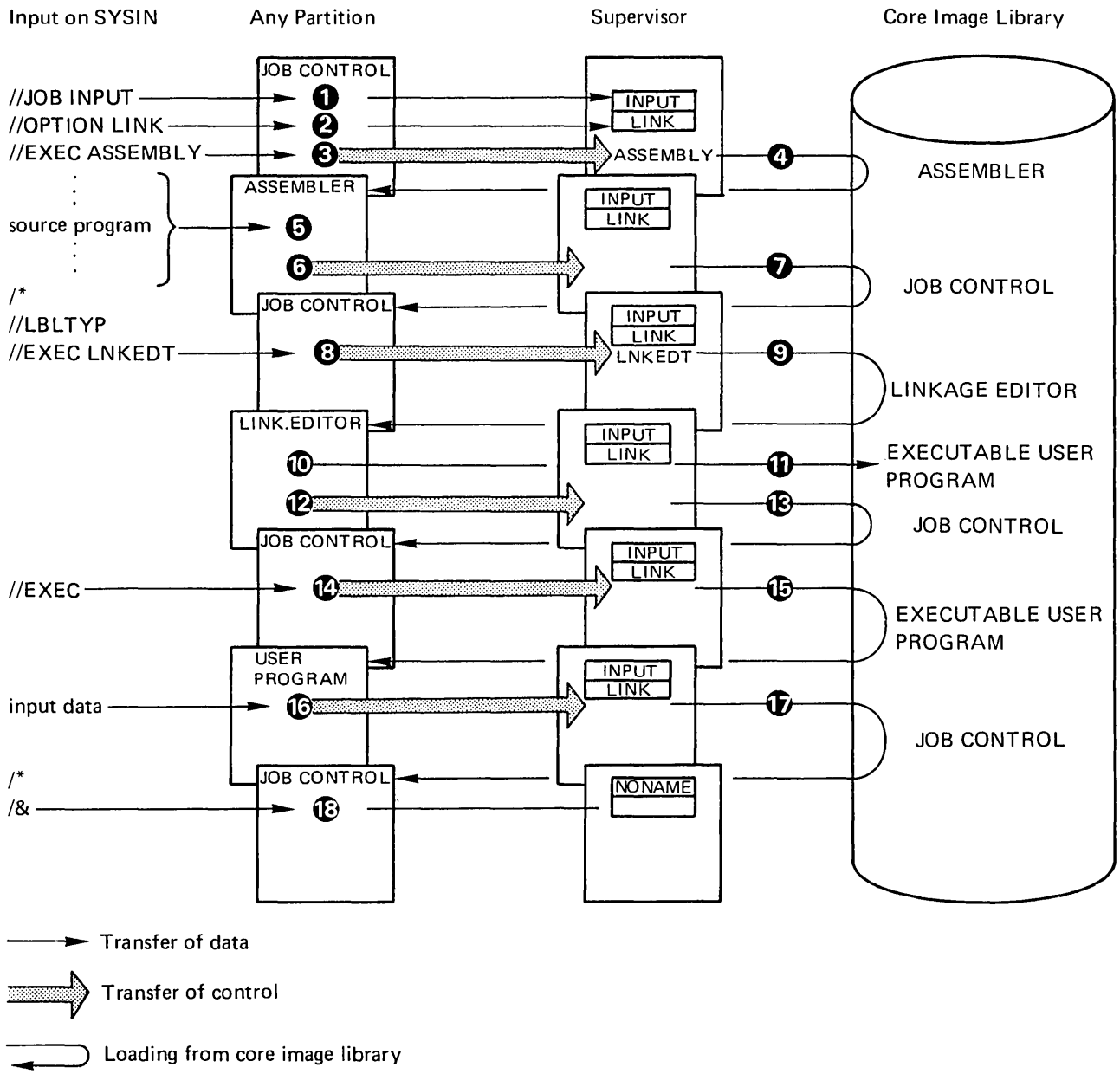


Figure 5.9. System Operation of an Assemble, Link-Edit and Execute Job

11 The linkage editor stores the executable program in the core image library.

12 The linkage editor transfers control to the supervisor.

13 The supervisor loads job control into storage.

14 Job control reads an EXEC statement without a program name.

15 The program last stored in the core image library by the linkage editor to be loaded and executed. (See also *Preparing a Program for Execution*).

16 The user program is executed. It reads and processes the data from SYSIPT and at EOJ relinquishes control to the supervisor.

17 The supervisor loads job control.

18 When job control reads the / & statement, it cancels the LINK option and replaces the jobname by NONAME in the communications region. Other functions of the / & statement are described under *Defining a Job,* earlier in this chapter.

Executing Cataloged Programs

Programs can be cataloged permanently in the core image library after they have been assembled and link-edited. This saves assembling and link-editing the program for every run.

Cataloging into the core image library is done by the linkage editor in response to an OPTION job control statement with the CATAL option (see *Chapter 6: Linking Programs*).

To execute a cataloged program you use an EXEC job control statement specifying the name under which the program was cataloged (as shown for the assembler and linkage editor in the preceding example).

For example, the following job executes a program that was cataloged in the core image library under the name PROGA; data cards are submitted on SYSIPT:

```
// JOB CAT
    .
    .
assignment and label
statements, if required
    .
    .
// EXEC PROGA
    .
    .
input data
    .
    .
/*
/&
```

Preparing Programs for Execution

Before any program can be executed it must be stored in the core image library by the linkage editor. Programs are stored either temporarily or permanently, depending on the option specified in the OPTION job control statement:

- If the LINK option is specified, the program is stored temporarily for immediate execution, in the same job. This program will be overwritten by the next program that is link-edited.
- If the CATAL option is specified, the program is stored permanently and can be executed any time after the catalog job. It can be deleted only by the library maintenance program (see *Chapter 7: Using the Libraries*), or by another program cataloged with the same name.

These two situations require different preparations for the loading of a program into a partition Figure 5.10 shows the functions performed by the linkage editor and the job control program to load programs into storage.

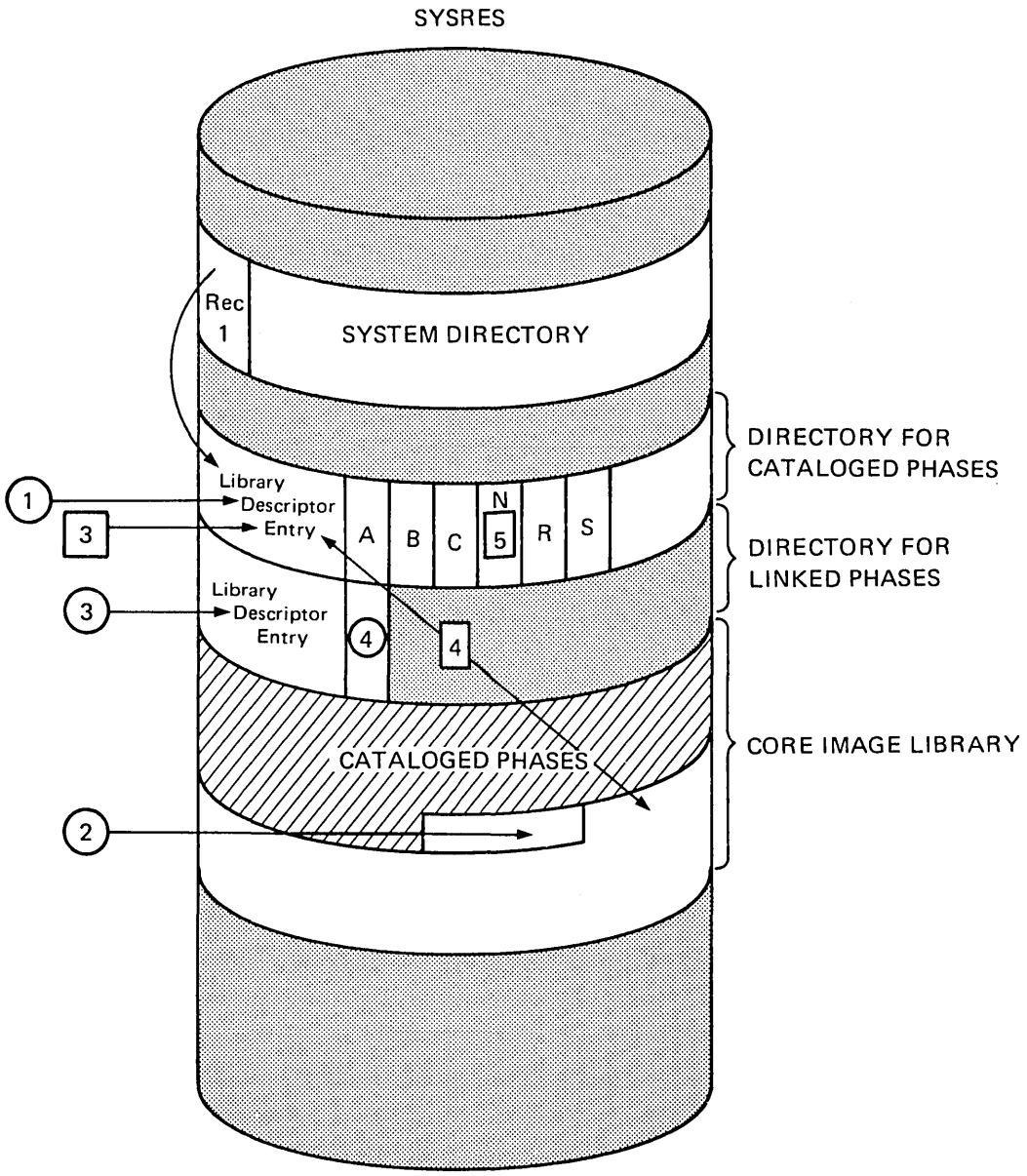


Figure 5.10. Preparing the Loading of Temporarily and Permanently Stored Programs

The core image directory comprises two directories: one for cataloged phases, and one for linked phases. The directory for linked phases begins at the first unused track of the core image directory.

`// OPTION LINK` **Linkage Editor**

(1) Uses the information in the library descriptor entry of the core image directory for cataloged phases to determine the first available block in the core image library.

(2) Stores the phase in the core image library.

(3) Updates the library descriptor entry of the core image directory for linked phases to indicate the first phase link-edited in the job step (in case of multiple phases).

(4) Makes a directory entry in the core image directory for linked phases, inserting this entry in alphameric sequence (in case of multiple phases).

// EXEC

Job Control

Uses the information in the library descriptor entry of the core image directory for linked phases to check which phase was the first link-edited and passes this information to the supervisor, which loads this phase into the partition.

Note: *The next phase link-edited (OPTION LINK or OPTION CATAL) into the core image library will overwrite the one just temporarily stored.*

// OPTION CATAL

Linkage Editor

(1) (2) Same as for OPTION LINK.

[3] Updates the library descriptor entry of the core image directory for cataloged phases to indicate the first phase link-edited in the job step (in case of multiple phases).

[4] Updates the library descriptor entry of the core image directory for cataloged phases to indicate the new address of the next available block in the core image library.

[5] Makes a directory entry in the core image directory for cataloged phases, inserting this entry in alphameric sequence.

// EXEC NAME

Job Control

Locates the corresponding entry in the core image directory for cataloged phases and passes this information to the supervisor, which loads the phase into the partition.

Note: *If no phase name is specified in the EXEC card, job control uses the information in the library descriptor entry of the core image directory for cataloged phases to check which was the first phase link-edited in this job step.*

Defining Options for Program Execution

In the preceding section, it was shown how the OPTION job control statement can be used

- to specify the type of label information to be stored for a file (USRLABEL, PARSTD, STDLABEL options), and

- to define whether a link-edited program is to be stored temporarily or permanently in the core image library (LINK, CATAL options).

There are a number of additional functions which you can invoke through the OPTION job control statement. The most important ones are:

- To log all job control statements submitted to the system on SYSLST. This faciliates diagnosing the job control statements in case of an error. The option is LOG.
- To dump the contents of the registers, the supervisor area, and the current partition (real or virtual) on SYSLST in case of abnormal program termination. This is useful for debugging. The option is DUMP.
- To cancel a job if an I/O assignment cannot be performed. The option is ACANCEL. (Note: If this option is suppressed, control is passed to the operator.)
- To put an object deck on SYSPCH. The object module can then be combined with other object modules by the linkage editor to form one executable program, or it can be used as input to the library maintenance program to catalog it into the relocatable library. The option is DECK.
- To print various listings produced by the language translators on SYSLST. These listings include object code, symbol table, cross-reference, and error lists which are useful debugging aids during the test period of a program. Among the possible options are LIST, LISTX, SYMA, and XREF.

Each of these options can be suppressed by specifying the prefix NO (for example, NOLIST, NODUMP). A complete list of the available options is given in *DOS/VS System Control Statements.*

You can establish a standard set of these options during supervisor generation by using the STDJC macro. Standard options are valid for all jobs unless superseded by an OPTION job control statement. Options specified in an OPTION statement remain in effect until (1) a contrary option is read or (2) a JOB or / & statement is encountered which resets the option to standard.

Communicating with Problem Programs via Job Control

Via job control a problem program can take a specific path of action dependent on some external event. Such an instruction is given at job control time by setting program switches in the communications region which can be tested by the problem program at execution time.

For example, an accounting program that prepares reports of daily, weekly, and monthly accounts can be instructed through these program switches when the weekly or monthly reports are due.

The program switches are set at job control time by the UPSI (user program switch indicator) job control statement. The specific meaning attached to each bit in the UPSI byte depends on the design of the problem program. When a JOB or / & statement is encountered, the UPSI byte is reset to zero.

Controlling Jobs in a Multiprogramming System

After IPL, the job control program is always loaded automatically into the virtual background partition. It is loaded into a foreground partition in response to a BATCH or START command issued by the operator and specifying the required partition. (More information on the operator commands that control partitions is given in *DOS/VS Operating Procedures.*)

A program is always loaded into the partition in which job control was loaded (or in the corresponding real partition).

If the program is relocatable and the relocating loader is supported in the system, the program can run in any partition. If the program (or single phase) is reenterable and resident in the shared virtual area, it can be shared by programs in more than one partition.

The relocating loader and self-relocating programs are discussed in *Chapter 6: Linking Programs.*)

Reserving Storage for VSAM

For VSAM, there are two general areas for storage considerations. First, Access Method Services must be utilized for file definitions, catalog manipulation and other VSAM file utility functions. Access Method Services modules cannot be loaded into the SVA and therefore have a virtual partition requirement that depends on the functions required for the current job. A partition GETVIS area must be provided by specifying SIZE=AUTO on the EXEC statement for Access Method Services. For further details, refer to the DOS/VS *Access Method Services User's Guide.*

Secondly, when user programs access VSAM files, the VSAM modules may be loaded into either the partition GETVIS area or the shared virtual area. For best performance, it is recommended that the SVA be used. This also reduces storage requirements for your virtual partition. The partition in which VSAM files are to be processed must allow for a GETVIS area to accomodate VSAM buffers and control blocks. Approximately 302K is required for VSAM modules in the SVA, while the partition must be large enough to accomodate the user program and the GETVIS area. The size of the partition GETVIS area depends on the number of VSAM files being accessed as well as their control interval sizes. For specific details on VSAM storage requirements, refer to the VSAM Module in the *DOS/VS System Generation* manual.

Reserving Storage for RPS

For programs using RPS (rotational position sensing), part of the virtual partition in which the program is to be executed must be reserved to accommodate the RPS DTF extensions. This is done by the SIZE parameter of the EXEC job control statement. These DTF extensions vary in size from a minimum of 256 bytes to a maximum of 512 bytes.

Example of a program requiring 75K:

```
// JOB WEEKLY
   .
   .
// EXEC WEEKEND,SIZE=AUTO
/&
```

If the job WEEKLY runs in a virtual partition of 100K, the program WEEKEND will occupy 76K as calculated by the system, while the remaining 24K are reserved as an additional storage pool, also available to RPS support for DTF extensions.

The RPS version of logic modules are loaded into and executed out of the SVA. The SVA must be large enough to accommodate the RPS versions of the logic modules and the GETVIS area of the SVA must have an additional 2K for the LDL (local directory list) used by RPS. (The GETVIS area must have this 2K space even if all the RPS logic modules are preloaded into the SVA.) The sizes of the SVA and of the GETVIS work area can be specified in the SVA parameter of the VSTAB macro during supervisor generation. This specification can be overridden by the SET SVA command issued immediately after IPL.

The RPS versions of the logic modules are contained in the core image library of the distribution medium. They can either be loaded into the SVA at IPL time or loaded dynamically as needed into the GETVIS area in the SVA at execution time. For a user who loads frequently used RPS versions of the logic modules into the SVA at IPL time, a typical specification might be SVA=(88K,12K) for the SVA and GETVIS area, respectively. While this might be a typical value, it is not intended to be totally representative of *every* RPS situation.

If there is insufficient virtual storage in either the user area, for the DTF extension, or in the GETVIS area of the SVA for the RPS version of the logic module, the file will be opened without RPS support and processing will continue.

Teleprocessing Balancing

The use of teleprocessing and batch processing at the same time may occasionally result in long or erratic teleprocessing response times. This may be especially true if you have overcommitted real storage, thus causing excessive paging. The teleprocessing application may have to compete so strongly for real page frames (because of high processing activity in the batch partitions) that response time increases substantially.

Teleprocessing balancing improves response time by trading off teleprocessing response time against batch throughput. TP balancing tends to reduce response times, or at least to stablize them.

After IPL, TP balancing can be activated by the operator's issuing the TPBAL command, which specifies the number of batch partitions that can tolerate delayed processing. These will be the lowest priority partitions. The TPBAL command is also used to change or display the current setting. For more information, see the *DOS/VS Operating Procedures.*

Once activated, the TP balancing function can be invoked by using TPIN/TPOUT macros. Refer to *Balancing Teleprocessing* in *Chapter 9: Designing Programs for Virtual-Mode Execution* for more details.

Restarting a Program from a Checkpoint

When you expect a program to run for an extended period of time, you can make provisions for taking checkpoint records periodically during the run. These records contain the status of the job and system at the time the records were written. Thus, they provide a means of restarting at some point rather than at the beginning of the job if, for any reason, processing is terminated before the normal end of the job.

Checkpoints are taken by means of a macro which you specify in your source program. How this is done is described in *Chapter 10: Using the Facilities and Options of the Supervisor*. To restart a program from a checkpoint the RSTRT job control statement is used. The sequence of job control statements that must be submitted to restart a program is as follows:

1. A JOB statement specifying the jobname used when the checkpoints were taken.
2. ASSGN statements, if necessary, to establish the I/O assignments for the program that is to be restarted.
3. A RSTRT statement specifying
   a) the symbolic name of the tape or disk device on which the checkpoint records are stored,
   b) the sequence number of the checkpoint record to be used for restart,
   c) for checkpoint records on disk, the filename (DTF name) of the checkpoint file.
4. An end-of-job (/ &) statement.

Figure 5.11 shows the sequence of job control statements needed to restart a checkpointed program that ended abnormally due to, for example, a power failure. Following are the characteristics of the checkpointed program that must be considered for restart:

- The job name specified in the JOB statement was CHECKP; the same name must be used for restart.
- The checkpoint records were written on magnetic tape; therefore, no filename need be specified in the RSTRT statement.
- The symbolic device name SYS005 was used for the checkpoint file; this name may be different for restart.
- The sequence number of the last checkpoint record written was 0013; this or any previous checkpoint record can be used for restart. (The sequence numbers are supplied by the checkpoint routine.)

```
// JOB CHECKP
// ASSGN SYS006,X'380'     CHKPT TAPE
// ASSGN ...
// ASSGN ...
// RSTRT SYS006,0013
/&
```

Figure 5.11. Example of a RESTART job

Additional restart considerations are given in *Chapter 10: Using the Facilities and Options of the Supervisor.*

Programs that reserve virtual storage with the SIZE operand of the EXEC job control statement, and allocate this storage with the GETVIS macro instruction, should checkpoint the full virtual partition to ensure a valid restart. Programs using VSAM, the ISAM interface program, or Access Method Services should checkpoint the full virtual partition since these programs use the reserved virtual storage. Programs using RPS support for SAM, DAM, ISAM, and VSAM must checkpoint the entire virtual partition. In addition, any RPS I/O phases to be used by the checkpointing program must be preloaded into the SVA. (See *Saving Data for Restart* in *Chapter 10: Using the Facilities and Options of the Supervisor* for additional Checkpoint/Restart considerations.)

Executing in Virtual or Real Mode

All programs invoked for execution through an EXEC job control statement are executed in virtual mode in the same virtual partition as the job control program. You can, however, force a program to run in real mode, that is, the program is executed in a real partition and no paging is performed. To run a program in real mode, you must specify the REAL operand in the EXEC statement. Example:

```
// JOB NAME
    .
    .
// EXEC PROGA,REAL
/&
```

If, for the above example, job control runs in virtual partition F2, then the program PROGA will be loaded and executed in real partition F2. This requires that the real partition F2 be large enough to hold the entire program PROGA. For all the considerations for enabling a program to run in a real partition see *Chapter 6: Linking Programs.*

If a program in real mode is smaller than its associated real partition the unused portion of that partition, should be given to the page pool by specifying the size of the program in the SIZE operand of the EXEC statement. Example:

```
// JOB NAME
    .
    .
// EXEC PROGA,REAL,SIZE=30K
/&
```

If the program PROGA which is 30K bytes long runs in a 50K real partition, the remaining 20K bytes of that partition will be given to the page pool.

If you specify SIZE=AUTO job control automatically uses the information in the core image directory entry to calculate the size of the program to be loaded. If you specify SIZE=(AUTO,nK) job control adds nK bytes to the calculated length. This is especially useful for programs that dynamically allocate storage during execution (such as compilers).

Running programs in real mode implies temporarily forfeiting a number of page frames in the page pool, which may lead to degradation of system throughput. Therefore, real mode execution should be used sparingly.

If phase names are present in the system directory list, a main page pool of at least 4K bytes must be available. If phases resident in the shared virtual area are to be executed, a main page pool of at least 18K must be available. For further details on page pool requirements, refer to *Defining the Size of Real Partitions* in chapter 3: *Planning the System.*

With a few exceptions, all IBM-supplied and user-written programs can be executed under DOS/VS either in virtual or real mode. These exceptions are listed in the following two sections.

Programs that Must Run in Virtual Mode

Besides job control, which always runs in a virtual partition, POWER/VS and all programs using VTAM, VSAM, the ISAM interface program, Access Method Services, or RPS support must be executed in virtual mode.

Programs that Must Run in Real Mode

The IBM-supplied programs OLTEP and the QTAM message control and message processing programs must be executed in real mode.

User-written programs must be executed in real mode if they contain channel programs for devices not supported by DOS/VS.

User-written programs must be executed in real mode or modified if they

- contain channel programs that are modified during command execution.
- contain I/O appendage routines causing page faults.
- contain MICR stacker selection routines or other time-dependent code for execution of I/O requests.

Summary of Job Control Statements and Commands

The following summarizes the job control statements and commands discussed in this section in relation to program execution.

EXEC

The EXEC statement indicates that the end of control information for a job step has been reached, and that execution of a program is to start. It is the last job control statement processed before a job step is executed.

If the program to be executed has just been processed by the linkage editor, the program name operand of the EXEC statement is blank.

To execute a program that is permanently cataloged in the core image library, the EXEC statement must specify the name of the first or only phase of that program..

All programs invoked through an EXEC statement are executed in virtual mode unless the operand REAL is specified. The SIZE parameter of the EXEC job control statement defines the low-end portion of the

partition which will be used during the job step. When the REAL operand is used, SIZE should also be specified to release the remainder of the partition to the page pool. SIZE must be specified for virtual mode programs that require the use of the GETVIS macro to obtain additional virtual storage during execution.

In response to an EXEC statement with the REAL operand, job control clears storage from the beginning to the end of the partition, a FETCH is issued for the desired program, and control is given to its entry point. When both REAL and SIZE are specified in the EXEC statement, only the portion of the real partition defined by SIZE is cleared.

(During execution of a virtual-mode program, the page management routine of the supervisor clears a page frame to zero if no page-in occurs when this page frame is assigned to the program.)

OPTION — The OPTION statement can be used to specify certain functions (options) to be performed by the system when a program is executed. Most of these functions pertain to the execution of the language processors.

A standard set of options can be established during system generation by the STDJC macro. If these standard options satisfy the requirements of your job, an OPTION statement is not needed. Exceptions are the options LINK, CATAL, PARSTD, and STDLABEL, which cannot be standard and must, if desired, be specified in an OPTION statement.

RSTRT — The RSTRT statement is used to restart a program from a checkpoint.

UPSI — The UPSI (user program switch indicator) statement can be used to set program switches in the communications region that can be tested by the problem program. The switches (UPSI byte) are reset to zero by a JOB or / & statement.

Checking and Altering Job Control Statements

It is often desirable to exercise a certain measure of control on the initiation of a job step. To this end a facility is provided which enables you to keep a running check on how a job step is executed, thereby enhancing security, serviceability, and reliability. After a job control statement has been read, control can be passed to a user exit routine for the purpose of examining and altering the statement prior to its being processed by the system.

The DOS/VS distribution volume contains a dummy phase $JOBEXIT in the system core image library. If you do not use the Job-control-exit facility, it has no effect on your system. For more information on the conventions for writing such a job control exit routine, together with an example, refer to *Writing a Job Control User Exit Routine* in *Chapter 10: Using the Facilities and Options of the Supervisor.*

System Files on Tape, Disk or Diskette

In the section *Symbolic I/O Assignment,* earlier in this chapter, it was stated that a physical I/O device (except DASD) cannot be assigned to more than one active partition at the same time. This means, for instance, that in an installation with only one card reader the input job stream on SYSRDR and SYSIPT for one partition must have been completely processed by job control and unassigned for that partition before job streams can be read by another partition. This also applies to the system output on SYSLST and SYSPCH if only one printer and one card punch are available.

Since this situation can cause a considerable decrease of system throughput, DOS/VS permits storing the input job streams and the system output on a direct access device or, if enough tape units are available, on magnetic tape. This allows several partitions simultaneously to read system input from or to write system output to high-speed devices, thus increasing system throughput and, due to reduced CPU wait time, improving the overall performance.

Note: *If system logical units (SYSIPT, SYSLST, SYSPCH, SYSRDR) are to be device independent, DTFDI must be used in the application program.*

The following section describes how to store system input and output on high-speed devices and to read and process the job streams from these devices.

The same improvements as those gained by having system files on high-speed devices - but far more efficient and easier to use - can be achieved by using an optional service program of DOS/VS: POWER/VS. POWER/VS stores the job streams on disk, transfers the jobs to the partitions for execution, and stores list and punch output on disk before it is finally printed or punched. In short, everything described in this section is done automatically by POWER/VS. Thus, if your installation works with POWER/VS, refer to *DOS/VS POWER/VS Installation Guide and Reference.*

System Files on Tape

If the system input units SYSRDR and SYSIPT are assigned to the same magnetic tape unit, they may (but need not) be referred to as SYSIN. If the system output units SYSLST and SYSPCH are assigned to the same magnetic tape they must be referred to as SYSOUT. The tapes may be unlabeled or they may have standard labels. SYSIPT assigned to a magnetic tape cannot be a multi-volume file.

To store the input stream on magnetic tape you must write your own program that transfers the job stream to the tape. Assume, in the following example, that you have written such a program and cataloged it in the core image library under the name CDTOTP; the program CDTOTP uses SYS004 to read the input job stream, and SYS005 for the tape onto which the job stream is to be written; the end of input data for CDTOTP is indicated by **. The example in Figure 5.12 shows how to use the program

CDTOTP to create a combined system input file on tape.

```
   // JOB BUILDIN
1  // ASSGN SYS004,X'00C'
2  // ASSGN SYS005,X'182'
3  // EXEC CDTOTP
   // JOB A
       .
       .
   /&
   // JOB B      job stream
       .
       .
   /&
4  **
   /&
```

1	SYS004 is assigned to the card reader from which CDTOTP reads the job stream.
2	SYS005 is assigned to the tape which is to receive the job stream.
3	The CDTOTP program is executed and writes the job stream onto tape.
4	** (or any other significant character combination) signals end-of-data to CDTOTP

Figure 5.12. Creation of SYSIN on Tape

After completion of the job BUILDIN shown in Figure 5.12 you can assign SYSIN to the tape containing the job stream; job control will then read and process the jobs A and B from the tape just as it would have done from the card reader.

In the same way you can direct the system output on SYSLST and SYSPCH to go on magnetic tape and then use your own or an IBM-supplied program to print or punch the contents of the tape on the printer or card punch, respectively.

System Files on Disk

System files on disk can be used only if the SYSFIL parameter was specified in the FOPT macro during supervisor generation. Systems without tape units should specify the SYSFIL parameter to facilitate system maintenance.

When both SYSRDR and SYSIPT are assigned to disk, they *must* refer to the same disk extent, and be referred to as SYSIN. Since the output units SYSLST and SYSPCH have different record lengths, they must be assigned to separate disk extents; SYSOUT can therefore *not* be used if SYSLST and SYSPCH are assigned to disk.

For system files on disk, you must provide the required label information by means of DLBL and EXTENT job control statements. Note that only single extent system files are supported. You must use the following predefined filenames when reading system input from disk or writing system output on disk:

IJSYSIN for SYSRDR, SYSIPT, SYSIN
IJSYSPH for SYSPCH
IJSYSLS for SYSLST

For example, the label information for SYSIN assigned to a disk extent could be submitted by the following job control statements:

```
// DLBL IJSYSIN,'DISKINFILE'
// EXTENT SYSIN,DOSRES,1,0,1260,30
```

The assignment of a system file to a disk extent must always be permanent (no //), and it must follow the DLBL and EXTENT statement. Example:

```
// DLBL IJSYSIN,'DISKINFILE'
// EXTENT SYSIN,DOSRES,1,0,1260,30
   ASSGN SYSIN,X'131'
```

After a system file on disk has been processed, it must be closed by a CLOSE job control command (no //). The second (optional) operand of the CLOSE command can be used to unassign a system logical unit or reassign it to another device. The following command closes the SYSIN file on disk and reassigns SYSIN to the card reader:

```
CLOSE SYSIN,X'00C'
```

The CLOSE command can either be entered on SYSLOG by the operator or it can be included at the end of the job stream on disk.

If SYSIPT is assigned to a disk extent, the CLOSE command must precede the / & . Multiple SYSIPT data files can be read via multiple job steps with one / & at the end of the job stream.

The example in Figure 5.13 shows the job control statements needed to

1. write a job stream on disk,
2. execute the job stream from disk and store the print output on disk, and
3. print the output from disk on the printer.

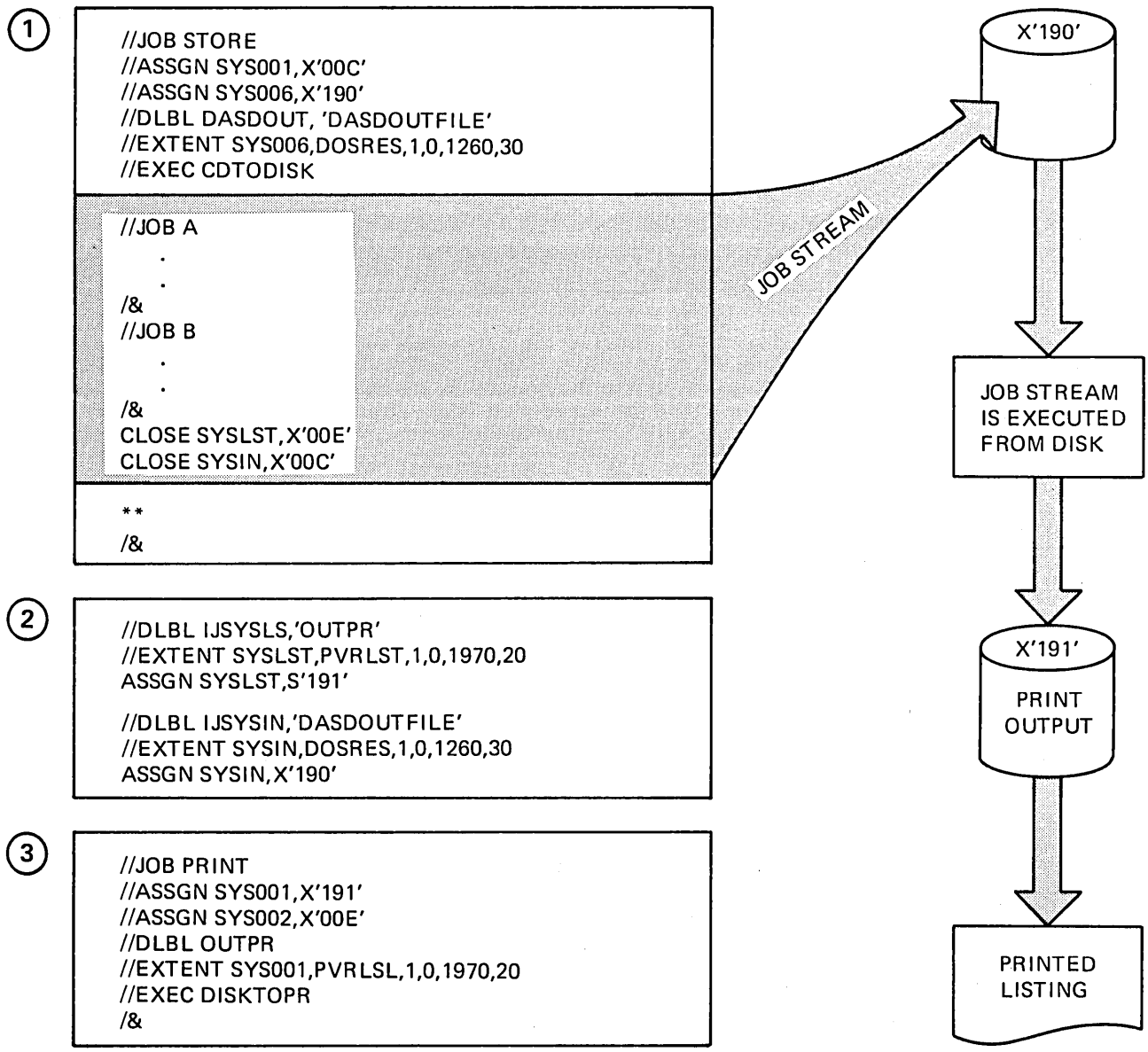
The example assumes that you have written your own programs to write the job stream on disk (CDTODISK) and to list on the printer the print output stored on disk (DISKTOPR).

System Files on Diskette

System files on diskette can be used only if the SYSFIL parameter was specified in the FOPT macro during supervisor generation.

If the system input units SYSRDR and SYSIPT are assigned to the same diskette extent, they *must* be referred to as SYSIN. Since the output units SYSLST and SYSPCH have different record lengths, they must be assigned to separate diskette extents; SYSOUT can therefore *not* be used if SYSLST and SYSPCH are assigned to diskette.

For system files on diskette, you must provide the required label information by means of DLBL and EXTENT job control statements. You must use the following predefined filenames when reading system input from diskettes or writing system output on diskettes.



1. The program CDTODISK reads the following job stream from the card reader (SYS001) and stores it on disk (SYS006). The end of the job stream is indicated to CDTODISK by **.

2. SYSLST and SYSIN are switched to disk. Job control now reads the job stream from the disk on device X'190'. The job stream is executed and the print output is stored on the disk on divice X'191'. The CLOSE commands at the end of the job stream will close the system files on disk and reassign them to the printer and card reader, resp.

3. The program DISKTOPR reads the print output from disk (SYS001) and lists it on the printer (SYS002).

Figure 5.13. Processing System Input and Output Files on Disk

IJSYSIN FOR SYSRDR, SYSIPT, SYSIN
IJSYSPH for SYSPCH
IJSYSLS for SYSLST

For example, the label information for SYSIN assigned to a diskette extent could be submitted by the following job control statements:

```
// DLBL IJSYSIN,'DISKETTE',,DU
// EXTENT SYSIN,DSKETE,1
```

The assignment of a system file to a diskette extent must always be permanent (no //), and it must follow the DLBL and EXTENT statement.

Example:

```
// DLBL IJSYSIN,'DISKETTE',,DU
// EXTENT SYSIN,DSKETE,1
   ASSGN SYSIN,X'060'
```

After a system file on diskette has been processed, it must be closed by a CLOSE job control command (no //). The second (optional) operand of the CLOSE command can be used to unassign a system logical unit or reassign it to another device. The following command closes the SYSIN file on diskette and reassigns SYSIN to the card reader:

```
CLOSE SYSIN,X'00C'
```

The CLOSE command can either be entered on SYSLOG by the operator or it can be included at the end of the job stream on diskette.

If SYSIPT is assigned to a 3540 diskette, the CLOSE command must be issued prior to reading the / & . Multiple input data files can be read via multiple job steps with one / & at the end of the job stream.

When job control encounters / & on SYSRDR during normal operation, the standard assignment for SYSIPT becomes effective and SYSIPT is checked for an end-of-file condition. If the standard assignments for SYSRDR and SYSIPT are not to the same device, SYSIPT is advanced to the next / & statement.

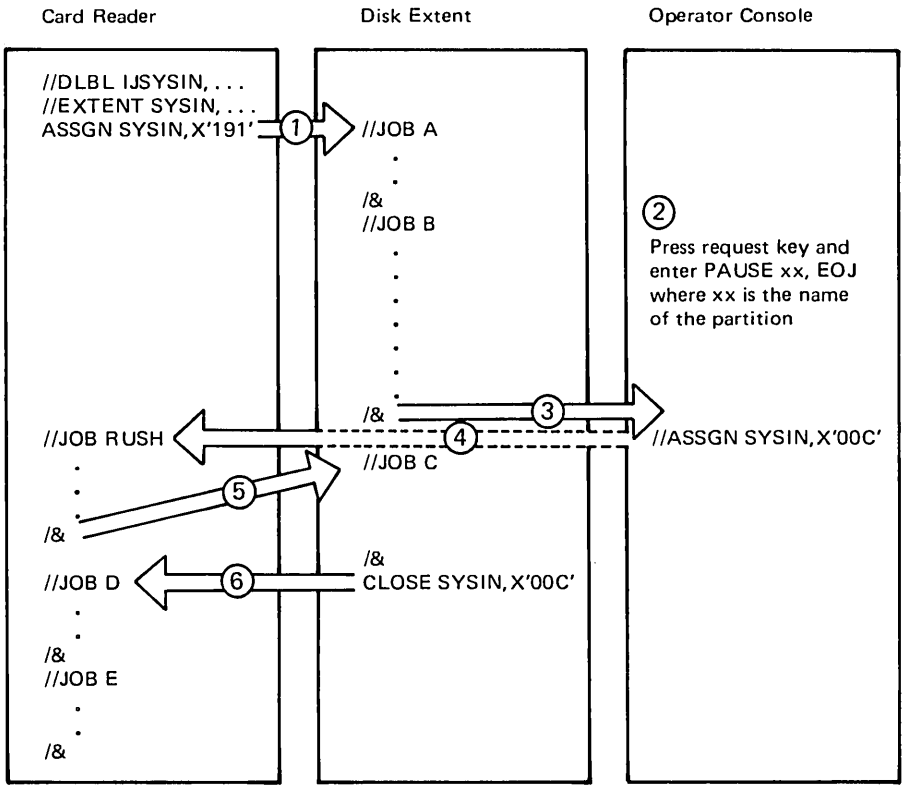
In the event of an abnormal termination, job control advances SYSRDR and SYSIPT to the next / & and proceeds, only if a JOB statement is provided.

Interrupting Job Streams on Disk, Diskette, or Tape

After a SYSIN or SYSRDR job stream has been prepared on tape, diskette, or disk, it may be necessary to interrupt the normal schedule to execute a special rush job. To do this, you press the request key on the operator console and enter a PAUSE command with the EOJ operand causing the corresponding partition to suspend processing at the end of the current job. At this point you can make a temporary assignment for SYSIN to the card reader to execute the rush job. At the end of this job, processing of the job stream on disk, diskette, or tape will resume at the point of interruption. This is illustrated in Figure 5.14. Starting an urgent job that uses a catalog procedure by means of a single EXEC statement is discussed in the section *Partition-Related Cataloged Procedures.*

Record Formats of System Files

SYSLST records are 121 characters and SYSPCH records 81 characters in length. From SYSRDR and SYSIPT, job control accepts either 80- or 81-character records. (For none of these files can the records be blocked.) You can use object modules written on tape, diskette, or disk as input to the linkage editor after the tape, diskette, or disk has been assigned to SYSIPT.



1. SYSIN is assigned to disk and processing of the jobstream on disk begins.
2. While job B is being executed a PAUSE command is entered at the operator console.
3. At the end of job B control comes to the operator who can now enter a temporary assignment for SYSIN to the card reader.
4. The job RUSH is read and processed from the card reader. Note that the temporary assignment of SYSIN is not reset by the //JOB RUSH statement but is retained to end of the job.
5. The /& statement resets the temporary assignment of SYSIN to permanent (X'190') and the next job in the stream on disk is read and executed.
6. The CLOSE command closes the system file on disk and reassigns SYSIN to the card reader to process jobs D and E.

Figure 5.14. Interrupting a Job Stream on Disk

The first character of the SYSLST and SYSPCH records is assumed to be an ASA carriage control or stacker selection character. SYSIPT, SYSRDR, SYSPCH, and SYSLST records assigned to DASD have no keys, and record lengths are the same as stated above.

Using Cataloged Procedures

This section describes how to retrieve a cataloged procedure from the procedure library and how to modify the contents of a cataloged procedure. How a procedure is cataloged in the procedure library is discussed in *Chapter 7: Using the Libraries.*

Note: *The procedure library should not be updated in a running multiprogramming system.*

Retrieving Cataloged Procedures

To retrieve a cataloged procedure from the procedure library you use the PROC parameter in the EXEC job control statement specifying the name of the cataloged procedure. Assume that a certain program called PAYROLL uses the following job control statements (in addition to the // JOB and / & statements):

```
// ASSGN SYS017,READER
// ASSGN SYS018,PUNCH
// ASSGN SYS019,X'00E'
// ASSGN SYS020,TAPE
// ASSGN SYS021,DISK,VOL=111111
// TLBL TAPFLE,'FILE-IN'
// DLBL DSKFLE,'FILE-OUT',99/365,SD
// EXTENT SYS021,111111,1,0,200,400
// EXEC PAYROLL
```

Assume further that these control statements have been cataloged in the procedure library under the name PAY. If the program PAYROLL is to be executed, the programmer or operator would simply prepare the following job control statements:

```
// JOB USER1
// EXEC PROC=PAY
/&
```

When the job control program starts reading the job control statements in the input stream on SYSRDR and finds the EXEC statement, it knows by the operand PROC that a cataloged procedure is to be inserted. It takes the name of the procedure to be used (PAY), retrieves the procedure with that name from the procedure library, and replaces the EXEC statement in the input stream by the retrieved procedure. The individual statements that are inserted are then processed from the very beginning. The statement

```
// EXEC PAYROLL
```

causes the program PAYROLL to be loaded and given control. When execution of PAYROLL is complete, the job control program finds the / & statement and performs end-of-job processing as usual.

Note: *The listing of job control statements on SYSLOG and/or SYSLST will show the message EOP PAY at the end of the inserted procedure.*

Modifying Cataloged Procedures

The preceding example is the simplest case of the use of cataloged procedures. It will work as long as the requirements of the program do not change.

It may happen, however, that some of the statements in a cataloged procedure must be modified for a specific run of a program. For example, the printer normally used (X'00E' in the preceding example) may be temporarily unavailable so that a different printer must be assigned. It does

not make much sense to delete the old version and to catalog the new one because the old version will be needed as soon as the normal printer becomes operational again.

Likewise, it may be necessary to add or remove certain statements to or from a cataloged procedure for a specific run of a program. You may wish, for example, to process a different copy of the file FILE-OUT (see the preceding example). You must therefore temporarily suppress the corresponding DLBL and EXTENT statements in the cataloged procedure and replace them by statements that identify the file you want to process instead.

For cases like this DOS/VS permits

- temporarily modifying one or more statements in a cataloged procedure (thus, overriding what was present).
- temporarily suppressing (deleting) one or more statements in a cataloged procedure without modifying them.
- temporarily incorporating one or more additional statements at desired locations in a cataloged procedure.

You can request temporary modification of statements in a cataloged procedure by supplying the corresponding modifier statements in the input stream. Normally, not all statements are to be modified.

You must therefore establish an exact correspondence between the statement to be modified and the modifier statement by giving them the same symbolic name. This symbolic name may have from one to seven characters, and must be specified in columns 73 through 79 of both statements.

Note: *An unnamed statement cannot be modified. To be able to modify a single statement in a cataloged procedure, you should name each statement when cataloging. Moreover, the modifier statements must be in the sequence in which modification is to be performed on the cataloged statements. The JOB and /& statements cannot be used as modifier statements.*

A single character in column 80 of the modifier statement specifies which function is to be performed:

A - indicates that the statement is to be inserted *after* the statement in the cataloged procedure that has the same name.

B - indicates that the statement is to be inserted *before* the statement in the cataloged procedure that has the same name.

D - indicates that the statement in the cataloged procedure that has the same name is to be *deleted.*

Any other character or a blank in column 80 of the modifier statement indicates that the statement is to replace (override) the statement in the cataloged procedure that has the same name.

In addition to naming the statements and indicating the function to be performed, you must inform the job control program that it has to carry out a procedure modification. This is done

(1) by specifying an additional parameter (OV for overriding) in the EXEC statement that calls the procedure, and

(2) by using the statement // OVEND to indicate the end of the modifier statements.

Placement of the // OVEND statement is as follows:

- Place directly behind the last modifier statement
- If the last modifier statement overwrites a // EXEC statement and is followed by data input, place the // OVEND between the /* and the /&.

The following examples show how you can temporarily modify a cataloged procedure.

Assume that a cataloged procedure named PROC5 for the program PAYROLL contains the following statements:

```
                                          73--79
// ASSGN SYS017,READER                    PAY0001
// ASSGN SYS018,PUNCH                     PAY0002
// ASSGN SYS019,PRINTER                   PAY0003
// ASSGN SYS020,X'181'                    PAY0004
// ASSGN SYS021,DISK,VOL=111111           PAY0005
// TLBL TAPFLE,'FILE-IN'                  PAY0006
// DLBL DSKFLE,'FILE-OUT'                 PAY0007
// EXTENT SYS021,111111,1,0,200,200       PAY0008
// EXEC PAYROLL                           PAY0009
```

Assume further that the programmer wants to use tape unit X'183' instead of X'181'. The input stream on SYSRDR, in this case, would have to be as follows:

```
                                          73--79
// JOB USER
// EXEC PROC=PROC5,OV
// ASSGN SYS020,X'183'                    PAY0004
// OVEND
/&
```

The form of the EXEC statement in the input stream indicated that (1) the procedure PROC5 is to be used and (2) this procedure is to be modified in some way. The first three procedure statements are processed without change. The procedure statement named PAY0004 is replaced by the corresponding statement in the input stream (a blank in column 80 specifies overriding). The remaining procedure statements are again processed without change.

As another example, assume that the program PAYROLL is to use the file FILE-OUT1 instead of FILE-OUT and that this file resides on two extents of a disk pack that has the volume serial number 111112. The input stream might then look as follows:

```
                                      Col.73--79 80
// JOB USER
// EXEC PROC=PROC5,OV
// DLBL DSKFLE,'FILE-OUT1'                PAY0007
// EXTENT SYS021,111112,1,0,100,200       PAY0008
// EXTENT SYS021,111112,1,1,500,200       PAY0008A
// OVEND
/&
```

Processing would be as follows: The JOB statement and all procedure statements up to the statement named PAY0006 are processed without modification. The procedure statements named PAY0007 and PAY0008 are replaced by the corresponding statements in the input stream (due to the blank in column 80). The second EXTENT statement in the input stream has the character A in column 80, which indicates that the statement is to be inserted after the (replaced) statement named PAY0008. The procedure statement named PAY0009 is again processed without modification.

The possibility of modification as described above makes the use of cataloged procedures more flexible. Often, however, it is simpler and more economical to have different procedures for the same program than to have a single procedure and modify it.

Several Job Steps in one Procedure

A cataloged procedure may contain more than one EXEC statement, that is, it may contain control statements for more than one job step (within the same job). Bear in mind that as the number of job steps in a procedure increases, so does the time required to re-execute the whole procedure after an error occurs. A program written in assembler language, for instance, requires three job steps to assemble, link-edit, and execute the program. For the use of a cataloged procedure, your input stream for the entire job (on SYSIN for simplicity) would contain the following:

```
// JOB USER
// OPTION LINK
// EXEC ASSEMBLY
source deck of program to be assembled
/*
// EXEC LNKEDT
// EXEC
data for program to be executed
/*
/&
```

If the OPTION statement and the three EXEC statements were cataloged under the name ASDPROC, the input stream could be simplified to the following (the shaded portions represent statements from the procedure library):

```
// JOB USER
// EXEC PROC=ASDPROC
// OPTION LINK
// EXEC ASSEMBLY
source deck of program to be assembled
/*
// EXEC LNKEDT
// EXEC
data
/*
/&
```

The same can be done for any number of job steps that logically belong together and are frequently executed. A stock control program STOCK, for instance, may be run daily to compile statistics that can be used to prepare the following lists:

1. An exception list that shows which items are low in stock. Required daily.

2. A list that shows the turnover in currency for a certain item or group of items. Required weekly.
3. A list that shows the turnover in number of units for each item or group of items. Required monthly.
4. An inventory list. Required semi-annually.

To simplify processing, four procedures may have been cataloged:

STKPR1 - two job steps: the first to execute STOCK, the second to prepare list 1.

STKPR2 - three job steps: the first two are the same as for STKPR1, the third to prepare list 2.

STKPR3 - four job steps: the first three the same as for STKPR2, the fourth to prepare list 3.

STKPR4 - five job steps: the first four the same as for STKPR3, the fifth to prepare list 4.

Which lists are printed after every run of STOCK then depends on what cataloged procedure is used.

Modifying Multistep Procedures without SYSIPT Data

Multistep procedures may be modified in the same way as the single-step procedure described earlier. A number of considerations, however, apply to the ordering of the modification statements in the input stream when a logical unit is assigned to the same physical unit as SYSRDR.

1. It is advisable to avoid using identical symbolic names for the statements in the procedure.
2. The modifier statements must be in the same sequence as the statements in the referenced procedure.
3. If one step of a procedure is unmodified, the first modifier statement for the following step must be placed either before the data input for the unmodified step or after the last modifier statement of the preceding job step. If it is the first modifier statement in the input stream, it must be placed immediately after the EXEC PROC statement.
4. If a modifier statement overwrites an EXEC statement, a subsequent modifier statement must be placed *after* the data input (and /*) for this step.

Figure 5.15 shows an example of modifying the second and third steps of a three-step procedure.

In the example given in Figure 5.1, it is assumed that SYSRDR and SYSIPT are assigned to the same physical unit. The following notes apply to the example:

1 This is the first modifier statement. It refers to step 2.

2 This statement provides SYSIPT data for PSERV.

3 This modification overwrites the EXEC statement.

4 This statement provides SYSIPT data for DSERV.

5 This statement provides SYSIPT data for DSERV.

	SYSIN Input Stream	Column 73–79	Procedure CAT01 Containing JCL Only	Column 73–79
	/ / JOB EXAMPLE			
	/ / EXEC PROC=CAT01,OV			
(1)	/ / ASSGN SYSRLB,UA	STMT3	/ / EXEC PSERV	STMT1
(2)	DSPLY CAT01			
	/ *		ASSGN SYSCLB,X'130'	STMT2
			/ / ASSGN SYSRLB,X'130'	STMT3
	/ / ASSGN SYSSLB,UA	STMT4	/ / ASSGN SYSSLB,X'130'	STMT4
(3)	/ / EXEC DSERV,REAL	STMT5	/ / EXEC DSERV	STMT5
(4)	DSPLY CD,RD,SD			
	/ *			
	ASSGN SYSCLB,UA	STMT6	/ / ASSGN SYSSLB,UA	STMT6
	/ / OVEND		/ / EXEC DSERV,REAL	STMT7
(5)	DSPLY CD,PD		/ +	
	/ *			
	/ &			

Figure 5.15. Example of Modifying a Three-Step Procedure

SYSIPT Data in Cataloged Procedures

Procedures may additionally contain SYSIPT inline data, such as control statements for system utility and service programs and source modules.

Note: *This extended support requires a supervisor that was generated with the SYSFIL option.*

SYSIPT inline data in procedures may also be any data that is processed under control of the device independent IOCS used by your program or IBM-supplied programs. Normally, though, you would not catalog source programs or data for your problem programs in the procedure library.

Including SYSIPT inline data in procedures is useful and convenient mainly in the case of control information for system utility and service programs.

A job stream for an initialize disk utility run could, for instance, contain the following control statements (the statements are shown in skeleton format only):

```
// ASSGN
// EXEC INTDK
// UID IR,C1,R=(0027003)
// VTOC STANDARD
VOL1111111
```

```
// END
/&
```

If SYSRDR and SYSIPT were not combined and no cataloged procedure was used, the job control statements would have to be placed on SYSRDR. whereas the utility control statements would have to be placed on SYSIPT. If, however, these control statements had been cataloged (for example, under the name INITDK), only the following statements would be required on SYSRDR:

```
// JOB NAME
// EXEC PROC=INITDK
/&
```

SYSIPT data can either be read from SYSIPT or be retrieved from the procedure library. Combining the two possibilities in a (single-step or multi-step) procedure is not permitted. Also, SYSIPT data read from the procedure library cannot be modified. In a cataloged procedure with in-line SYSIPT data, you should not delete or overwrite an EXEC statement that gives control to a program that uses the SYSIPT data.

For multistep procedures, SYSIPT data must be read in all job steps either from SYSIPT or from the procedure library. If the SYSIPT data for the first job step is read from SYSIPT, having SYSIPT data for any of the following job steps in the procedure would lead to an error. Conversely, if the SYSIPT data for the first job step is contained in the procedure, any SYSIPT data for subsequent job steps must also be contained in the procedure.

Partition-Related Cataloged Procedures

In some instances, a particular cataloged job may need a specific set of job control statements, dependent on the partition of execution. For example, you may want to run a job to store DLBL and EXTENT statements onto the partition label track for each partition (OPTION PARSTD). Since each partition requires a different set of label information, you would need a cataloged procedure for each partition.

Partition-related cataloged procedures, then allow you to retrieve and execute the appropriate procedure with a single EXEC statement. One benefit of this feature lies in the ease with which unscheduled jobs can be started. At execution time, the system selects the proper procedure--including the appropriate EXTENT and DLBL statements--based on the partition in which the job is to be executed.

To use the feature, you must first create separate sets of job control statements that conform to the specific partitions in your system. Most probably, the difference in these sets will be in the EXTENT and DLBL statements, because of the different device and DASD space assignments from partition to partition. Second, in order to distinguish between the procedures and relate them to the appropriate partitions, the following naming convention must be used for procedures to be placed in the library:

First character of name	-	$
Second character	-	B for BG partition
	-	1 for F1 partition
	-	2 for F2 partition
		etc.
Third-eighth characters	-	any alphameric characters

In the EXEC statement used to start the job, however, the first two characters of the procedure name must be $$, with the remaining characters identical to the cataloged name.

On reading the EXEC statement, the system replaces the second $ with the identifier for the partition in which the job is to run. The procedure with this name is then retrieved, read, and executed.

To continue the previous example, the procedures may be named $BPARSTD for the BG partition, $1PARSTD for the F1 partition and so on. The statement thus needed to invoke the appropriate procedure is // EXEC PROC=$$PARSTD.

Use of Cataloged Procedures by the Operator

All the previously described functions and advantages of cataloged procedures are also available to the operator. Of special importance in the operator's use of cataloged procedures is the starting of urgent jobs or long-running jobs like POWER/VS or teleprocessing.

Full details on the use of cataloged procedures by the operator are given in *DOS/VS Operating Procedures.*

Chapter 6: Linking Programs

Prior to execution in storage, all programs must be placed in the core image library by the linkage editor. This chapter describes the role of the linkage editor and how you can communicate with it through control statements.

The name *linkage editor* appropriately reflects the editing and the linking operations that this program performs. The linkage editor prepares a program for execution by editing the output of a language translator into core image format. The linkage editor also combines separately assembled or compiled program sections or subprograms (called *object modules*) into phases. This process is referred to as linking.

A program can be link-edited and

- cataloged permanently,
- cataloged permanently and executed immediately, or
- cataloged temporarily and executed immediately.

When a program is cataloged permanently into the core image library, the linkage editor is no longer required for that program*, because the supervisor can load it directly from the library in response to an EXEC job control statement, or a FETCH or LOAD macro. On the other hand, if the program is cataloged temporarily and executed immediately, the linkage editor is required again the next time the program is to be run.

If a private core image library is assigned to the partition in which the execution of the linkage editor occurs, the phases produced are entered into this private core image library. Otherwise (for the background partition), the phases are entered into the system core image library. To execute the linkage editor in a foreground partition, a private core image library must be uniquely assigned to that partition. For more information on using private libraries, refer to *Chapter 7: Using the Libraries.*

Structure of a Program

To understand the functions of the linkage editor, you must understand the structure of a program during the various stages of its development. Figure 6.1 summarizes the three sections that follow, which discuss source modules, object modules, and program phases.

---

*If the partition boundaries change so that the cataloged program's START and END addresses no longer fall within the partition, the program must be link-edited again. This restriction does not apply to relocatable programs loaded by the relocating loader.

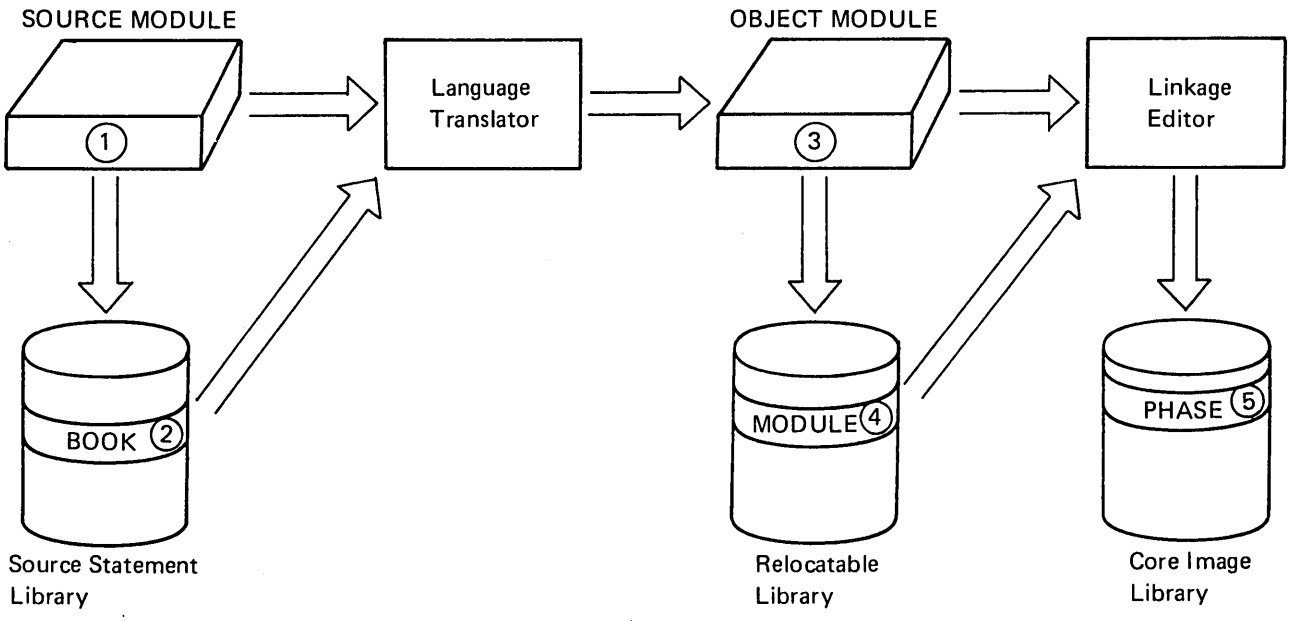


Figure 6.1. Stages of Program Development

A set of source statements, or source module (1), must be processed by a language translator, but can first be cataloged as a book (2) into the source statement library. The output of the language translator is called an object module (3), which must be processed by the linkage editor, but can first be cataloged as a module (4) into the relocatable library. The output of the linkage editor is called a phase (5), which is cataloged into the core image library temporarily or permanently, and can also be loaded into the shared virtual area. (A phase is cataloged temporarily if // OPTION LINK is specified; a phase is cataloged permanently if // OPTION CATAL is specified.) At execution time, either the system loader loads a phase from the core image library into the problem program partition, or (if appicable) the partition requesting the phase uses the copy available in the shared virtual area.

Source Modules

After planning the most logical approach to the problem you are to submit to the computer, you write a set of source statements in a programming language. Your set of source statements, called a source module, must be processed by a language translator. The language translator *assembles* source modules written in assembler language, or it *compiles* source modules written in a high-level language (for instance, ANS COBOL, FORTRAN, PL/I, or RPG II). The language translator transforms the source module into an object module, which is in machine language.

You can either submit your source module directly to the language translator for processing, or you can catalog it into a sublibrary of the source statement library for processing at a later time by the language translator. (Refer to *Chapter 7: Using the Libraries* for guidelines on how to catalog into the source statement library.)

A source module written in assembler consists of definitions for one or more control sections (CSECTs). Source modules written in a high-level language do not have this structure.

Object Modules

An object module, the output of a language translator, consists of the dictionaries and text of one or more control sections. The dictionaries contain the information necessary for the linkage editor to modify portions of the text for relocation and to resolve cross-references between different object modules. The text consists of the actual instructions and data fields of the object module.

You can either submit your object module directly to the linkage editor for processing, or catalog it into the relocatable library for later inclusion in a linkage editor job stream. (Refer to *Chapter 7: Using the Libraries* for guidelines on how to catalog into the relocatable library.)

The language translator produces four types of cards for each object module. An identifier field in columns 2-4 indicates the content of each card. Column 1 contains a multipunch (12-2-9) that identifies the card as being part of an object module (also referred to as a loader card). The four types of cards are: ESD, TXT, RLD, and END. The contents of these cards are summarized below.

ESD (External Symbol Dictionary). This card contains all the symbols defined in this module that are referred to by another module and all the symbols referred to by this module that are defined in another module. There are six classifications of the ESD card, which are described in *DOS/VS System Control Statements.*

TXT (Text). This card consists of the actual code of the object module. It contains the assembled (or compiled) address of the instructions or data included in the card, and the number of bytes contained in the card. It also includes a reference to the control section where this text occurs. The linkage editor uses this reference when applying a relocation factor. If address constants are present, TXT information is modified as required by RLD information.

RLD (Relocation List Dictionary). The RLD cards identify portions of the text that must be modified if the program is subsequently relocated. They provide information necessary to perform the relocation.

END (End of Module). The END card indicates the end of the module to the linkage editor. The END card may supply a transfer address (where execution is to begin). It may also contain the control section length, which was not previously specified in the ESD section definition or private code (unnamed CSECT).

If you want to change information in a TXT card, you can prepare a REP card (user replace card) and submit it with your object module for cataloging into the relocatable library or for linkage editor processing. A REP card must be submitted between the TXT card it modifies and the END card; otherwise, the TXT card is not modified. Usually, you place the REP card(s) immediately before the END card.

For the exact formats of the ESD, TXT, RLD, REP, and END cards, refer to *DOS/VS System Control Statements.*

Program Phases

The linkage editor produces a program phase from the object module(s) you identify in linkage editor control statements. A phase is the smallest functional unit (one or more control sections) that the system loader can load into a partition in response to a single EXEC job control statement or a FETCH or a LOAD macro instruction.

In the PHASE control statement you can instruct the linkage editor to produce one of three types of phases: relocatable, self-relocating, or non-relocatable.

Relocatable Phases

The linkage editor can produce a relocatable phase for those phases with an origin that is not an absolute address and that is not relative to a non-relocatable phase. If the supervisor was generated to support the relocating loader, a relocatable phase can be loaded into any partition for execution.

For each relocatable phase the linkage editor prepares special relocation information that the relocating loader uses to modify the text if necessary. Relocation is not performed if the program is to be executed at the same address for which it was link-edited.

For more information on relocatable phases, refer to the section *Link-editing for Execution at Any Address.*

If a relocatable phase is also designed as a reenterable phase, it is eligible to be loaded into the shared virtual area (SVA). Phases resident in the SVA can be shared concurrently by programs running in either real or virtual mode.

Self-Relocating Phases

Prior to the availability of the relocating loader in DOS/VS, users had to write self-relocating programs in order to gain the advantages of relocatability. If you have to perform maintenance on such a program, you must write this program in assembler language according to the rules described in *DOS/VS Supervisor and I/O Macros.* In the PHASE control statement you indicate an origin of +0. The program must relocate all its addresses at execution time to correspond with the addresses available in the partition where the program is loaded.

You do not need to write a self-relocating program if your supervisor includes support for the relocating loader. (Refer to *Relocatable Phases* above.)

Non-Relocatable Phases

A non-relocatable phase is link-edited to be loaded at a specific location (absolute address) in a partition. Phases link-edited without relocating loader support are also non-relocatable. When you request execution of a

non-relocatable phase in a given partition, the starting and ending addresses of the phase must be included within that partition. Otherwise, the job is canceled. If you wish to execute a non-relocatable phase in more than one partition, you must catalog a separate copy of the phase for each partition.

The Three Basic Applications of the Linkage Editor

The three basic applications of the linkage editor are referred to as:

1. cataloging phases into the core image library
2. link-edit and execute
3. assemble (or compile), link-edit, and execute.

The following sections include a discussion of the system flow during each of these applications.

Cataloging Phases into the Core Image Library

When you have an operational program that you expect to use frequently, you should catalog it into a core image library. You can do this in a single job step, which is shown in Figure 6.2, and described below.

When job control reads the CATAL operand of the OPTION statement, it sets a switch that causes the linkage editor input file, SYSLNK, to be opened. Job control copies onto SYSLNK the linkage editor control statements present on SYSRDR, and the INCLUDE statement signals job control to read any object modules that are to be included from SYSIPT. If an ENTRY statement is not encountered before the // EXEC LNKEDT statement, job control includes one on SYSLNK. This signals termination of the input to the linkage editor.

The linkage editor is then loaded into the partition where the job stream was submitted, and uses SYS001 as a work file to process the input found on SYSLNK.

Because the CATAL option was specified, the linkage editor places the executable program permanently into a core image library. If a private core image library is assigned to this partition, the program is cataloged there; otherwise, (for the background partition) it is cataloged into the system core image library. The library descriptor entry in the core image directory for cataloged phases is updated.

If the phase is eligible for the shared virtual area and is indicated as SVA eligible in the system directory list, the phase is also loaded into the SVA.

Note: *System and work files such as SYSLNK and SYS001 must be defined.*

Cataloging a Supervisor. Supervisors may also be cataloged permanently into the core image library as described above. Be sure, when doing this, to specify a unique name (eight alphameric characters) for each supervisor. Because the name of the supervisor must reside on the first cylinder of the

core image directory, give the name a low collating sequence (for example, use $$ as the first two characters).

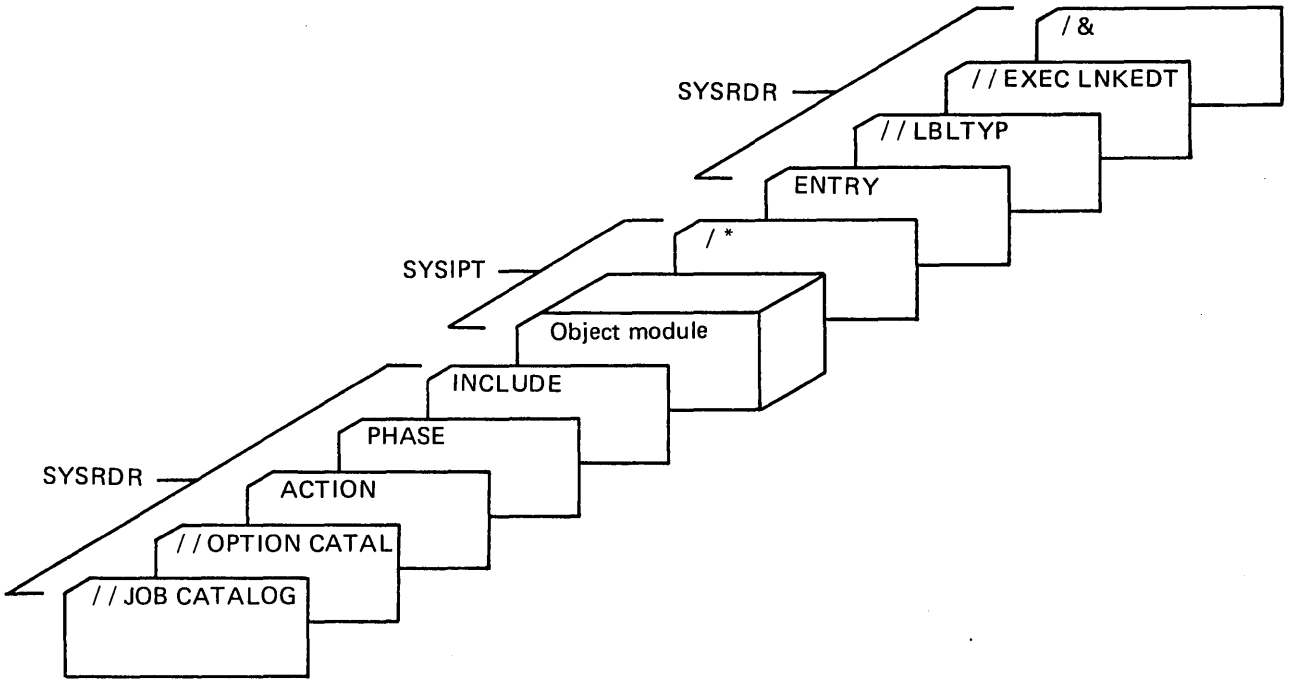


Figure 6.2. A Job Stream to Catalog a Program into the Core Image Library

The input to the linkage editor may consist of the linkage editor control statements ACTION, PHASE, INCLUDE, and ENTRY submitted on SYSRDR and object modules on SYSIPT.

Link-Edit and Execute

You do not always need to catalog a permanent copy of your program into the core image library. For instance, you have modified parts of your program and want to test these modifications with the entire program. In this case, you can specify the LINK option, which requests that the linkage editor place a temporary copy of the program into the core image library. The INCLUDE statement signals job control to read the following input from SYSIPT.

By specifying an EXEC statement without a program name operand after the EXEC LNKEDT statement, the program just link-edited is loaded for execution. The space temporarily occupied by this program in the core image library is overwritten the next time a program is link-edited.

The shaded portions of Figure 6.3 illustrate how this job stream differs from Figure 6.2.

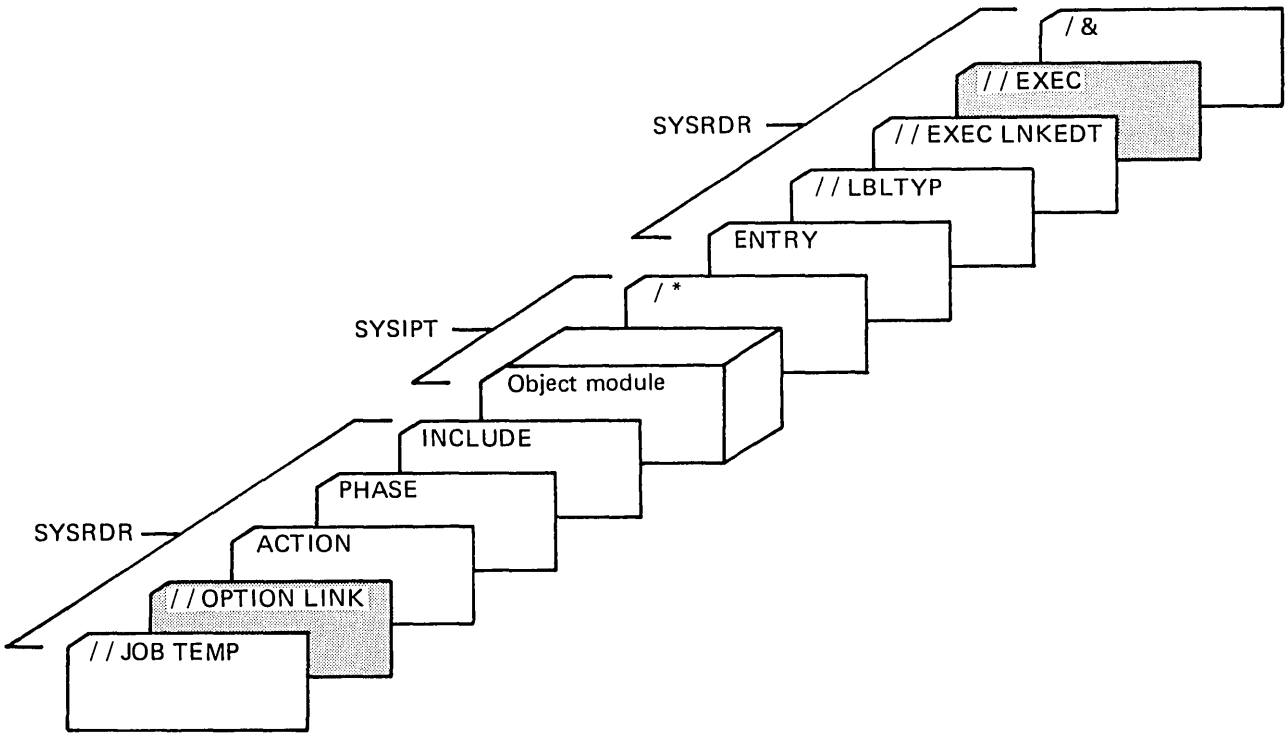


Figure 6.3. A Job Stream to Link-edit a Program for Immediate Execution

The // OPTION LINK card causes the linkage editor to place a temporary copy of the program into the core image library. INCLUDE (with NO operand) signals job control to read the program from SYSIPT. The // EXEC card (without a program name operand) causes this same program to be loaded for execution immediately thereafter.

The // OPTION CATAL card may also be used in this job stream. In this case, the program that was cataloged (permanently) is executed immediately.

Assemble (or Compile), Link-Edit, and Execute

You can also combine the job steps described above with a job step for assembly (or compilation) of your source program. This is especially useful when you are developing a program. Figure 6.4 shows how your job stream should be set up. The shaded portions of the figure illustrate how this job stream differs from that shown in Figure 6.3. Linkage editor control statements are not required when linking single-phase programs temporarily into the core image library.

You direct the language translator to write the object module directly onto SYSLNK by specifying the LINK option at the beginning of the job. After the linkage editor processes the input on SYSLNK, the same program is loaded for execution.

If errors occur in one job step causing an abnormal termination, the remaining job steps are ignored. Other types of errors that do not cause termination of a job step remain throughout the entire job. If you do not want to execute the program when errors occur during the link-edit step, you can specify ACTION CANCEL after the // OPTION LINK.

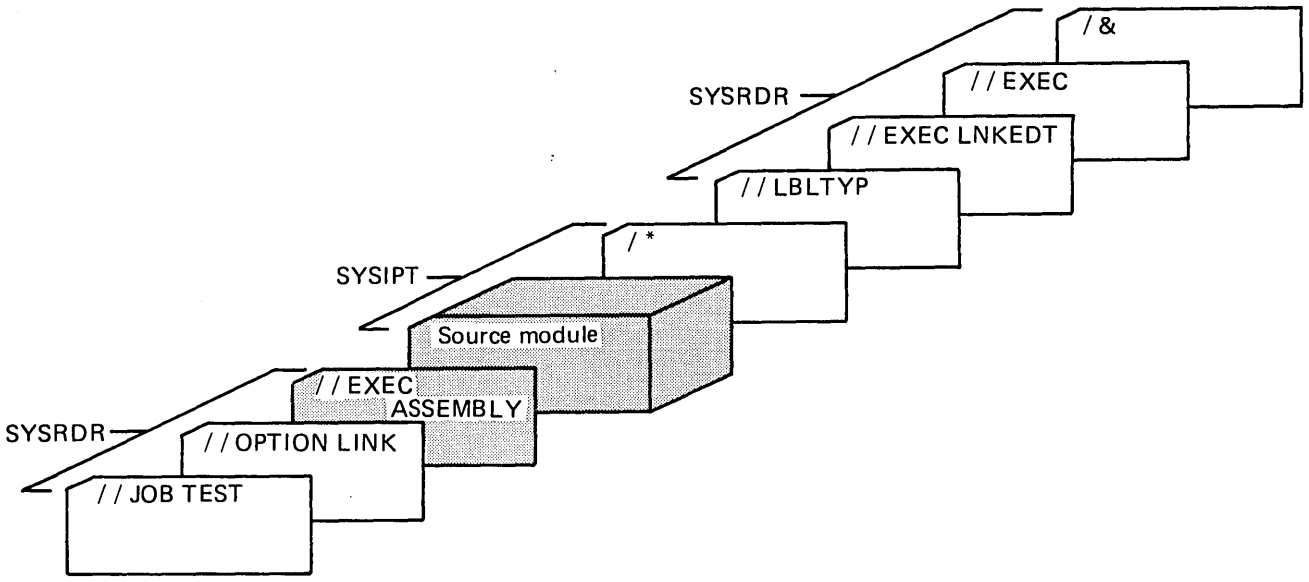


Figure 6.4. A Job Stream to Assemble, Link-edit, and Execute

You can omit linkage editor control statements when you specify // OPTION LINK. If you specify // OPTION CATAL, you must supply at least one PHASE card with a phase name before // EXEC ASSEMBLY.

Processing Requirements

In a system without private core image library support, the linkage editor can be executed in the background partition only and places phases into the system core image library on SYSRES. In a multiple-partition system where the supervisor supports private core image libraries, the linkage editor can be executed in any partition. When the linkage editor is executed in a foreground partition, a private core image library (SYSCLB) must be uniquely assigned to that partition and phases are placed there. When the linkage editor is executed in the background partition where no private core image library is assigned, phases are placed into the system core image library by default.

The size of the partition in which the linkage editor is operating directly influences the number of phases and ESD items that can be processed in one job step. By referring to the specific formulas listed in *DOS/VS System Control Statements,* you can determine if a particular combination can be processed within a given partition.

Symbolic Units Required

The linkage editor requires the following symbolic units:

SYSIPT Module input

SYSLST Programmer messages and listings (if SYSLST is not assigned no map is printed and programmer messages appear on SYSLOG)

SYSLOG Operator messages

SYSRDR Control statement input (via job control)

SYSLNK Input to the linkage editor

SYS001 Work file.

Note that SYSRDR and SYSIPT may contain input for the linkage editor. This input is written on SYSLNK by job control.

If output from the linkage editor is to be placed in a private core image library, the following symbolic unit is required:

SYSCLB The private core image library may be assigned anywhere in the job stream but must be before the // EXEC LNKEDT statement.

If object modules from a private relocatable library are to be link-edited, the symbolic unit SYSRLB must be assigned.

Preparing Input for the Linkage Editor

The input you prepare for the linkage editor consists of job control statements, linkage editor control statements, and object modules. Job control reads the job control statements and the linkage editor control statements from the device assigned to SYSRDR and object modules from SYSIPT. The linkage editor control statements and object modules are copied onto the disk extent assigned to SYSLNK.

The linkage editor control statements direct the execution of the linkage editor. The four linkage editor control statements are: ACTION, ENTRY, INCLUDE, and PHASE. Position 1 must be blank on linkage editor control statements; no // is used. In all other respects their format is the same as that for job control statements.

The job control statements that directly influence the linkage editor are: OPTION CATAL, OPTION LINK, and LBLTYP.

A description of how to prepare these control statements is given on the following pages. Here, the various operands of the control statements are described under headings that indicate their function. In the section *Summary of Control Statements Related to Link-editing,* these operands are briefly described again under the control statements to which they pertain.

Assigning a Name to a Program Phase

Each program phase you submit for link-editing should have a name, which you specify in the PHASE statement. When a phase is cataloged in the core image library, the phase name identifies that phase for subsequent retrieval. In other words, the same phase name you supplied in the PHASE statement when permanently cataloging the initial or only phase of a

program must be used as the operand in the EXEC job control statement or in a FETCH or a LOAD macro instruction.

The first four characters of the phase name of a single-phase program should be unique. Any phases with the same first four characters of their phase name will be classified as a multiphase program. When a phase of a multiphase program is fetched, the partition must be large enough to contain the largest phase.

You are not required to supply a phase name if you have specified the LINK option. The linkage editor will construct a dummy phase name (PHASE***) and your program can still be executed. A program with a dummy phase name cannot be permanently cataloged into a core image library; that is, you must supply a phase name in the PHASE statement when you specify the CATAL option. If the CATAL option is specified and no phase card is supplied before the first object module (or the phase card is invalid), a dummy phase card is created (phase name PHASE***). The link-edit job is canceled after a map has been printed (provided SYSLST is assigned and ACTION NOMAP was not in effect).

Defining a Load Address for a Phase

At link-edit time you can specify where your program is to be loaded for execution. You have several choices.

A phase can be link-edited to be loaded and executed from:

- a virtual partition
- a real partition
- the shared virtual area
- an absolute address (either in a virtual or a real partition).

A phase can be link-edited as a

- relocatable phase
- self-relocating phase
- non-relocatable phase.

You define the load address for a phase in the origin operand of the PHASE statement. (The load address can be changed by the system at execution time if the link-edited phase is relocatable and the relocating loader is supported in the supervisor. This is described in the sections that follow.) You can specify the origin in six different formats:

1. symbol [(phase)][± relocation]	Specifies a load address relative to the beginning of a virtual partition or to another phase.
2. * [± relocation]	
3. S [+ relocation]	
4. ROOT	
5. +displacement	Specifies an absolute address.
6. F+address.	

These specifications are described in *DOS/VS System Control Statements.*

Aligning a Phase on a Page Boundary

For performance reasons it can be advantageous to load a phase on a page boundary. If you specify the PBDY parameter in the PHASE statement, the linkage editor will align the load point of the phase on the nearest page boundary (the next higher).

Link-editing for Execution at Any Address

If you want to ensure that your program can be loaded at any storage address (except within the supervisor area), you can make use of the relocating loader.

Phases produced by the linkage editor for loading by the relocatable loader are called relocatable phases. If a relocatable phase is also reenterable it can be specified for inclusion in the shared virtual area. (See the section *Link-editing for Inclusion in the Shared Virtual Area*).

Using the Relocating Loader If your supervisor supports the relocating loader (refer to this supervisor generation option in the section *Tailoring the Supervisor* in *Chapter 3: Planning the System*), you do not need to write a self-relocating program to enable that program to execute in any real or virtual partition. The linkage editor will produce relocatable phases whenever possible. The linkage editor determines whether a phase can be made relocatable by inspecting the origin of the PHASE statement. If the origin specified is in one of the following formats, the phase is eligible for relocation:

- symbol[(phase)][± relocation]
- *[± relocation]
- S [+ relocation]
- ROOT

Note: *The first format specifies a symbolic load origin. If the phase referred to in a symbolic origin is not relocatable, the referring phase cannot be made relocatable. If a phase is relative to another phase whose origin is specified as an absolute address, none of the phases can be made relocatable during this linkage editor execution.*

If the linkage editor determines that a phase is to be given the the relocatable format, it flags the core image directory entry for that phase, prints a message (*relocatable*) after the phase information in the linkage editor map (see *Obtaining a Storage Map*), and inserts the relocation information behind the text of the phase in the core image library. This relocation information consists of a set of pointers to address constants, the length of these address constants, and an indication as to whether the supervisor should add or subtract a relocation factor when loading the phase into storage.

If your supervisor does not contain the relocating loader, the linkage editor can still produce a relocatable phase if you specify ACTION REL for a phase eligible for relocation. Such a supervisor, however, loads relocatable phases into storage as link-edited without performing any relocation.

If your supervisor contains the relocating loader and you do not want the linkage editor to produce a relocatable phase for a particular program, specify ACTION NOREL.

The default action taken depends on whether or not the supervisor contains the relocating loader. If it does, ACTION REL is the default; otherwise, ACTION NOREL is the default.

The REL operand and a partition-identifier operand (described in the section *Link-editing for Execution in a Virtual Partition*) are not mutually exclusive. For instance, if a program is normally to be executed in the virtual F1 partition, but not exclusively, specify ACTION F1,REL. Whenever this program is to be executed in the virtual F1 partition, relocation will not be necessary and the load time will be minimized.

Link-editing for Inclusion in the Shared Virtual Area

If a relocatable phase is also reenterable, it can be included in the shared virtual area (SVA). Phases resident in the SVA can be shared concurrently by more than one partition. It is advantageous to include frequently-used phases in the SVA because these are then resident when requested for execution (they are not reloaded from the core image library). All phases resident in the SVA must also be cataloged in the system core image library.

To indicate that a phase should reside in the SVA, you must specify the SVA operand in the PHASE statement when cataloging the phase. This operand is ignored if the phase is not relocatable (see above); otherwise, the SVA operand is accepted and the phase is said to be SVA-eligible.

The linkage editor cannot check whether a phase is reenterable; however, a protection check can occur when executing a phase from the SVA that is not reenterable, since the SVA is key zero storage. Since the SDL is sorted prior to the loading of phases into the SVA, the packaging of phases to be executed together should be done using the linkage editor.

Immediately after an SVA-eligible phase is cataloged into the system core image library, it is loaded into the SVA *if* this phase is listed as SVA-eligible in the system directory list (SDL). The SDL can be created only immediately after IPL; see the section *Building the SDL and Loading the SVA* in *Chapter 4: Starting the System.*

Link-editing for Execution in a Virtual Partition

Unless otherwise specified in the PHASE statement, a program is link-edited to execute in the same virtual partition in which the linkage editor function occurs. When the linkage editor is running in a real partition, the program is link-edited to execute in the corresponding virtual partition.

By using the ACTION statement with one of the partition identifiers (BG, F1, F2, etc.), however, you specify the virtual partition in which the program is to be executed. It is necessary to specify a partition identifier

only if the "run" partition differs from the partition in which the linkage editor is being executed.

Use of the ACTION statement with a foreground partition identifier requires that the virtual partition be allocated; if not, the action is ignored.

An ACTION statement with a partition identifier is effective only for those phases designated to be loaded at an address relative to the beginning of a partition: that is, for those phases with a load address specification (origin operand in the PHASE statement) in any of the following formats:

- symbol [(phase)] [± relocation]
- *[± relocation]
- S [+ relocation]
- ROOT.

These operands are described in more detail in *DOS/VS System Control Statements.*

An example of the use of the ACTION F1 statement follows. Assume that three virtual partitions are allocated: background, foreground-two, and foreground-one. If you are executing the linkage editor in the background, the statement PHASE PROG1,S causes PROG1 to have its origin at the beginning of the virtual background partition (plus the BG save area and the BG label area). The sequence

```
ACTION F1
PHASE PROG1,S
```

causes PROG1 to have its origin at the beginning of the virtual foreground-one partition (plus the length of the F1 save area and the F1 label area. The length of the F1 label area is determined from the LBLTYP statement, if any, supplied in the partition in which the linkage editor is running.)

Link-editing for Execution in a Real Partition

If you specify an absolute address in the origin operand of the PHASE statement, the phase is link-edited to be loaded at this specific address. If you specify an origin that is not an absolute address, the phase is link-edited to be loaded in the virtual partition where the linkage editor function occurs, regardless of whether the linkage editor is running in real or virtual mode.

To link-edit a program that will execute in a real partition, you can:

- Link-edit the program in such a way that it can be relocated to the real partition at the time the program is loaded. Relocatable programs are loaded by the relocating loader in a real partition if you specify REAL in the EXEC job control statement. (See the section *Link-editing for Execution at Any Address.*)
- Write the program to be self-relocating if the supervisor does not contain the relocating loader. (See the section *Using Self-Relocating Programs.*)
- Link-edit the program with a PHASE statement that contains an absolute address within a real partition. (See the section *Link-editing for Execution at an Absolute Address.*)

Link-editing for Execution at an Absolute Address

If you specify an absolute address in the PHASE statement (other than zero), your program can be loaded only at this address at execution time. Not only must the address you specified be within the address range of your installation's virtual storage, but also the entire program must be included within the boundaries of the partition where you request the execution.

Using Self-Relocating Programs

You should identify self-relocating programs by a PHASE statement with an origin point of +0:

```
PHASE PROGA,+0
```

The linkage editor assumes that the program is loaded at location zero, and computes all addresses accordingly. The job control EXEC function recognizes a zero phase address and adjusts the origin address to compensate for the current partition boundary save area and label area. It then gives control to the updated entry address of the phase. The programming techniques used in writing self-relocating programs, which are always in assembler language, are described in *DOS/VS Supervisor and I/O Macros.*

Building Phases from Object Modules

You indicate which object modules or parts of object modules are to be included in a phase by specifying the INCLUDE statement. The format of the INCLUDE statement indicates the location of the modules. The object modules can be either on the card reader, tape unit, disk or diskette device assigned to SYSIPT, or in the relocatable library, or on the disk device assigned to SYSLNK. The modules are extracted in the same order as the INCLUDE statements are issued.

Including Modules from SYSIPT

If the object modules you want to include in a phase are on the SYSIPT file, specify the INCLUDE statement without operands. Job control copies the data from SYSIPT until it encounters end-of-data (/*).

Including Modules from the Relocatable Library

You may want to include in a phase object modules or parts of an object module that are cataloged in the relocatable library. To include an entire module, specify the module name in the INCLUDE statement. To include part of a module, specify the name of the module followed by the name of the control section(s) in that module you wish included.

Including Parts of Modules from SYSLNK

You do not need an INCLUDE statement unless you want to change the sequence of control sections or to extract certain control sections from an object module. For either of these cases, specify the names of the control sections in an INCLUDE statement.

Using the AUTOLINK Feature

For each phase the automatic library look-up feature (referred to as AUTOLINK) collects any unresolved external references and attempts to resolve them. An unresolved external reference is an ER item in the control dictionary that has not been matched with an entry point. AUTOLINK searches the private relocatable directory (if assigned) and then the system relocatable directory until a cataloged module with the same name as the unresolved external reference is found (or the end of the directory is reached). If found, the module is included in the phase (autolinked). This retrieved module must have an entry point matching the external reference in order to resolve its address.

Studying the following examples, may help you to understand how the AUTOLINK feature works.

Assume that the relocatable library contains the following:

Module Name	Entry Names	External References
A	A, B, C	
D		A
E		B
F		A, C

Examples:

In your linkage editor input stream you specify INCLUDE D. A will be autolinked (included with module D) because the external reference A is also a module name in the relocatable library.

If you specify INCLUDE E, then A will not be autolinked because the external reference B does not relate to a module name. In this case, you must also specify INCLUDE A, so that the external reference B can be resolved. No autolink is required.

If you specify INCLUDE D and INCLUDE E, then A will be autolinked by module D and the external reference B in module E can then be resolved.

If you specify INCLUDE F, then module A will be autolinked by the reference to A, and the reference to C will also be resolved.

Suppressing the AUTOLINK Feature

You can suppress the AUTOLINK feature in two ways:

- By specifying NOAUTO in a PHASE statement, AUTOLINK is canceled for that phase only.

- By specifying NOAUTO in the ACTION statement, AUTOLINK is canceled for this execution of the linkage editor. By writing a weak external reference (WXTRN), AUTOLINK is canceled for one symbol.

You can do this in assembler language by specifying for example:

```
DC    A(LABEL)
WXTRN   LABEL
```

or

```
DC    V(LABEL)
WXTRN   LABEL
```

For more information, refer to the assembler language publications.

NOAUTO can be used to force a CSECT into a specific phase within an overlay structure. For example, four phases of a program have a V-type address constant called PETE, but in the overlay structure you want the coding for PETE included only in the third phase.

```
PHASE PROGA,*,NOAUTO
PHASE PROGB,*,NOAUTO
PHASE PROGC,*
PHASE PROGD,*,NOAUTO
```

cause PETE to be included in PROGC only.

Reserving Storage for Labels

If your program uses standard tape files or nonsequential DASD files (direct access, indexed sequential, or DTFPH with all packs mounted), you must ensure that storage is reserved for the tape and disk labels. These labels are brought into the label save area of the partition containing your program when the file is opened.

You reserve a label save area by specifying the LBLTYP job control statement. The amount of storage you specify to be reserved must be large enough to contain all the labels of the file with the most extents processed by the program. The operand specified in the LBLTYP statement for tape files is different from that for DASD files. For their formats, refer to *DOS/VS System Control Statements.*

The LBLTYP statement is to be submitted immediately before the // EXEC LNKEDT statement. If your program is self-relocating, however, submit the LBLTYP statement immediately before the // EXEC statement for your program.

The LBLTYP statement is not required if only unlabeled tape files, sequential DASD files, or VSAM files, are to be processed. For more information on file organizations, refer to the *DOS/VS Data Management Guide.* For information on file labeling, refer to *DOS/VS DASD Labels,* or *DOS/VS Tape Labels.*

Specifying Linkage Editor Aids for Problem Determination or Prevention

You can specify that the linkage editor aid you in avoiding certain problems in your programs or determining what they are. The actions discussed below

are CLEAR, MAP, and CANCEL, which may be specified as operands of the ACTION statement.

Clearing the Unused Portion of the Core Image Library

If you used DS (define storage) statements in your source module, it may be advantageous to fill these areas with binary zeros when the program is link-edited. This eliminates the risk that residual data from a previously linked program be loaded with your program at execution time. Such irrelevant data might disrupt your program considerably. By specifying CLEAR in the ACTION statement, you request that the unused portion of the core image library is to be set to binary zeros.

Because CLEAR is a time-consuming function, you might want to use DC statements instead of DS statements when designing future programs.

Obtaining a Storage Map

You can obtain a linkage editor storage map and a listing of linkage editor error diagnostics, which assist you in determining the reasons for particular errors in your program. If SYSLST is assigned, ACTION MAP is the default. You can specify ACTION NOMAP if you are not interested in this service of the linkage editor.

The storage map contains such information as:

- The lowest and highest addresses that each phase occupies in the partition for which it is link-edited.
- The starting disk address of the phase in the core image library.
- The names of all control sections and entry points, their load addresses and relocation factors.
- The names of all external references that are unresolved.
- An indication whether the phase is relocatable, non-relocatable, self-relocating, or SVA eligible.

The error diagnostics warn you, for example, if:

- The ROOT phase has been overlaid.
- A control section has a length of zero.
- An address constant could not be resolved.

A sample storage map, together with a description of how to interpret it, is included in *DOS/VS Serviceability Aids and Debugging Procedures.*

Terminating an Erroneous Job

If errors are present in the input to the linkage editor, the output of the linkage editor will most likely also be erroneous. If you specify CANCEL in the ACTION statement, the entire job is terminated when the type of errors represented by messages 2100I through 2170I occur. Refer to these messages in *DOS/VS Messages.*

Designing an Overlay Program

The nature of virtual storage makes it unnecessary to write programs in an overlay structure, because virtual partitions can be allocated to accommodate very large programs.

Organizing Control Sections in an Overlay Tree Structure

Overlay programs consist of control sections organized in an overlay tree structure. A tree is a graphic representation that shows how phases use storage at different times. An example of an overlay tree structure is shown in Figure 6.5. This structure does not imply the order of execution, although the root phase is normally the first to receive control.

The manner in which control should pass between control sections is discussed in the section *Using FETCH and LOAD Macros.*

Relating Control Sections to Phases

After having organized the control sections of your program into an overlay tree structure, you must prepare a corresponding set of linkage editor control statements. If you first want to test the program, specify // OPTION LINK. When you are satisfied that the overlay structure you designed is a workable combination, specify // OPTION CATAL to catalog a permanent copy of the program in the core image library.

Link-edit your complete overlay program in a single job step, and conversely, do not include in this job step any phases that are not related to the overlay. Otherwise, the linkage editor may not be able to resolve external references correctly.

The PHASE and INCLUDE statements you prepare are critical to ensure the overlay tree structure you designed. Figure 6.6 is an example of the job stream that ensures the overlay tree structure shown in Figure 6.5.

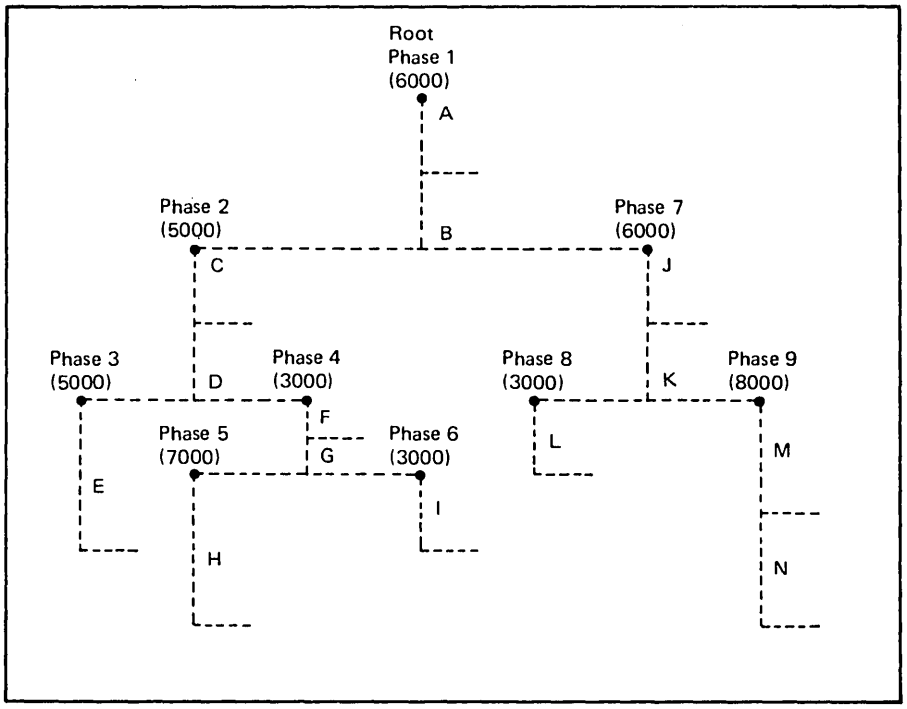


Figure 6.5. Overlay Tree Structure

The letters A through N represent control sections, which are organized to form nine phases in one program. The root phase resides in storage during the entire execution of the program. The remaining phases can overlay each other during execution.

You must guarantee a partition size that is equal to the longest combination of phases that can possibly reside in storage together, namely, phases 1, 2, 4, and 5, which total 21,000 bytes. If the program had not been organized in an overlay structure, it would have required an address space of 46,000 bytes.

```
// JOB OVERLAY
// OPTION CATAL
   PHASE   PHASE1,ROOT          PHASE1 stays in storage during
   INCLUDE ,(CSECTA,CSECTB)     execution of the entire program.
   PHASE   PHASE2,*             PHASE2 is to be loaded
   INCLUDE ,(CSECTC,CSECTD)     immediately behind PHASE1.
   PHASE   PHASE3,*             Since PHASE3 needs PHASE2, PHASE3 is
   INCLUDE ,(CSECTE)            not allowed to overlay PHASE2.
   PHASE   PHASE4,PHASE3        PHASE4 will occupy the same
   INCLUDE ,(CSECTF,CSECTG)     storage locations as PHASE3.
   PHASE   PHASE5,*             PHASE5 will be loaded
   INCLUDE ,(CSECTH)            immediately behind PHASE4.
   PHASE   PHASE6,PHASE5        PHASE6 will be loaded at the
   INCLUDE ,(CSECTI)            same address as PHASE5.
   PHASE   PHASE7,PHASE2        PHASE7 will be loaded at the
   INCLUDE ,(CSECTJ,CSECTK)     end of the root phase.
   PHASE   PHASE8,*             PHASE8 will be loaded at the
   INCLUDE ,(CSECTL)            end of PHASE7.
   PHASE   PHASE9,PHASE8        PHASE9 will overlay
   INCLUDE ,(CSECTM,CSECTN)     PHASE8.
   INCLUDE
      (Object modules containing CSECTs A through N)
/*
// LBLTYP
// EXEC LNKEDT
/&
```

Figure 6.6. Link-editing an Overlay Program

Using FETCH and LOAD Macros

During execution, an overlay program communicates with the supervisor to request that a subsequent phase be brought into the partition. You include FETCH or LOAD macros within your phases for this purpose.

Use a LOAD macro in a phase that is to remain in control after the requested phase is brought into the partition. A phase loaded by the LOAD macro is relocated (if necessary) so that the displacement between the start of the partition and the beginning of the phase is the same as at link-edit time. By using a LOAD macro with an explicit address, you can violate the overlay tree structure you defined. When a relocatable phase is loaded, all address constants will be relocated with the same relocation factor as computed for that phase. This means that address constants referring to entry points in other phases of this same relocatable program will be incorrect.

Use a FETCH macro if you want the requested phase to gain control immediately after it is brought into the partition. If a phase loaded by the FETCH macro is relocatable, it will be relocated if necessary. You cannot issue a FETCH macro for a self-relocating phase.

Parameters in FETCH and LOAD allow these macros to use the SDL and to execute code from the SVA, thereby reducing fetching and loading time. The benefits that stem from using the SDL apply to phases that are used frequently throughout the day by many programs in an installation. For a phase that is used heavily at one time only, however, it is preferable to use the GENL macro rather than to include the phase in the SDL. The

GENL macro places the directory entry of a phase in storage, where it can be accessed rapidly by FETCH and LOAD for use by the program that requires it.

DOS/VS Supervisor and I/O Macros contains details on the format of the FETCH, LOAD and GENL macros.

Summary of Control Statements Related to Link-Editing

The following sections summarize the linkage editor control statements and the job control statements that are associated with a linkage editor run. This summary is provided to make it easier for you when referencing the formats of the statements in *DOS/VS System Control Statements.*

Job Control Statements

The job control statements that relate to a linkage editor job stream and that are summarized below are:

- `// OPTION  CATAL`
  `LINK`
- `// LBLTYP`

OPTION

To make use of the linkage editor, you must specify either the LINK or CATAL operands in the OPTION job control statement. These options set switches in the supervisor that are tested when the linkage editor program is requested. Linkage editor control statements are accepted only after one of these options has been specified. SYSLNK must be assigned; otherwise, the LINK and CATAL options are ignored (switches are not set).

By specifying the LINK option (// OPTION LINK), you indicate that the output of the language translators is to be written on the SYSLNK file. Because SYSLNK is the required input file for each linkage editor operation, the CATAL option (// OPTION CATAL) also sets the LINK switch. The differences between LINK and CATAL are described below.

The CATAL option causes a phase to be entered permanently into the core image library. The object module is link-edited and placed in the first available area of the core image library (immediately after the last cataloged phase). An entry identifying the name of the phase, load address, entry point, and starting disk address of the phase in the core image library is then inserted in alphameric sequence in the core image directory for cataloged phases. If the same phase name was previously cataloged, the new directory entry replaces the old. A status report of the core image library and directory is then printed.

The LINK option causes a phase to be entered temporarily in the core image library in order to be executed immediately; that is, for an assemble, link-edit, and execute operation, or a link-edit and execute operation. The linkage editor prepares the phase just as described for the CATAL option, except that an entry is made in the core image directory for linked phases. When you specify the EXEC statement without the program name operand the phase is executed immediately. The space taken up by the phase in the

core image library is overwritten by the next phase cataloged or linked to the core image library. No status report is printed.

LBLTYP

The LBLTYP job control statement reserves a label save area for tape labels or DASD labels. You must specify the LBLTYP statement if your program uses standard tape files or nonsequential DASD files.

For a non-self-relocating program, you must submit the LBLTYP statement immediately before the // EXEC LNKEDT statement. For a self-relocating program, you must submit this card immediately before the // EXEC statement for the program.

Linkage Editor Control Statements

The linkage editor control statements that are summarized below are:

- ACTION
- PHASE
- INCLUDE
- ENTRY.

ACTION

ACTION statements, if used, must be the first statements in the linkage editor input stream. An ACTION statement is effective only for this linkage editor execution.

The ACTION statement can indicate that the linkage editor do any or all of the following:

- Set the unused portion of the core image library to binary zeros (CLEAR).
- Write a storage map and error diagnostics on SYSLST (MAP), or not (NOMAP).
- Suppress the automatic library lookup feature for this entire linkage editor run (NOAUTO).
- Terminate the job if errors are present in the linkage editor input (CANCEL).
- Link-edit the program to run in a specific virtual partition.
- Produce a relocatable program if possible (REL) or do not produce a relocatable program (NOREL).

PHASE

The PHASE statement indicates the beginning of a phase by providing the linkage editor with the phase name and the storage address (origin point) where the phase is to be loaded. The origin point defines whether the phase is to be relocatable, self-relocating, or non-relocatable. The PHASE statement may also indicate that the automatic library lookup feature (AUTOLINK) be canceled for this phase only, that the phase is considered to be SVA eligible, or that the load point of the phase be aligned on a page boundary.

The first (or only) object module in the input for the linkage editor should include a PHASE statement before the first ESD item. A PHASE

statement must be supplied if you specify the CATAL option. A PHASE statement is not required if you specify the LINK option.

INCLUDE — The INCLUDE statement specifies that an object module is to be included for link-editing. The format of the statement indicates where the object module is located and whether all or parts of it are to be included. The object module may be on SYSIPT or SYSLNK, or cataloged in the relocatable library.

ENTRY — The ENTRY statement signals the end of the input to the linkage editor. If the entry point operand is used it also indicates a transfer address for the first phase (the name of a control section or label definition). In case of a label definition, it must occur in an ENTRY source statement.

Examples of Linkage Editor Applications

The linkage editor examples on the following pages illustrate the use of and relation between the control statements just discussed. After studying these examples, you should be able to set up a link-edit job for your own purposes.

Catalog to Core Image Library Example

```
   // JOB CATALCIL
   *  LINK EDIT AND CATALOG TO CORE IMAGE LIBRARY
   *  SINGLE PHASE, ELIGIBLE FOR LOADING INTO SHARED
   *  VIRTUAL AREA, MULTIPLE OBJECT MODULES,
   *  MIXTURE OF CATALOGED AND UNCATALOGED OBJECT MODULES
   *  LABELED TAPE FILES AND SEQUENTIAL DASD FILES TO
   *  BE PROCESSED
1  // ASSGN SYSLNK,X'190'
2  // OPTION CATAL
3     PHASE PROGB,*,SVA
4     INCLUDE
      Object deck
   /*

      INCLUDE SUBRX
      INCLUDE SUBRY
      INCLUDE
      Object deck
   /*
   // LBLTYP TAPE
5  // EXEC LNKEDT
6  /&
```

Explanation for Catalog to Core Image Library. This example illustrates the cataloging of a single phase composed of multiple object modules. These modules are located in the input stream and the relocatable library. Labeled tape files and sequential DASD files are processed when the phase is executed. The program is to be executed in a foreground partition. The linkage editor run occurs in the virtual background partition.

Statement 1: The SYSLNK assignment indicates the relationship of ASSGN statements to the OPTION statement. ASSGN statements are not required if they are standard assignments.

Statement 2: The OPTION CATAL statement sets the LINK switch, as well as the CATAL switch. If SYSLNK is not assigned, the statement is ignored. The linkage editor control statements are not accepted unless the OPTION statement is processed. Link-editing and cataloging to the core image library will occur.

Statement 3: Only one PHASE is constructed. It is cataloged to the core image library and retrieved by the name PROGB. Because there is only one phase, the origin point * indicates that this phase originates at the starting address of the virtual partition plus the length of the partition save area, the label area (if any), and the COMMON pool (if any). The SVA operand indicates that the phase should be considered SVA-eligible. If the phase name PROGB is already entered as SVA-eligible in the system directory list, PROGB is loaded into the shared virtual area immediately after it is cataloged into the system core image library. (This could not occur had PROGB been link-edited with OPTION LINK.)

Note: *COMMON is used by FORTRAN programs to store data shared by multiple programs.*

Statement 4: Four modules make up this phase. The first and last are not cataloged in the relocatable library; therefore the object decks must be on SYSIPT, and each must be followed by the end-of-data record (/*). SUBRX and SUBRY were cataloged previously to the relocatable library by those names. Job control puts the uncataloged modules on SYSLNK in place of their INCLUDE statements. Job control copies onto SYSLNK the INCLUDE statements for the cataloged modules.

Statement 5: The EXEC LNKEDT statement causes the system loader to bring in the linkage editor program. SYSLNK now becomes input to the linkage editor. It contains the following:

```
PHASE PROGB,F+32768
First uncataloged relocatable deck
INCLUDE SUBRX
INCLUDE SUBRY
Second uncataloged relocatable deck
ENTRY
```

The modules are link-edited into one phase so that they occupy contiguous addresses in the sequence in which they appear in the input stream. When the linkage editing is completed, cataloging to the core image libary occurs because of the CATAL option.

The core image directory is checked to make sure the new phase entry fits. If not, the job is canceled. The directory for cataloged phases is scanned for any match to a phase being cataloged. If there is a match, the earlier directory entry is replaced by the new entry. The descriptor entry is updated to reflect the changes. Job control is brought into the virtual background partition.

Because the CATAL option was specified, a status report is printed to reflect the usage and available space in the core image library. (This does not occur in a LINK situation.)

Statement 6: The / & resets the CATAL option, that is, it turns off the LINK and CATAL switches.

The example can be modified to illustrate a catalog-and-execute operation by inserting the following statements between the EXEC LNKEDT and / & statements:

1. Any job control statements required for execution of PROGB.
2. A // EXEC statement
3. Any card reader input for PROGB.

Catalog to Private Core Image Library Example

```
   // JOB CATLCIL
   *  LINK EDIT AND CATALOG TO PRIVATE CORE IMAGE LIBRARY
   *  LINKAGE EDITOR EXECUTING IN FOREGROUND
   *  SINGLE PHASE, ALIGNED ON A PAGE BOUNDARY, MULTIPLE
   *  OBJECT MODULES, FOREGROUND PROGRAM
   *  MIXTURE OF CATALOGED AND UNCATALOGED OBJECT MODULES
   *  LABELED TAPE FILES AND SEQUENTIAL DASD FILES TO
   *  BE PROCESSED
1   ASSGN SYSCLB,X'130'
2  // ASSGN SYSLNK,X'190'
3  // OPTION CATAL
4     PHASE PROGB,S,PBDY
5     INCLUDE
      object deck
   /*
      INCLUDE SUBRX
      INCLUDE SUBRY
      INCLUDE
      Object deck
   /*
6  // LBLTYP TAPE
7  // EXEC LNKEDT
8  /&
```

Explanation for Catalog to Private Core Image Library. This example illustrates the execution of the linkage editor in a foreground partition; therefore the phase is cataloged to a private core image library. This function is possible only in a system supporting multiple-partitions and private core image library options. The phase being cataloged is the same as that in the previous example where the linkage editor was executed in the background.

Statement 1: The assignment of a private library is accomplished by the ASSGN SYSCLB command. The label for SYSCLB must be stored on PARSTD or STDLABEL cylinder, or, of the DLBL and EXTENT statements are included in the job stream, they must precede the ASSGN SYSCLB command.

Statement 2: The SYSLNK assignment indicates the relationship of ASSGN statements to the OPTION statement. ASSGN statements are not required if they are standard assignments.

Statement 3: The OPTION CATAL statement sets the LINK switch, as well as a CATAL switch. If SYSLNK is not assigned, the statement is ignored. The linkage editor control statements are not accepted unless the OPTION statement is processed. Linkage editing and cataloging to the private core image library will occur.

Statement 4: Only one PHASE is constructed. It is cataloged to the private core image library and retrieved by the name PROGB. An origin point of S origins PROGB at the starting address of the foreground partition, plus the length of the save area and the label area (if any) and the COMMON pool (if any). PBDY indicates that the load point of the phase is to be aligned on a page boundary.

Statement 5: Four modules make up this phase. The first and last are not cataloged in the relocatable library; therefore, the object decks must be on SYSIPT, and each must be followed by the end-of-data record (/*). SUBRX and SUBRY were cataloged previously to the system relocatable library by those names. Job control puts the uncataloged modules on SYSLNK in place of their INCLUDE statements. Job control copies onto SYSLNK the INCLUDE statements for the cataloged modules.

Statement 6: The LBLTYP statement has the operand TAPE, rather than NSD, because labeled tapes and sequential DASD files are processed when the phase is executed. 80 bytes are reserved ahead of the actual phase for label information. LBLTYP NSD is also satisfactory because it generates a minimum of 104 bytes and tapes require only 80.

Statement 7: The EXEC LNKEDT statement causes the system loader to bring in the linkage editor program. SYSLNK now becomes input to the linkage editor. It contains the following:

```
PHASE PROGB,S,PBDY
First uncataloged relocatable deck
INCLUDE SUBRX
INCLUDE SUBRY
Second uncataloged relocatable deck
ENTRY
```

The modules are link-edited so that they occupy contiguous areas in storage in the sequence in which they appear in the input stream. When link-editing is completed, cataloging to the private core image library occurs because of the CATAL option. The private core image directory is checked to make sure the new phase entry fits. If not, the job is canceled. The directory is scanned for any match to a phase being cataloged. If a match is found, the earlier phase directory entry is replaced. The system library descriptor record is updated to reflect the changes. Job control is then brought into this virtual foreground partition.

Because CATAL was specified, a status report is printed to reflect the usage and the available space in the private core image library and directory. (This does not occur in a LINK situation.)

Statement 8: The / & resets the CATAL option, that is, it turns off the LINK and CATAL switches.

The example can be modified to illustrate a catalog-and-execute operation by inserting the following statements between the EXEC LNKEDT and / & statements:

1. Any job control statements required for execution of PROGB
2. A // EXEC statement
3. Any card reader input for PROGB.

Link-Edit and Execute Example

```
   // JOB LINKEXEC
   *  LINK EDIT AND EXECUTE SINGLE PHASE, SINGLE OBJECT
   *  MODULE NOT CATALOGED, BACKGROUND PROGRAM
   *  NONSEQUENTIAL DASD & LABELED TAPE FILES TO
   *  BE PROCESSED
1  // ASSGN SYSLNK,X'190'
2  // OPTION LINK
3      PHASE PROGA,*
4      INCLUDE
      object deck
   /*
5  // LBLTYP NSD(2)
6  // EXEC LNKEDT
7      Any job control statement required for execution
      such as ASSGN or label statements
8  // EXEC
      Input Data as required
   /*
   /&
   * 1TO CATALOG AND EXECUTE, CHANGE STATEMENT @
   *  TO // OPTION CATAL
   * 2TO CATALOG ONLY, CHANGE STATEMENT 2 TO
   *  // OPTION CATAL AND REMOVE ALL STATEMENTS
   *  FOLLOWING EXEC LINKEDT EXCEPT /&
   * 3TO USE A MODULE FROM RELOCATABLE LIBRARY,
   *  CHANGE STATEMENT 4 TO INCLUDE MODULES AND
   *  REMOVE ALL STATEMENTS UP TO // LBLTYP AND
   *  IF THE PHASE CARD IS IN THE RELOCATABLE LIBRARY,
   *  ALSO REMOVE STATEMENT 3.
```

Explanation for Link-edit and Execute. This example illustrates the basic concept of link-editing and executing by using a single phase that is constructed from a single object module contained in punched cards. Labeled tape and nonsequential DASD files are to be processed when the phase is executed. No more than two extents are used by any DASD file.

Statement 1: No assignments are necessary because the system units required for link-editing are in the assumed configuration. However, an ASSGN for SYSLNK is included to illustrate its position relative to the OPTION statement in case an assignment is required.

Statement 2: The OPTION LINK statement indicates that a link-edit operation is to be performed. If SYSLNK has not been assigned, the statement is ignored. Linkage editor control statements are not accepted

unless the OPTION statement is processed. Because the option is LINK, and not CATAL, only link-editing is performed; permanent cataloging to the core image library does not occur. To catalog, change the statement to // OPTION CATAL.

Statement 3: The PHASE statement is copied on SYSLNK because the LINK switch is on. The first operand is checked; the following operands are not examined until SYSLNK is used as input to the linkage editor program.

When the PHASE statement is processed by the linkage editor, only one phase is constructed, because only one PHASE statement is submitted for the entire run. The name of this phase is PROGA, as specified in the first operand. The second operand indicates the origin point for the phase. Because an * has been used, the phase begins in the next storage location available, with forced doubleword alignment. Because this is the first and only phase, it is located at the beginning of the virtual partition plus the length of the save area and label area (reserved by LBLTYP) plus the length of any area assigned to the COMMON pool (as designated by a CM entry in the object module).

A displacement, either plus or minus, may be used with the *, such as *+1024. This causes the origin point of the phase to be set relative to the * by the amount of the displacement. This displacement is expressed as:

X'hhhhhh' -- 1 to 6 hexadecimal digits
ddddddddd -- 1 to 8 decimal digits
nK -- where K = 1024.

*+1024 uses the second format and adds 1024 bytes to the origin location. +1K or +X'400' gives the same result as +1024.

Statement 4: The INCLUDE statement has no operands so the system reads the records from SYSIPT and writes them on SYSLNK until SYSIPT has an end-of-data (/*) record. The data on SYSIPT is expected to be the object module in card image format that is used in this linkage editor operation.

Statement 5: The LBLTYP statement causes a computation of the number of bytes that are required for label data in the program to be link-edited. In this example, 124 bytes are reserved (84 + [2x20]). The calculation is saved by job control and passed on first to the linkage editor and later to LIOCS.

Statement 6: On encountering the EXEC LNKEDT statement, job control writes an ENTRY statement with no operand on SYSLNK and causes the system loader to bring in the linkage editor program.

Using the data just placed on SYSLNK as input, the linkage editor develops executable code. The output is placed in the next available space of the core image library (immediately after the last cataloged phase). This is true regardless of whether the program is cataloged permanently (OPTION CATAL) or temporarily (OPTION LINK). Cataloging permanently causes the updating of the library descriptor entry in the core image directory to reflect a new ending point for the library. If OPTION LINK is specified, however, the next program that is link-edited overlays it.

For this reason, a program that is cataloged temporarily is said to be placed in the temporary area of the core image library. Such a program must be link-edited each time it is used. No ACTION options are specified. Therefore, in resolving the external references, the system makes use of the AUTOLINK feature. Error diagnostics and a storage map are written on SYSLST, assuming that SYSLST is assigned and ACTION NOMAP was not specified.

Statement 7: Because the program is not cataloged, it must be executed immediately. Any pertinent job control statements are entered at this point.

Statement 8: An EXEC statement with no program name operand indicates that the phase to be executed was just link-edited. Therefore, no search of the core image directory for linked phases is required, and the system loader brings the program into storage from the temporary area and transfers control to its enty point. Because the automatic ENTRY statement is in effect for this example, the entry point is either the address specified in the END record, or the phase load address if the END address is omitted.

This example can be modified to illustrate the following:

1. *Catalog and execute.* To cause this phase to be cataloged permanently, change the OPTION (statement 2) from LINK to CATAL.

2. *Catalog only.* To catalog only, change the OPTION (statement 2) from LINK to CATAL and remove all statements following the EXEC LNKEDT (statement 6) up to the / & statement.

3. *Include object module from relocatable library.* The name of the object module in the relocatable library must be added to the INCLUDE statement. If the name is RELOCA, the statement becomes INCLUDE RELOCA. The relocatable object deck and /* statement are removed. This form of the INCLUDE statement is written on SYSLNK when it is read by job control. The linkage editor retrieves the object module when it encounters the INCLUDE statement because it uses SYSLNK for input.

Compile and Execute Example

```
   // JOB COMPEXEC
   *  COMPILE OR ASSEMBLE, LINK EDIT AND EXECUTE
   *  SINGLE PHASE, MULTIPLE OBJECT MODULES, BACKGROUND
   *  PROGRAM SEQUENTIAL DASD FILES TO BE PROCESSED
   *  INPUT TO LINKAGE EDITOR FROM LANGUAGE TRANSLATOR,
   *  RELOCATABLE LIBRARY AND SYSIPT
1  // ASSGN SYSLNK,X'190'
2  // OPTION LINK
3       PHASE PROGA,S
4  // EXEC FCOBOL
      COBOL source statements
   /*
      INCLUDE SUBRX
5     INCLUDE
      object module
   /*
6     ENTRY BEGIN1
   // EXEC LNKEDT
```

```
7     Any job control statements required for PROGA
      execution
   // EXEC
      Any input data required for PROGA execution
   /*
   /&
```

Explanation for Compile and Execute. The language translators provide the option of placing their output on SYSLNK. Because the linkage editor uses SYSLNK for input, a program can be assembled or compiled, link-edited and executed. This operation is illustrated by this example.

All three sources of object module input to the linkage editor are used: SYSIPT, the relocatable library, and the output from a language translator. It is assumed that the phase is executed in the background partition, and that only sequential DASD files or unlabeled tape files are processed.

Statement 1: The SYSLNK assignment is given to illustrate the relationship of ASSGN statements to the OPTION statement. ASSGN statements are not required if they are standard assignments.

Statement 2: Because SYSLNK is assigned, the OPTION LINK statement sets the link indicator switches.

Statement 3: The PHASE statement must always precede the relocatable modules to which it applies; therefore, it is written on SYSLNK first for later use by the linkage editor. S is the origin point, that is, the phase originates with the first doubleword at the end of the supervisor plus the length of the partition save area and label area plus the length of the area assigned to the COMMON pool (if any). This gives the same effect as * gives for a single phase or the first phase. As with the *, the S may be used with a relocation factor, for example, S+1024. The factor must always be positive, because a negative factor could cause the origin point to overlay the supervisor.

Statement 4: The appropriate language translator is called (in this case, COBOL). The normal rules for compiling are followed; the source deck must be on the unit assigned to SYSIPT and the /* defines the end of the source data. Because the LINK switches are set, the output of the language translator is written on SYSLNK. Except for PL/I, FORTRAN(F), ANS or VS COBOL, and the assembler, the DECK option is ignored when SYSLNK is used.

Statement 5: The INCLUDE SUBRX statement is written on SYSLNK. The linkage editor retrieves the named module from the relocatable library. Because the operand is blank, the next INCLUDE statement signifies that the relocatable module is on SYSIPT. The data on SYSIPT is copied on SYSLNK up to the /* statement.

Statement 6: The ENTRY statement is written on SYSLNK as the last linkage editor control statement. The symbol BEGIN1 must be the name of a CSECT or a label definition (which occurs in an ENTRY source statement) defined in the first or only phase. The address of BEGIN1 becomes the transfer address for the first or only phase of the program. The ENTRY is used to provide a specific entry point rather than to use the point specified in the END record or the load address of the phase.

Statement 7: No LBLTYP statement is required, because only sequential DASD files are to be processed. The rest of the statements follow the same pattern as discussed in the Link-Edit and Execute example. The input from SYSLNK to the linkage editor is:

```
PHASE PROGA,S
Relocatable module produced by COBOL compilation
INCLUDE SUBRX
Relocatable module from SYSIPT
ENTRY BEGIN1
```

If certain types of errors are detected during compilation of a source program, the LINK option is suppressed. Under these circumstances the EXEC LNKEDT and EXEC statements are ignored and the message 'STATEMENT OUT OF SEQUENCE' results. This LINK option suppression should be kept in mind if a series of programs is to be compiled and cataloged as a single job. Failure of one job step would cause failure of all succeeding steps.

An OPTION LINK cannot be given if OPTION CATAL is in effect. The message 'STATEMENT OUT OF SEQUENCE' results. Therefore, the CATAL switch remains on (until reset), and link-editing only cannot be performed.

Chapter 7: Using the Libraries

After you have planned the size, contents, and location of the libraries (see *Chapter 3: Planning the System*), you need to know how to allocate space to a library, how to create private libraries and how to alter, copy, and inspect the contents of the libraries. All these functions are performed by a group of library processing programs, collectively referred to as the librarian.

This chapter describes how you can use the librarian to manage the system and private libraries in your installation. The chapter is divided into three major sections:

- The first section looks at the libraries from a system point of view, that is, it shows how the system stores or retrieves elements into or from the libraries. Although knowledge of this internal processing is not essential for working with the libraries, it contributes to a better understanding of the librarian functions.
- The second section introduces the three functional components of the librarian (maintenance, organization, and service) and gives a detailed description of their applications to the individual libraries.
- The third section describes the creation and use of private libraries.

The information in this chapter is useful for programmers and perhaps also for operators.

How the System Accesses the Libraries

DOS/VS supports four types of libraries. Their purpose and contents are described in *Chapter 3: Planning the System* and are summarized here:

- *Core image library* -- contains the output from the linkage editor (executable program phases).
- *Relocatable library* -- contains the output of a language translator (object modules) which is used as input to the linkage editor.
- *Source statement library* -- contains books (source language statements, macro definitions, and pre-edited macro definitions) used as input to a language translator.
- *Procedure library* -- contains collections of frequently-used control statements (cataloged procedures). These cataloged procedures can include job control and linkage editor control statements and (if the SYSFIL option was specified during supervisor generation) inline SYSIPT data.

The following describes how these libraries are accessed by the system when a maintenance function for one of the libraries is requested.

The Directories

Associated with each library is a directory that occupies the first track(s) allocated to the library. For each element in a library, the corresponding directory contains a unique entry describing the element. A directory entry contains such information as name, disk address, size, load address (core image library only), and version number (relocatable, source statement, and procedure libraries only) of the element. These directory entries are used by the system to locate and retrieve elements from a library.

The begin addresses of the individual system library directories are stored in a separate directory, the system directory. For the core image library, the first entry of the core image directory (library descriptor entry) contains such information as the address of the next available record, the number of active and deleted blocks, and the amount of space allocated to the library. For the other libraries, this information is contained in the first five entries of their own directories.

A core image library normally contains a large number of program phases. Thus, searching for a specific phase can become rather time consuming. To reduce the search time, the phases in the core image library are entered in the core image directory in alphameric sequence. The highest phase name on each track of the core image directory is listed in the second level directory contained in the supervisor. If the phase cataloged is eligible for the shared virtual area (SVA) (that is, its phase name is already entered with the SVA operand in the system directory list, and it was cataloged in the core image library with the SVA operand), the phase is loaded into the SVA. When requested for execution, such a phase is always available in virtual storage.

The organization of the directories on SYSRES is shown in Figure 7.1. A more detailed description of the complete SYSRES organization is given in *Appendix A: System Layout on Disk.*

Naming Elements in the Libraries

The choice of a phase name has a bearing on retrieval efficiency and the subsequent use of the librarian programs. In general, you should not catalog a phase with the same name as a phase already residing in the core image library. When you do, the earlier phase-name entry is deleted from the core image directory (and, if applicable, the system directory list) and cannot be accessed again.

Job control scans the directory of the appropriate library for all phases starting with the same four characters as the program name specified in the EXEC statement. The highest storage address of these phases is stored in the communication region of the partition. All phases just link-edited will be taken if no program name is specified in the EXEC statement.

Phase names may only be formed from the characters 0-9, A-Z, /, #, $, and @. Otherwise, the phase card is invalid.

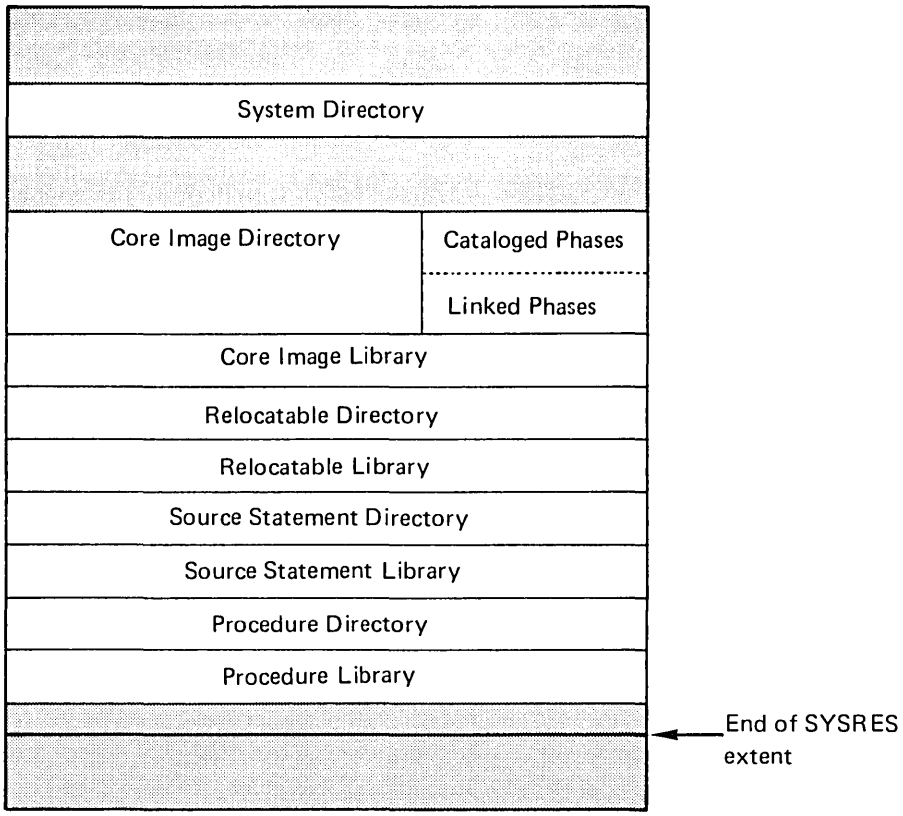


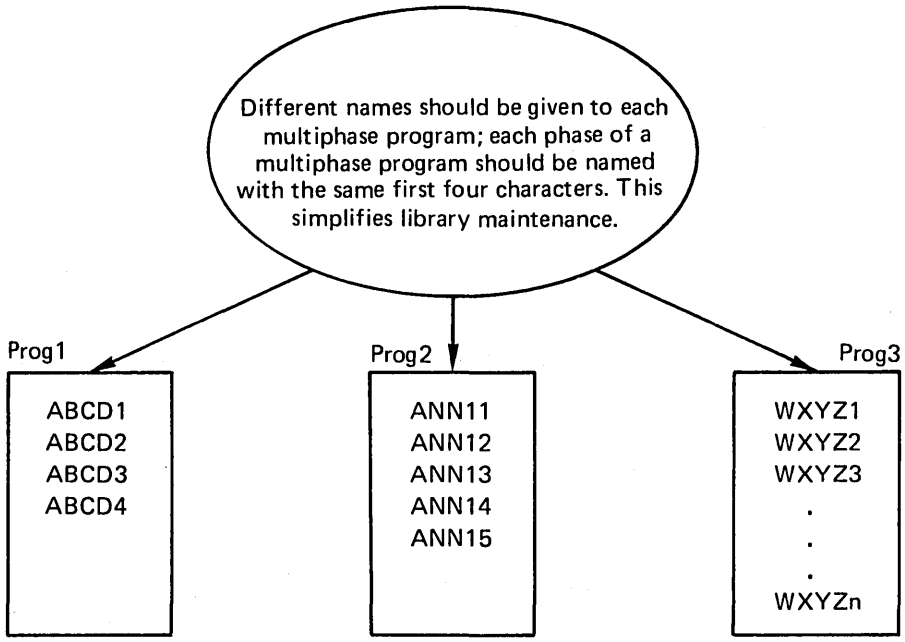
Figure 7.1. Organization of the Directories on SYSRES

In choosing a name for any multiphase program, make sure that the first four characters are the same for all phases of that program but different from those of other programs. Such unique names simplify the procedures of deleting, displaying, punching, merging, and copying the entire program. Figure 7.2 summarizes the above recommendations.

There is one other restriction when choosing a phase name. The linkage editor interprets the phase name "ALL" as invalid because this would subsequently be misinterpreted by the librarian programs. (This applies to the control statements DELETC ALL and COPYC ALL.) Also 'S' and 'ROOT' as phase names may cause calculation of an invalid load address.

Certain special naming considerations apply depending on the library in which an element is stored:

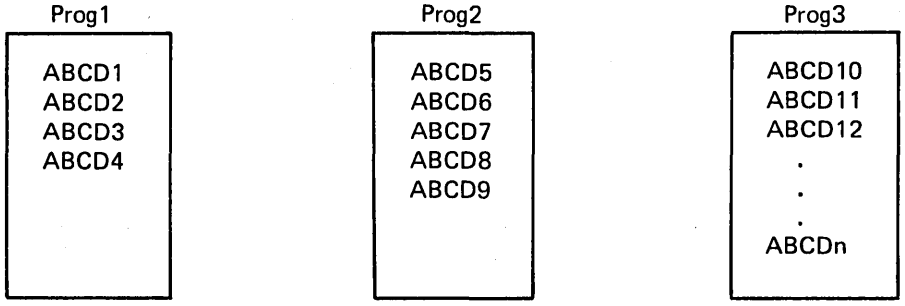
- **Core Image Library.** The names of some IBM programs in the core image library begin with $; the IBM programs are normally stored in the system core image library where they can be retrieved faster. User-program names should not begin with $, because this is specifically reserved for certain IBM programs, and user programs should be placed in a private core image library if fast retrieval is desired. The reason for this is that the system searches the system core image directory first for phase names beginning with $ and the private library directory first for other phase names, provided a private library is assigned to the partition in question. When an attention routine command results in a phase being loaded into virtual storage, DOS/VS



Simplified library maintenance means, for example, that one simple control statement deletes all phases of Prog1:

```
DELETC ABCD.ALL
```

If the programs had been named:



the statement required to delete Prog1 would be:

```
DELETC ABCD1, ABCD2, ABCD3, ABCD4
```

Figure 7.2. Naming Multiphase Programs

searches only the system core image library. Phases such as buffer image phases for an FCB or a UCB are to be cataloged in the system core image library and not in a private core image library.

- **Relocatable Library.** User-written modules should not use names beginning with I since this is used as the first letter of the names of IBM-supplied modules.

- **Source Statement Library.** Initial letters A, B, C, D, E, F, G, H, I, and Z refer to sublibraries reserved for IBM components, and you should avoid as far as possible cataloging into one of these reserved sublibraries. If you have an earlier version of DOS with books cataloged in one of the sublibraries reserved under DOS/VS, you can easily transfer them by using the librarian rename function.
- **Procedure Library.** Names for procedures cataloged in the procedure library can consist of any combination of alphanumeric characters. The naming convention to follow when creating partition-related cataloged procedures is given in *Chapter 5: Controlling Jobs.*

Change levels can be appended to names of elements in the relocatable, source statement, and procedure libraries to help you keep track of the current versions of your programs. The change level is specified in the catalog control statement, and the procedure is described in detail later in this chapter under *Cataloging.*

Storing and Accessing Elements in the Libraries

Whenever an element is to be stored (cataloged) in one of the libraries, the system:

- obtains the address of the library directory from the system directory
- determines the locations in the directory and the library where the directory entry and the element should be placed.
- places the element into the library and creates a new directory entry; searches for duplicate entries and, if found, deletes the earlier entry.

If a phase is added to the core image library, the applicable information in the library descriptor entry is updated. If the phase is eligible for the shared virtual area and is indicated as SVA-eligible in the system directory list, the phase is also loaded into the SVA. The second level directory is updated, if necessary.

In general, the library elements and their respective directory entries appear in the order in which they were cataloged. For the core image library, however, the directory entries are sorted in alphameric sequence.

Source statements cataloged in the source statement library are stored in compressed form, that is, all blanks are eliminated. When a source statement book is retrieved, the statements are expanded to their original 80-character format. Control statements in the procedure library are not compressed but are stored in card image format.

To access an element in a library, the system searches the corresponding directory if it contains an entry with the name of the requested element.

Working with the Libraries

This section describes how you can manage and control your libraries with the use of the librarian programs. The librarian programs fall into three functional groups: maintenance, organization, and service. The functions are applicable both to the system and private libraries. Figure 7.3 is a summary

of the librarian programs, their functions, and partitions allowed for execution.

GROUP	PROGRAM NAME	FUNCTIONS	PARTITION (EXEC)
Maintenance	MAINT	Catalog Delete Condense Reallocate (Note 1) Rename Update	BG BG (Note 2) BG (Note 3) BG BG (Note 2) BG
Organization	CORGZ	Create a new system pack Create private libraries Transfer elements between any two libraries of the same type	ANY (Note 4)
Service	DSERV	Display the contents of the library directories	ANY
	CSERV RSERV SSERV PSERV	Display, punch, or display and punch the contents of the Core image, Relocatable, Source statement, or Procedure library. Display, update the contents of the assembler sublibrary (in source statement format).	
	ESERV	Convert edited macros to source format. Display and/or catalog converted macros in the source statement library.	

Note 1 Reallocate cannot be used for private libaries.

Note 2 This function may be executed in a foreground partition for a private core image library.

Note 3 Refer to figure 7.6 for restrictions related to execution of the CONDS and CONDL functions of the MAINT program.

Note 4 A MERGE to SYSRES function must be run in BG.

Figure 7.3. Summary of Librarian Programs and Their Functions

You invoke the individual functions of the librarian programs by means of librarian control statements. The use of these control statements is described and demonstrated by examples in the following section. Their formats are contained in *DOS/VS System Control Statements*.

Note 1: *If the extended support for the procedure library (SYSFIL) was selected during supervisor generation, the librarian control statements can be cataloged into the procedure library. This excludes maintenance functions for the procedure library itself and reallocation of library sizes.*

Note2: *If a cataloged procedure is used to start POWER/VS no maintenance functions can be performed on the procedure library as long as POWER/VS is active.*

Note3: *Results may be unpredictable if librarian programs access a library while this library is being updated in another partition. Therefore , if a private library is assigned to more than one partition, the library should not be updated.*

Processing Requirements

No special considerations apply to executing the librarian programs in a virtual partition. If you wish to run a librarian program other than MAINT, CORGZ, or DSERV in either a real partition or a large virtual partition, specify AUTO in the SIZE parameter of the EXEC job control statement. Since MAINT, CORGZ, and DSERV dynamically allocate storage during execution, the SIZE=AUTO specification should not be used for these programs; SIZE=64K should be specified instead.

The merge function (for copying elements onto SYSRES) of the CORGZ program and most functions of the MAINT program must always be executed in the background partition (real or virtual). If the relocating loader is not used, note the following statements. The MAINT, CSERV, and DSERV programs are self-relocating so that they can be executed in any partition. The ESERV, PSERV, RSERV, SSERV, and CORGZ programs run only in the virtual background partition, unless they were link-edited to be relocatable and loaded by the relocating loader.

The table in figure 7.3 shows which partitions you can use to have the librarian programs executed. For instance, the CORGZ program may be used to create both system and private libaries in any partition.

When you execute MAINT in a foreground partition, a private core image library must be uniquely assigned to that partition. The maintenance functions then apply only to this private core image library. Neither the system libraries nor the private relocatable or source statement libraries can be accessed by MAINT executing in the foreground.

Maintaining the Libraries

The maintenance functions of the librarian will probably be the ones most frequently used in daily operation. They include:

1. Cataloging elements to the libraries
2. Deleting elements from the libraries
3. Condensing the libraries (or establishing limits for automatic condense)
4. Allocating space to the libraries
5. Renaming elements of the libraries
6. Updating books in the source statement library.

The maintenance program is invoked by the job control statement:

```
// EXEC MAINT
```

The functions to be performed are specified in librarian control statements which must follow the EXEC MAINT statement on SYSIPT. (If SYSIPT is assigned to a tape unit, it must be a single file and a single volume.) Any combination of the maintenance functions can be performed in a single run. A sample maintenance job in skeleton form is shown below:

```
// JOB ANYMAINT
   .
   .
assignments, if necessary
   .
   .
// EXEC MAINT
   .
   .
librarian control statements
   .
   .
/*
/&
```

When the /* is processed after completion of the maintenance run, a status report of the library just updated is printed on SYSLST.

The symbolic unit assignments required for the individual maintenance functions are described in *DOS/VS System Control Statements.* The

examples in this chapter assume that all necessary assignments are established as standard assignments.

Cataloging

The catalog function adds a module to a relocatable library, a book to a source statement library, or a procedure to the procedure library. You cannot use the catalog function of the librarian to add a phase to the core image library; this is done by the linkage editor (see *Chapter 6: Linking Programs*).

The catalog control statements specify the name of the element to be cataloged and, optionally, a change level number. The control statements are:

```
Relocatable library . . . . . CATALR
Source statement library  . . . . . CATALS
Procedure library . . . . . CATALP
```

Elements added to a library by cataloging can be removed by deleting (see *Deleting,* later in this section). Under certain circumstances the catalog function itself implies a delete function. For instance, if a module to be cataloged has the same name as a module already existing in the relocatable library, the existing module is automatically deleted and the new module is cataloged. No warning message is issued. The same is true for a book in the source statement library and a procedure in the procedure library.

When you add to the contents of a library, watch the status of the system directory, which is printed at the end of the catalog run. If the libraries are becoming full, you may wish to condense them or allocate more space to them. (Condensing and allocating are described later in this section.)

Cataloging to the Relocatable Library. To catalog an object module to the relocatable library you must submit the object deck on SYSIPT following the CATALR control statement. The following job catalogs two object modules, named MOD1 and MOD2, to the relocatable library; the object decks were produced by language translators in previous jobs:

```
// JOB CATREL
// EXEC MAINT
   CATALR MOD1
   .
   .
object deck for MOD1
   .
   .
   CATALR MOD2
   .
   .
object deck for MOD2
   .
   .
/*
/&
```

You can also compile or assemble a program and catalog the resulting object module in the relocatable library in the same job, without obtaining a card deck of the object module. In this case, you assign SYSPCH, which receives the output of the language translator, to a disk, diskette or tape and then use the object module on that device as input to the MAINT program. An example using a magnetic tape for SYSPCH is shown in

Figure 7.4. To assign SYSPCH to a disk, or diskette, the SYSFIL option must have been specified during supervisor generation, and you must supply the necessary DLBL and EXTENT job control statements (see also *System Files on Tape, Disk, or Diskette* in *Chapter 5: Controlling Jobs*).

```
   // JOB CATREL
   // OPTION DECK
1  // ASSGN SYSPCH,X'180'
2  // EXEC ASSEMBLY
3           PUNCH  'CATALR MODULE1'
   source module
      .
   /*
4  // MTC WTM,SYSPCH,2
5  // MTC REW,SYSPCH
6  // RESET SYSPCH
7  // ASSGN SYSIPT,X'180'
8  // EXEC MAINT
   /&
```

1	A magnetic tape device is assigned to SYSPCH to receive the assembler output.
2	The assembler processes the source module and writes the object module onto SYSPCH following the CATALR statement.
3	The assembler will punch a CATALR statement.
4	Tapemarks are written on SYSPCH to indicate the end of the object module.
5	The tape is rewound to its load point.
6	The tape is unassigned as SYSPCH.
7	The tape is assigned to SYSIPT to serve as input for the MAINT program.
8	MAINT reads the object module from the tape and catalogs it in the relocatable library.

Figure 7.4. Assembling and Cataloging to the Relocatable Library in the Same Job

All modules in the relocatable library that have the first three characters of the module name in common are considered to belong to one program. This simplifies the control statements to delete, display, punch, merge, and copy an entire program. The names of IBM-supplied modules in the relocatable library begin with the letter I, which should therefore be considered reserved so that you can readily distinguish your modules from IBM's.

Cataloging to the Source Statement Library. To add a book to the source statement library you use the CATALS statement specifying the name of the book and the sublibrary to which it belongs. A sublibrary is defined by an alphameric character preceding the bookname. For example, the statement

```
CATALS   P.NEWBOOK
```

adds the book NEWBOOK to sublibrary P. Note that the sublibraries in the range from A to I, and Z are reserved for IBM components.

A -- is the assembler copy sublibrary. It contains books of assembler source code and source macro definitions. See *DOS/VS System Control Statements* for details.

B -- is the VTAM network definition sublibrary.

D -- is the alternate copy sublibrary. It contains non-edited macros and copy books for programs that are to be executed in a teleprocessing network control unit.

E -- is the assembler macro sublibrary. It contains IBM-supplied and user-written macro definitions in an edited (partially processed) format. See *DOS/VS System Control Statements* for details.

F -- is the alternate macro sublibrary. IBM uses it to distribute edited macros for use by programs that are to be executed in a teleprocessing network control unit.

C -- is the COBOL sublibrary.

Z -- contains sample programs supplied by IBM.

The rest of the reserved characters (G, H, I) will be used by IBM for future additions to the source statement library. You should avoid, wherever possible, cataloging to one of the reserved sublibraries. If you must catalog to a sublibrary that is reserved for IBM components, ensure that you do not use duplicate names. You can obtain a listing of the contents of each sublibrary by means of the SSERV librarian program (see *Using the Service Functions of the Librarian* later in this chapter). You can obtain a listing of the book names within each sublibrary by means of the DSERV librarian program.

Users of previous versions of DOS, who have books in a sublibrary which is reserved under DOS/VS can easily transfer this sublibrary from the *IBM range* to the *user range* by means of the librarian rename function (see *Renaming,* later in this section).

Edited macro definitions that are to be cataloged in the assembler sublibrary must be preceded by a MACRO statement and followed by a MEND statement. Example:

```
// JOB CATMAC
// EXEC MAINT
   CATALS E.MBOOK
   MACRO
    .
edited macro definition statements
    .
   MEND
/*
/&
```

Books other than macro definitions that are to be cataloged must be preceded and followed by BKEND-statements. Example:

```
// JOB CATBOOK
// EXEC MAINT
   CATALS P.SBOOK
   BKEND
    .
    .
source statements
    .
    .
   BKEND
/*
/&
```

The BKEND statement can have optional operands specifying that a sequence check or a card count be performed on the statements to be cataloged, or that the book to be cataloged is in compressed format. If you desire these functions when you catalog a macro definition, BKEND

statements can be included in addition to the MACRO and MEND statements.

Cataloging to the Procedure Library. To catalog a procedure in the procedure library you submit a CATALP statement specifying the procedure name. Procedure names can consist of any combination of alphanumeric characters. The control statements to be cataloged follow the CATALP statement; they can be job control or linkage editor control statements or both. The end of the control statements to be cataloged must be indicated by /+.

Each control statement cataloged in the procedure library should have a unique identity. This identity is required if you want to be able to modify the job stream at execution time. Therefore, when cataloging, identify each control statement in columns 73-79 (blanks may be embedded). Refer also to the section *Modifying Cataloged Procedures* in *Chapter 5: Controlling Jobs.*

The following job catalogs the procedure PROCA in the procedure library:

```
// JOB CATPROC
// EXEC MAINT
   CATALP PROCA
   .
   .
control statements to be cataloged
   .
   .
/+ END OF PROCEDURE
/*
/&
```

If your supervisor was generated with the SYSFIL option, you can also include inline SYSIPT data in the cataloged procedure. The presence of SYSIPT data must be indicated to the MAINT program by the DATA parameter of the CATALP statement. In addition, you must indicate the end of inline data by the /* statement. The following example catalogs a procedure consisting of control statements and SYSIPT data:

```
// JOB CATPROC
// EXEC MAINT
   CATALP PROCA,DATA=YES
   .
   .
control statements
   .
   .
   SYSIPT data
   .
   .
/* END OF SYSIPT DATA
   .
   .
control statements
   .
   .
/+ END OF PROCEDURE
/*
/&
```

The following restrictions apply when you catalog procedures to the procedure library:

1. A cataloged procedure cannot contain control statements or SYSIPT data for more than one job.
2. If the cataloged control statements include the JOB statement you must not have a JOB statement when you retrieve the procedure through the EXEC statement.
3. A cataloged procedure with DATA=YES must not include either of the following statements:

   ```
   [//] RESET SYS
   [//] RESET ALL
   ```

   In addition, it must not include any of the following statements for SYSIN, SYSRDR, or SYSIPT:

   ```
   [//] ASSGN
   [//] CLOSE
   [//] RESET
   /&
   ```

4. A cataloged procedure with DATA=NO must not include either of the following statements:

   ```
   [//] RESET SYS
   [//] RESET ALL
   ```

   In addition, it must not include any of the following statements for SYSIN or SYSRDR:

   ```
   [//] ASSGN
   [//] CLOSE
   [//] RESET
   /&
   ```

5. Cataloged procedures cannot be nested, that is, a cataloged procedure cannot contain an EXEC statement that invokes another cataloged procedure.
6. When cataloging a procedure that contains an imbedded // JOB statement, in a POWER/VS controlled partition, POWER/VS * $$ JOB and * $$ EOJ statements must define the cataloging job.

Refer to *Chapter 5: Controlling Jobs* for a detailed description of how to retrieve cataloged procedures from the procedure library and how to modify cataloged control statements using the overwrite facility.

Assigning Change Levels. When you catalog an element in one of the libraries, you can assign a change level to the element, which will enable you to keep track of the current version of your programs. The change level is specified in the catalog control statement by a version and a modification number. The following statement catalogs version 1, modification 3, of module MOD1 in the relocatable library:

```
CATALR   MOD1,1.3
```

Change levels are stored in the directory entry for the element and can be displayed by the librarian service program DSERV. A change level is not used by the system for identification purposes, that is, a change level is *not* sufficient to allow two elements having the same name to coexist in a library.

For the source statement library only, you can request verification of the change level before a book is updated. This can prevent an accidental updating of the wrong version of a book in a particular sublibrary. Specify

the character C in the CATALS statement to request change level verification. Example:

```
CATALS    M.BOOK1,1.1,C
```

To update the book you must supply the current change level of the book in the update control statement. This change level is then checked against the change level in the directory entry and, if they match, the book is updated and its change level is increased by one to reflect the new status of the book. If you want to overwrite the version and modification numbers of a book, supply the new change level information in the END statement of the update function. If change level verification is requested for a particular book, the letter C will appear in the column headed LEV CHK (level check) in the DSERV listing.

Deleting

You can delete an unwanted element from a library either by cataloging a new element with the same name or by means of the delete function of the librarian, using the following control statements:

```
Core image library . . . . . DELETC
Relocatable library  . . . . . DELETR
Source statement library . . . . . DELETS
Procedure library  . . . . . DELETP
```

To delete individual elements from the libraries, you must specify each element name in full in the delete control statement. If a group of elements is to be deleted, however, you can simplify the specification of the control statement provided that the recommended naming conventions were used when the elements were cataloged.

1. If all the phases of one program in the core image library were named with the same first four characters, you need to specify only these four characters to delete the entire program.
2. You can delete all modules in the relocatable library that have the first three characters in common by specifying these three characters in one delete control statement.
3. Similarly, you can delete an entire sublibrary from the source statement library by specifying the sublibrary name.

Since no special naming conventions apply to the procedure library, each cataloged procedure to be deleted must be individually specified.

You can also use the delete function to remove all elements of a relocatable library, source statement library, procedure library, or *private* core image library. In this case, the system directory information is updated to show that all blocks of the library in question are available for cataloging programs; no condense operation is required. You cannot delete the entire *system* core image library, but only individual phases or programs.

The following job deletes (1) all phases starting with PHAS from the core image library, (2) modules MOD1 and MOD2 from the relocatable library, (3) sublibrary P from the source statement library, and (4) all the elements of the procedure library:

```
// JOB DELETE
// EXEC MAINT
```

```
   DELETC PHAS.ALL
   DELETR MOD1,MOD2
   DELETS P.ALL
   DELETP ALL
/*
/&
```

When you request the deletion of a library element, the name of the element is removed from the corresponding directory entry. The system is then no longer able to recognize the element although it is still physically present in the library. The area taken up by such an element can be referred to as unavailable free space. To make such space available again for cataloging programs, use the condense function. The delete and condense functions are illustrated in Figure 7.5.

When a phase is deleted from the core image library, it is also flagged as not present in the system directory list (if applicable). The shared virtual area cannot be condensed; it must be recreated. See *Building the SDL and Loading the SVA* in *Chapter 4: Starting the System.*)

Condensing

When you delete an element from a library, the space occupied by the 'deleted' element -- referred to as unavailable free space -- is unavailable for cataloging new elements (see Figure 7.4). To make this space available for cataloging, you use the condense function of the MAINT program.

To condense any of the libraries you use the CONDS control statement specifying which of the libraries is (are) to be condensed. The following job condenses the core image, relocatable, and source statement libraries after the deletion of elements from the libraries:

```
// JOB DELCOND
// EXEC MAINT
   DELETC PHAS1,PHAS5,PROGA
   DELETR MOD.ALL
   DELETS P.ALL
   DELETP ALL
   CONDS CL,RL,SL
/*
/&
```

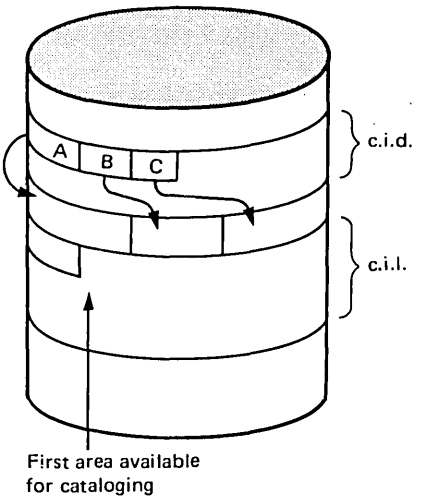
Note that you need not condense a library -- in the above example, the procedure library -- if that library is deleted entirely.

The reallocation function of the MAINT program automatically causes the libraries to be condensed. Refer to the section *Reallocating.*

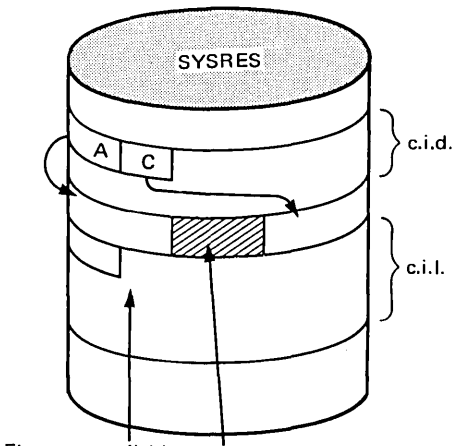
If a condense operation is interrupted by a hardware error or by operator intervention before the next statement is read, the library being condensed is unusable and must be reconstructed. Note that the condense program shows all the symptoms of a looping program, but should never be canceled by the operator.

Specifying the condense limit. You can specify that a message is to be delivered to the operator whenever the number of available blocks in a library drops below a specified minimum, which is referred to as the condense limit. Through the CONDL statement you specify the library or libraries and the condense limit(s).

(1) Assume that phases A, B, and C are cataloged in the core image library (c.i.l.). Each core image directory (c.i.d.) entry, which refers to one of these phases, points to the beginning disk address of the phase.



(2) If phase B is no longer desired in the core image library, specify `DELETC B`, which deletes the name B from the directory.



(3) To make full use of the core image library, eliminate the unavailable free spaces by specifying `CONDS CL`.

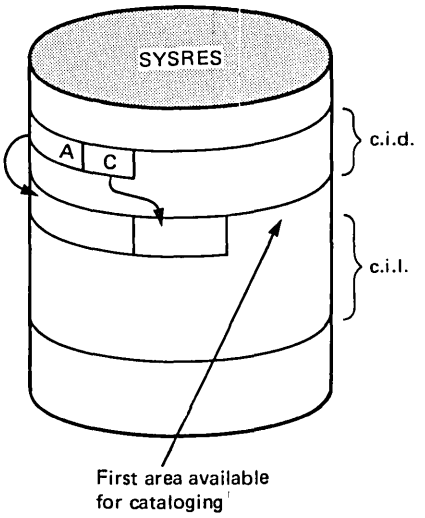


Figure 7.5 Example of Deleting and Condensing

Example:

```
// JOB CONDSLMT
// EXEC MAINT
   CONDL CL=10
/*
/&
```

In the above example, the CONDL statement specifies that, whenever the number of available library blocks falls below 10, a message is to be issued. (The block size of the core image library is 1024 bytes.)

The condense limit should always be less than the number of blocks allocated to the library; otherwise this message is given after each maintenance function. The MAINT program stores the condense limits in the system directory, which can be displayed at the end of each librarian maintenance job. If a library has reached a condense limit, this is indicated in the status report by a note.

When Condense Can Be Performed. While the condense function is being executed, the library directories do not represent the actual status of the library. Thus, if a program in any partition were to attempt to use the library in any way, the results would be unpredictable. For this reason, various controls are provided to minimize the chances of unpredictable results:

- The system core image library and either the system or private relocatable and source statement libraries can only be condensed from the background partition, and then only if there are no active foreground partitions.
- A private core image library may be condensed in any partition, provided it is exclusively assigned to that partition.
- The procedure library can be condensed from the background partition unless it is being accessed by the job control program in another partition or a procedure is being executed. Thus, a job stream to condense the procedure library cannot be cataloged.

A summary of where/when condense can be performed is shown in Figure 7.6.

	Core Image		Relocatable		Source Statement		Procedure (system)
	System	Private	System	Private	System	Private	
CONDS	Yes if FG is inactive.	Yes if issued from the only partition to which the PCIL is assigned.	Yes if FG is inactive.				Yes if not being accessed by job control or if a procedure is not being executed.

Figure 7.6. When Can Condense Be Performed?

The CONDL control statement (which sets the condense limits) can be submitted with the MAINT program at any time.

A partition is inactive if it has never been activated with a START or BATCH command or has been deactivated with an UNBATCH command.

Even though POWER/VS may not be doing any work, if it is resident in a partition, the partition is considered to be active.

Reallocating

You can use the reallocation function of the MAINT program to

- increase the size of a library for further additions
- decrease the size of a library, for example, to provide space for expanding other libraries.
- eliminate a library if it is replaced by a private library or is no longer required
- reestablish a library after it has been eliminated.

Each library that is reallocated is automatically condensed. You can reallocate any combination of the system libraries on SYSRES within a single run. You *cannot* reallocate private libraries. To change the track and cylinder allocation of a private library, you must create a new private library using the CORGZ program (see *Creating Private Libraries,* later in this chapter). If a private library is assigned and you attempt to reallocate the corresponding system library, a message is issued and the job is canceled.

The reallocation function of the MAINT program must always be executed in the background partition and all foreground partitions must be inactive. This ensures that no program can access any library during reallocation; otherwise, the results would be most unreliable because the final addresses may not have been established and (similar to the condense function) because the directory entries do not reflect the actual status of the libraries until end-of-job.

You invoke the reallocation function through the ALLOC control statement. In the operand field you specify the libraries to be reallocated, the number of cylinders to be allotted to each library, and the number of tracks to be reserved for the library directory. The ALLOC statement can be submitted together with any other maintenance control statements.

Changing the Size of the Libraries. When you increase the size of one library, you must consider the space remaining for the libraries that follow. The combined number of cylinders for all system libraries cannot exceed

198 for 2314 or 2319
402 for 3330 or 3333
804 for 3330-11
345 for 3340 with 3348 data module Model 35
693 for 3340 with 3348 data module Model 70.
554 for 3350

Note: *You must allow 1 cylinder for the VTOC and 1 or 2 (on 3340) cylinders for the label cylinder(s).*

If not enough space is available for the following libraries, you must reduce one or more of these libraries to compensate for the increase.

Assume, for example, that the SYSRES library space on a 2314 was allocated during system generation as

```
ALLOC CL=90(5),RL=40(2),SL=60(3),PL=6(5)
```

An attempt to reallocate only the core image library to 120 cylinders would fail, because cylinder 199 would be exceeded. To avoid this, you can reduce the combined sizes of the relocatable and source statement libraries by 28 cylinders. In this case, the ALLOC statement should read:

```
ALLOC CL=120(7),RL=30(2),SL=42(3),PL=6(5)
```

When you alter the size of the SYSRES file by reallocating libraries, you must define the new SYSRES extent by means of DLBL and EXTENT job control statements. The new SYSRES extent must begin with cylinder 0, track 1, and end with the last track of the label cylinder. The ALLOC statement starts calculating from cylinder 0, track 0. This means that EXTENT information for the SYSRES file must be one cylinder (or two for 3340) larger than the total number of cylinders specified in the ALLOC statement to include the label cylinder(s).

The following example shows the job control statements required to reallocate the system libraries as discussed above when the SYSRES device type is 2314/2319:

```
// JOB REORG
// DLBL IJSYSRS,'DOS/VS SYSTEM RESIDENCE',99/365
// EXTENT SYSRES,111111,1,0,0001,3979
// EXEC MAINT
   ALLOC CL=120(7),RL=30(2),SL=42(3),PL=6(5)
/*
/&
```

Note that the filename specified in the DLBL statement for the SYSRES file must always be IJSYSRS. The new label information for the SYSRES file is stored in the volume table of contents (VTOC) of the SYSRES pack.

No special considerations apply for reducing the size of a library except that you must also supply the necessary label information for the new SYSRES extent. Reducing a library does not cause any gaps, that is, the libraries following the one that was reduced are 'moved up' to close the gap.

Eliminating Libraries. If you have created a private relocatable or source statement library containing all the modules or books that you require from the corresponding system library, you can use the reallocation function to eliminate that system library. You do this by setting the track and cylinder indications in the ALLOC statement to zero. This is only effective, however, if all the directory entries have first been cleared by the DELETS or DELETR control statements.

Similarly, you can eliminate the procedure library if it contains no active elements and you are sure that you do not want to use cataloged procedures.

The following job eliminates the system relocatable library. The example assumes that the libraries were allocated with CL=80(5), RL=40(2), SL=30(3), PL=10(5). (SYSRES device type assumed to be 2314/2319.)

```
// JOB ELIMNT
// DLBL IJSYSRS,'DOS/VS SYSTEM RESIDENCE',99/365
// EXTENT SYSRES,111111,1,0,0001,3219
// EXEC MAINT
   DELETR ALL
   ALLOC RL=0(0),CL=120(7),SL=30(3),PL=10(5)
/*
/&
```

You cannot eliminate the system core image library because it is required for system operation. If you inadvertently specify a zero allocation for the system core image library, the job is canceled.

Once eliminated, the relocatable, source statement, or procedure library can be added again to the SYSRES file. The same considerations apply to adding a library as to increasing the size of a library. Using the reallocation function to add a library does not include adding the actual elements of the library. Once a library exists you can add elements either by cataloging or by merging from a private library or another SYSRES. (The merge function is described in *Copying and Reorganizing the Libraries,* later in this chapter.)

Renaming

To change a name of a cataloged phase, module, book, or procedure, use the rename function. In a control statement unique to each type of library, you supply the existing name and the name to which you want to change it. If the *new* name is identical to a name already cataloged in the library, an error message is issued. You must then select a different name and resubmit the job.

When you name a phase in the system core image library that is also listed in the system directory list, the phase name is changed in both directories.

After a valid rename control statement is processed, the system recognizes only the new name. The version and modification level (change level) is not changed by the rename function.

Each type of library has a unique rename control statement:

```
Core image library . . . . . RENAMC
Relocatable library  . . . . . RENAMR
Source statement library . . . . . RENAMS
Procedure library  . . . . . RENAMP
```

The rename function can be used to establish naming conventions. All phases in the core image library that have the first four characters in common are considered to belong to one program. All modules in the relocatable library that have the first three characters in common are considered to belong to one program. Since the names of IBM-supplied relocatable modules begin with the letter I, it is advantageous to avoid this first character when naming user modules. Similarly, you should avoid the use of the first characters A-I and Z when renaming sublibraries in the source statement library. These prefixes are reserved for IBM-supplied components. Names for procedures cataloged in the procedure library can consist of any combination of alphanumeric characters.

Renaming a member of a library can be advantageous in a testing environment. For instance, after making changes to your source deck, rename the previous version residing in the library and catalog the new source under the original name. This assures you of backup until your new program is in working order, at which time you can delete the old (renamed) version(s).

Updating Object Modules and Phases

During or after system generation, you may need to update already cataloged object modules or phases. The IBM system utility Copy File and Maintain Object Module (OBJMAINT), allows you to modify object code by submitting approriate control statements via card, tape, sequential disk or diskette; the program supports the following functions:

LIST	provides formatted listings of the object code
ACTION	allows OBJMAINT processing of SYSIN files on disk, tape or diskette
BLOCK	blocks output data to tape or disk
SELECT	selects jobs to be processed from an input file containing SYSIN data
EXCLUDE	excludes jobs from an input file
UNREP	eliminates any or all user REP cards in a named CSECT
EXIT	names a precataloged user phase to optionally process the input file
CARD	specifies an alternate EOF delimiter, other than /*, when input is on cards
EXPAND	allows specification of a new CSECT length
REP	allows object code patching; may be followed with user REP statements or used to advance to a subsequent CSECT.

EXPAND/REP allows both CSECT length modification and patching with user REP statements

COPY allows file-to-file copying with blocking and deblocking capability

DEBLOCK same as COPY except that the output is always in 80-byte blocks

END required when doing a LIST only operation with card input.

To update object programs cataloged in either the relocatable or the core image library, you must

1. use RSERV or CSERV to write the object modules to be modified to SYSPCH, which should be assigned to tape, disk, or diskette

2. execute OBJMAINT with control statements on SYSIPT or from the procedure library. Input object modules are assigned to SYS004 and OBJMAINT output is assigned to SYS005.

3. execute MAINT or LNKEDT to catalog updated object modules.

The following job stream shows the three steps involved in updating an object module, PAYRL01, from the relocatable library. After PAYRL01 is written out to disk, OBJMAINT is used to remove any prior updates (user REP statements) before adding a new user REP statement. The updated module, on tape, is then recataloged using the MAINT program.

```
   // JOB UPDTMOD
   // DLBL IJSYSPH,'PTF.WORK.FILE.1',0,SD
1  // EXTENT SYSPCH,CPMDY5,,,5681,38
   // ASSGN SYSPCH,X'130',PERM,VOL=CPMDY5,SHR
   // EXEC RSERV
2    PUNCH PAYRL01
   /*
3  CLOSE SYSPCH,X'00D'
   // DLBL UIN,'PTF.WORK.FILE.1',0,SD
4  // EXTENT SYS004,CPMDY5
   // ASSGN SYS004,3330,TEMP,VOL=CPMDY5,SHR
5  // ASSGN SYS005,X'281'
   // MTC REW,SYS005
6  // MTC WTM,SYS005
   // MTC REW,SYS005
   // EXEC OBJMAINT
7  ./ LIST PARM=SHORTTXT
8  ./ UNREP
9  ./ REP NM=PAYRL01,SD=(001)
10 +REP 000010 001FFFF,FFFF,FFFF,FFFF,FFFF,FFFF,FFFF,FFFF
   /*
11 // MTC REW,SYS005
   // RESET SYS005
12 // ASSGN SYSIPT,X'281'
13 // EXEC MAINT
   // MTC REW,SYSIPT
   // RESET SYSIPT
   /*
14 // EXEC RSERV
    DSPLY PAYRL01
   /*
   /&
```

1 Label and device assignment information required for the SYSPCH disk file, to be used by the RSERV program. The assignment is a permanent one (PERM), with disk volume CPMDY5 available for multiple assignments (SHR).

2 These statements cause writing of the object module PAYRL01 to SYSPCH from the relocatable library. A CATALR statement is included on SYSPCH.

3 The SYSPCH FILE IS CLOSED AND SYSPCH assigned to device, X'00D'.

4 Label and device assignment information for the disk file containing object module PAYRL01, now assigned to SYS004 as OBJMAINT input.

5 Output of OBJMAINT (in deblocked SYSIN format) will be written to the SYS005 tape on drive X'281'.

6 These statements cause writing of a tape mark and rewinding of the SYS005 tape.

7 This OBJMAINT LIST function will print one TXT statement per line.

8 This OBJMAINT UNREP function will eliminate any existing user REP statements from PAYRL01.

9 This OBJMAINT REP function will allow application of new user REP statements.

10 This user REP statement (X'02' or + in column 1) contains the new object code modification for PAYRL01.

11 After the OBJMAINT step the output tape is repositioned to the beginning; SYS005 assignment is reset to the default.

12 The SYS005 output file of OBJMAINT becomes the SYSIPT file for the MAINT program.

13 The updated module is recataloged to the relocatable library.

14 RSERV is used to list PAYRL01 for verification and documentation of the update.

For further details on the use of OBJMAINT, refer to *DOS/VS System Utilities*; for details on the format of the user REP statement for the patching of object modules, refer to *DOS/VS System Control Statements*.

Updating the Source Statement Library

The update function applies only to a source statement library. This function revises one or more source statements within a particular book. By using update you can make minor changes to a book, without having to catalog an entire, new book.

Besides adding, deleting, or replacing a certain number of source statements within a book, the update function allows you to:

- resequence statements within a book
- revise a change level (version and modification) of a book
- add or remove the requirement for change level verification
- copy an entire book and rename the old book (for backup purposes).

The UPDATE control statement identifies the update function. This statement may also be followed by one or more of these additional statements as required:

ADD -- To add source statements
DEL -- To delete source statements
REP -- To replace source statements.

The END statement indicates the end of updates to the particular book specified in the UPDATE control statement.

If the requirement for change level verification was specified in the CATALS control statement when a book was cataloged, the version and modification level must be specified in the UPDATE control statement that refers to this book. This change level must agree with the current change level in the directory entry for that book. (Check the DSERV listing for the current change level and/or requirement for change level verification. For more information on the DSERV program, refer to the section *Displaying the Directories.*) The specification of the version and modification level in the UPDATE statement prevents you from inadvertently making an update based on a book with the the wrong version and modification. Regardless of whether or not the requirement is in effect, the version and modification level are incremented by one after each update. If a version and modification level is specified in the END statement, this overrides the current change level.

Organizing the Libraries

The copy (CORGZ) program is an important tool for establishing and organizing your libraries during system generation or any time thereafter. The following section discusses this program, its functions, and its application to your library organization requirements. Its functions are to:

- Create a new system residence (SYSRES)
- Transfer elements between any two existing libraries of the same type, as follows:
  - all elements, or
  - some elements, or
  - only those elements which do not yet exist in the receiving library.
- Create private libraries.

The first two points are described in this section. The creation of private libraries is discussed in *Creating and Working with Private Libraries,* later in this chapter.

The organization program can be executed in any partition, except for the merge function (to copy elements onto SYSRES), which must be executed in the background partition. It is invoked by the statement

```
// EXEC CORGZ
```

When /* is processed after completion of the CORGZ program, a status report of the library just updated is printed on SYSLST.

You cannot have different device types for input and output.

The functions to be performed by the CORGZ program are specified in a set of librarian control statements, which will be introduced in the course of the following discussions.

Note: *The library-compare and copy-select functions of the COPYSER V program, which was made available with release 33, have been integrated in the CORGZ program, and the COPYSER V program is removed from DOS/VS. Integrating these COPYSER V functions in CORGZ eliminates the need for an extra job when two libraries are to be merged.*

When system generation is completed, you will want a backup SYSRES, which can save you regenerating the system from your distribution medium if the operational pack is inadvertently destroyed. This backup SYSRES is usually kept on tape, but may also be kept on a disk of the same device type as the original SYSRES. If the backup is to be on tape, use the Backup/Restore DOS/VS System program. When required the tape may be restored to a disk of the same or different device type. If the backup SYSRES is to be on disk, use the CORGZ program with the ALLOC and a COPY control statements to define the new SYSRES file and copy the entire contents of the original SYSRES file onto it.

You can also copy the SYSRES file selectively; that is, the new system residence will contain only part of the original SYSRES. This may be useful in an installation that uses certain components only during specific processing periods. For instance, if teleprocessing and support for five partitions is required only during the prime shift, a different system configuration (for instance, no teleprocessing and three partitions) could be used during the second shift. Therefore, you could copy onto a new SYSRES file only those components required for the second shift and add any additional components needed to that SYSRES. In this case, you must assemble a new supervisor and catalog it into the new SYSRES file. The effect is a smaller supervisor and smaller libraries on both system residence packs which means faster access to library elements and, thus, improved overall system performance.

When you create a new system residence, SYS002 must be assigned to the device on which the new SYSRES pack resides. In addition, you must define the extents of the new SYSRES file by means of DLBL and EXTENT job control statements. The filename in the DLBL statement must be IJSYSRS. The lower extent must be cylinder zero, track one, and the upper extent must include the label information cylinder(s). The information to be copied from the original to the new SYSRES is specified in one or more of the following COPY control statements:

COPY ALL	to copy the entire system residence file. Note that you can use this form of the COPY statement only if all four system libraries are allocated on the original SYSRES file; otherwise, you must use a combination of the following COPY statements.
COPYC	to copy one or more elements, one or more
COPYR	groups of elements, or all elements of the
COPYS	**C**ore image, **R**elocatable, **S**ource statement
COPYP	or **P**rocedure library.

The following job creates a backup SYSRES file on disk drive X'131'. The example assumes that the original SYSRES file does not contain a procedure library:

```
// JOB BACKUP
// ASSGN SYS002,X'131'
// DLBL IJSYSRS,'DOS/VS SYSRES BACKUP',99/365,SD
// EXTENT SYS002,111111,1,0,0001,2219
// EXEC CORGZ
   ALLOC CL=50(5),RL=30(5),SL=30(5),PL=0(0)
```

```
   COPYC ALL
   COPYR ALL
   COPYS ALL
/*
/&
```

For each CORGZ run an ALLOC control statement is required, preceding any COPY statements. If you wish to exclude an entire library from being copied, specify a 'zero' allocation (for example, RL=0(0)).

Assume that you have a SYSRES file that contains all four system libraries and you want to create a second SYSRES file containing only selected information from the core image library and the entire relocatable library. The following job creates this new SYSRES file (device type 2314/2319 assumed):

```
// JOB SYSRES
// ASSGN SYS002,X'131'
// DLBL IJSYSRS,'DOS/VS SYSRES II',99/365,SD
// EXTENT SYS002,111111,1,0,0001,1619
// EXEC CORGZ
   ALLOC CL=50(5),RL=30(5),SL=0(0),PL=0(0)
   COPYC PHAS.ALL,PROG.ALL,ABCD.ALL
   COPYR ALL
/*
/&
```

Note that all components essential to a minimum system are copied automatically by the CORGZ program. These components are:

- Supervisor
- Initial program loader (IPL)
- All logical and physical transients
- Job control
- Linkage editor
- Partition and system standard labels (cataloged with the PARSTD and STDLABEL options) from the label information cylinder(s).

Thus, if you execute the CORGZ program without any COPY statements, the above components will be copied automatically onto the new SYSRES file.

Transferring Elements between Libraries

If you work with more than one system residence pack or private library, you may want to transfer elements from one library to another. Instead of punching the elements into cards and re-cataloging them, you can use the CORGZ program with a MERGE statement to transfer the elements. This is especially useful for system generation when a new version of the system is installed; you can then copy the library elements directly from the old version to the new one. (For backup purposes you should of course have a duplicate of the library to which elements are transferred.)

You use the MERGE control statement to define the characteristics of the libraries to be merged and the direction of transfer between the libraries. The operands of the MERGE control statement are:

RES -- For the system libraries on the system residence file

NRS -- For the system libraries on a modified or duplicate system residence file

PRV -- For any private libraries.

For example, the statement MERGE RES,PRV indicates to the CORGZ program that elements are to be transferred from one or more libraries on the system residence file to the corresponding private libraries. The type of library involved and the elements to be transferred are specified in COPY statements immediately following the MERGE statement. (The COPY statements are the same as those described in the preceding section *Creating a New System Residence*).

You must define the extents of the libraries involved in a merge operation by DLBL and EXTENT job control statements. The filenames to be used and the necessary symbolic unit assignments are described in detail in *DOS/VS System Control Statements.*

Note that, when the CORGZ program performs a merge operation, it *does not* automatically copy the basic system components as it does when a new system residence is created (see preceding section). You must specify COPYC ALL to transfer the entire core image library or COPY ALL to transfer the entire SYSRES extent. Moreover, when the merge function is being performed, you cannot reallocate the libraries with an ALLOC statement.

The job in the following example adds the contents of the core image library on a duplicate SYSRES file (NRS) to the elements in a private core image library (PRV). Any elements with duplicate names (supervisor, job control etc.) are deleted from the receiving library.

```
// ASSGN SYS002,X'130'
// DLBL IJSYSRS,'DOS/VS SYSRES II',99/365,SD
// EXTENT SYS002,111111,1,0,0001,2519
// DLBL IJSYSCL,'PRIVATE CIL',99/365,SD
// EXTENT SYSCLB,222222,1,0,1600,200
   ASSGN SYSCLB,X'131'
// EXEC CORGZ
   MERGE NRS,PRV
   COPYC ALL
/*
/&
```

Alternatively, for the COPYC, COPYR, COPYS, and COPYP statements, the NEW operand can be used to copy only those members that do not already exist in the receiving library. Note, however, that for COPYC NEW:'

- supervisor phases are never copied, and
- a number of system phases are always copied.

See *DOS/VS System Control Statements* for a list of these phases. In addition, you must ensure that your receiving library has sufficient space allocated to accommodate the library members that are copied from the other library.

The job in the following example also adds the contents of the core image library on a duplicate SYSRES file (NRS) to the elements in a private core image library (PRV). In this example, only nonduplicate elements are copied.

```
// JOB NRSPRV
// ASSGN SYS002,X'130'
// DLBL IJSYSRS,'DOS/VS SYSRES II',99/365,SD
// EXTENT SYS002,111111,1,0,0001,2519
// DLBL IJSYSCL,'PRIVATE CIL',99/365,SD
// EXTENT SYSCLB,222222,1,0,1600,200
ASSGN SYSCLB,X'131'
// EXEC CORGZ
   MERGE NRS,PRV
   COPYC NEW
/*
/&
```

Using the Service Functions of the Librarian

The service functions of the librarian enable you

- to obtain reports on the contents of your libraries by displaying the directories on SYSLST
- to print and/or punch the contents of your libraries on SYSLST or SYSPCH in order to transfer the library elements to a different location or to correct them
- to prepare macro definitions in the assembler macro (E) sublibrary for update.

The directories are displayed by the DSERV program. Edited macros in the E-sublibrary can be de-edited for update by the ESERV program. To print or punch the contents of the libraries, a separate program is available for each type of library:

CSERV -- Core image library
RSERV -- Relocatable library
SSERV -- Source statement library
PSERV -- Procedure library

If you use private libraries, the service functions apply only to the private libraries assigned. Private libraries must be unassigned before the corresponding system libraries can be accessed by the service programs.

Displaying the Directories

Using the directory service program (DSERV) you can obtain a listing of the following directories:

- Core image directory, or the directory entry of a specific phase or group of phases (transients, for instance) in the core image library together with their change level, if present
- System directory list (SDL)
- Relocatable directory
- Source statement directory
- Procedure directory
- System directory. This directory is always listed before any of the directories is printed. This information is called a status report.

Depending on the control statement used, the directories can be displayed in one of two formats:

- An alphamerically sorted listing of the directory entries (DSPLYS control statement)
- A listing of the entries in the order in which they appear in the directory (DSPLY control statement).

Note: *The entries in the core image directory are always displayed in alphameric sequence.*

Within a single job step you can obtain multiple displays of the same directory, either sorted or unsorted, by supplying a separate control statement for each desired display. Similarly, any number of directories can be displayed within one job step, depending on the operands in the control statement. The following job will produce a sorted listing of all transients ($-phases) and unsorted listings of the relocatable and source statement libraries:

```
// JOB DISPDIR
// EXEC DSERV
   DSPLYS TD
   DSPLY RD,SD
/*
/&
```

If you specify // EXEC DSERV without any control statements, a status report of all libraries present on SYSRES and all private libraries assigned (if any) is printed on SYSLST.

Displaying and Punching the Contents of the Libraries

You can use the library service programs to obtain a listing, a card deck, or a card image copy of the elements in a library. There is a service program for each library:

CSERV -- Core image library
RSERV -- Relocatable library
SSERV -- Source statement library
PSERV -- Procedure library.

You request the library service functions by means of three control statements which are used for all four library service programs. These control statements are:

DSPLY -- To print the elements of a library
PUNCH -- To punch the elements of a library
DSPCH -- To print and punch the elements of a library.

Each of these statements can specify one or more individual elements, one or more groups of elements, or all elements of a library to be printed or punched. The following job prints the entire sublibrary P and punches phases PHAS1 and PHAS3 of the core image library:

```
// JOB LIBSERV
// EXEC SSERV
   DSPLY P.ALL
/*
// EXEC CSERV
   PUNCH PHAS1,PHAS3
/*
/&
```

The punched output (either in cards or on tape, diskette, or disk) of any service program can be used as input for recataloging into the type of library from which it was extracted. Except for the CSERV punched output, the service programs automatically punch a CATALR, CATALS, or CATALP statement immediately preceding each element, and a /* statement (/+ in case of the procedure library) immediately following the last element. Such output can therefore be submitted with a // EXEC MAINT statement for recataloging.

Punched output of the CSERV program is suitable for input to the linkage editor for recataloging to the core image library. The control statement stream would be as follows:

```
// JOB RECATAL
// OPTION CATAL
   INCLUDE


/*
// EXEC LNKEDT
/&
```

Phases punched from the core image library are relocatable if ACTION REL was active when the phases were originally cataloged. If relocatable phases are recataloged, their origin is at an address relative to the end of the supervisor (S+displacement). If nonrelocatable phases are recataloged, their origin is at the same absolute address as when they were originally link-edited.

Phases originally cataloged with the SVA operand are punched and displayed with this indication.

Printed output from any of the service programs is useful for debugging purposes. For instance, after determining an error from a dump or source listing, you implement a change to the RSERV object deck by inserting the appropriate REP card(s) directly before the END card and run the MAINT program to recatalog the object module; then to verify that the REP card was correct, execute the RSERV program to obtain a listing. An SSERV listing may be necessary before a single statement update can be performed; after locating the statement in error in the listing, submit an UPDATE maintenance run to implement the change in the source statement library.

Preparing Edited Macros for Update

The assembler uses two sublibraries of the source statement library: the macro sublibrary (sublibrary E) and the copy sublibrary (sublibrary A). All macro definitions in the assembler macro (E) sublibrary have been preprocessed by the assembler; they are said to be edited. An edited macro definition cannot be directly updated; instead, the source macro, either in a card deck or in the copy (A) sublibrary is updated. After the changed macro has been tested and debugged, it must be edited again before it can be recataloged in the macro sublibrary.

If the macro to be updated is not available in source format, you can use the ESERV program to convert the edited macro back to source

format: this is called de-editing. If the output of the ESERV program is to be used directly as input to the assembler, you can specify the GENEND control statement to cause the END card and a /* card to be included after the last macro. If the output is to be cataloged directly into the copy (A) sublibrary, you can specify the GENCATALS control statement. This causes a CATALS card to be generated before each macro in the run and a /* card after the last macro. If neither the GENEND nor the GENCATALS control statement is specified after the // EXEC ESERV statement, GENCATALS is assumed.

The remainder of the control statements that you submit to the ESERV program are the same as for the other librarian service programs: DSPLY, PUNCH, and DSPCH. The following job de-edits the macro named MAC1:

```
// JOB DEEDIT
// EXEC ESERV
   GENEND
   PUNCH E.MAC1
/*
/&
```

The output of the above job is the macro MAC1 in source format on SYSPCH. An END card and a /* card is included after the macro. You can now update the macro, edit it, and catalog it back into the E sublibrary of the source statement library.

You can de-edit and update a macro in a single run by submitting the necessary update control statements. The following job de-edits and updates the macro MAC2. The result will be the updated macro in source format on SYSPCH and a listing of the updated macro on SYSLST:

```
// JOB EDTUPDTE
// EXEC ESERV
   GENCATALS
   DSPCH E.MAC2
   .
   .
   update control statements
   .
   .
/*
/&
```

The update function of the librarian is described in *Updating the Source Statement Library,* earlier in this chapter. Detailed information on editing, de-editing, and updating macro definitions is given in *Guide to the DOS/VS Assembler.*

Creating and Working with Private Libraries

Private libraries are created and maintained by the system librarian programs. All librarian functions are performed in the same manner for private libraries as for system libraries. The reallocate (ALLOC) function is the only one not available to private libraries. To change the extents of a private library you must create a new private library and copy the contents of the old library into it.

The following sections describe how to create private libraries and what you must consider when you use private libraries.

Creating Private Libraries

You can create private libraries either during system generation or at any time thereafter. Private libraries can reside on the SYSRES pack (outside the SYSRES extent) or on separate disk packs which (except for a private core image library) must be of the same device type as the SYSRES pack. You can define any number of private core image, relocatable, and source statement library; private procedure libraries are not supported.

You create private libraries with the CORGZ librarian program. The creation of an operational private library involves two stages:

1. Defining the extents of the library by means of a NEWVOL (new volume) control statement.
2. Transferring information to the library from an existing library by means of COPY and/or MERGE control statements.

You can execute the two stages either in one job step by one invocation of the CORGZ program or in separate job steps. Exception: creation of a private core image library requires separate job steps.

To define the device on which a private library is to be created and the disk extents occupied by the library, you must supply a set of ASSGN, DLBL, and EXTENT job control statements specifying predetermined symbolic unit names and filenames (see Figure 7.7).

Private Library	Symbolic Unit Name	Filename
Core image	SYS003	IJSYSPC
Relocatable	SYSRLB	IJSYSRL
Source statement	SYSSLB	IJSYSSL

Figure 7.7. Symbolic Unit Names and Filenames Required to Create Private Libraries

You can store the label information submitted by DLBL and EXTENT statements either temporarily (option USRLABEL) or permanently (option PARSTD or STDLABEL). Temporary labels must be resubmitted with every job (or job step, if new labels are submitted in an intermediate job step) that accesses the corresponding library; permanent labels are valid for all subsequent jobs.

Note: *If you catalog additional permanent labels with the STDLABEL option you must also resubmit all existing standard labels; otherwise, they are lost (see also Types of Label Information in Chapter 5: Controlling Jobs).*

The following example shows the job control and librarian control statements necessary to define the extents of a private relocatable and a private source statement library. The NEWVOL control statement indicates the type of library to be created and the number of cylinders (tracks) to be allocated to each library (directory).

```
// JOB DEFINE
// ASSGN SYSRLB,X'191'
// ASSGN SYSSLB,X'192'
// DLBL IJSYSRL,'DOS/VS PRIVATE RL',99/365,SD
// EXTENT SYSRLB,111111,1,0,20,800
// DLBL IJSYSSL,'DOS/VS PRIVATE SSL',99/365,SD
// EXTENT SYSSLB,222222,1,0,500,600
// EXEC CORGZ
   NEWVOL RL=40(5),SL=30(5)
/*
/&
```

After you have defined the extents of the private libraries you can either use the merge function of the CORGZ program to transfer elements from existing libraries or the catalog function of the MAINT program to store new elements.

To create a private library and at the same time copy information into it from the corresponding system library, you submit a COPY statement following the NEWVOL statement. To transfer information from an existing private library, a MERGE statement must precede the COPY statement. The following job creates a private relocatable library and copies into it the contents of the system relocatable library and of an existing private relocatable library:

```
// JOB CREATE
// ASSGN SYSRLB,X'191'
// ASSGN SYS001,X'192'
// DLBL IJSYSRL,'NEW PRIVATE RL',99/365,SD
// EXTENT SYSRLB,111111,1,0,1700,1200
// DLBL IJSYSPR,'OLD PRIVATE RL',99/365,SD
// EXTENT SYS001,222222,1,0,700,400
// EXEC CORGZ
   NEWVOL RL=60(8)
   MERGE PRV,PRV
   COPYR ALL
/*
/&
```

Note: *To merge from a private relocatable library, you must assign SYS001 to the device containing the library and specify the filename IJSYSPR in the DLBL statement. The logical unit assignments and filenames required for the various merge operations are described in DOS/VS System Control Statements.*

Creating Private Core Image Libraries

The organization of a private core image library is the same as the system core image library. A private core image library, however, may start on any track. The space requirements must be entered in the NEWVOL statement.

For example, on a 2314 device, the statement NEWVOL CL=14(5) creates a directory of five tracks and a library of 14 cylinders. To create this private core image library on a 2314 device starting at relative track number 120, you submit the following control statements:

```
// JOB PCIL
// ASSGN SYS003,X'191'
// DLBL IJSYSPC,'DOS/VS PRIVATE CL',99/365,SD
// EXTENT SYS003,111111,1,0,0120,280
// EXEC CORGZ
```

```
   NEWVOL CL=14(5)
/*
/&
```

In the above example, the core image directory resides on cylinder 6 (tracks 0-4), and the private core image library on cylinders 6-19.

If you desire to start a private core image library in the same relative location as the system core image library (that is, the library directory starting at cylinder 0 track 2), the relative track specification in the EXTENT statement must be 0002. The EXTENT statement in the preceding example then reads:

```
// EXTENT SYS003,111111,1,0,0002,280
```

Transferring phases from another core image library would require a second job step.

Using Private Libraries

To access the private libraries, you must assign the following symbolic unit names to the device(s) containing the libraries:

SYSCLB -- Private core image library
SYSRLB -- Private relocatable library
SYSSLB -- Private source library

Note that the symbolic unit name required to *create* a private core image library is SYS003; for private relocatable and source statement libraries, however, the symbolic unit names are the same for creation and subsequent access.

You can assign private relocatable libraries and private source statement libraries either temporarily or permanently by an ASSGN command or statement; you can assign private core image libraries only by an ASSGN command (that is, permanently). You cannot establish standard assignments for private core image libraries with the ASSGN macro during supervisor generation.

Unless you have cataloged standard labels for your private libraries, you must submit label statements with every job that accesses the libraries. The filenames and file identifications in the DLBL statements must be identical to those specified when the libraries were created (except for a private core image library, where the filename IJSYSPC is used for creation, and IJSYSCL is used thereafter).

A private library must be unassigned if maintenance and service functions are to be performed on the corresponding system library. The librarian programs assume that the private library is intended whenever assigned. So if, by mistake, your private relocatable library is assigned when you request changes in the system relocatable library, these changes will be performed on the private relocatable library and reconstruction of this library may be necessary, depending on the nature of the changes. The only system service programs that can access the system libraries when SYSRLB and SYSSLB are assigned are the linkage editor and the CORGZ librarian program.

You can have an unlimited number of private libraries in your system; however, no more than one private core image, one private relocatable, and one private source statement library can be assigned at one time to the same partition. For read access you can also assign a private library to more than one partition, but if you want to update a private library, it must be assigned to one partition only (see Figure 7.8).

If you have more than one private library of the same type, each must be distinguished by a unique file identification in the DLBL statement for the library.

Using Private Core Image Libraries

Private core image libraries provide an efficient multiprogramming environment. The linkage editor can be executed not only in the background but also in a foreground partition to which a private core image library is assigned. You can then link-edit a program in any given partition to be executed in the same or in a different partition. If the linkage editor is executed in more than one partition at the same time, you must assign a separate SYSLNK and SYS001 file for each of these partitions.

A separate private core image library can be defined for each partition. Such a private core image library is then said to be *dedicated* to a given partition. Separate versions of the same non-self-relocating program may be link-edited for execution in each partition. This is not necessary, however, for relocatable phases, when the system includes support for the relocating loader.

If you work with the relocating loader, private core image libraries are nevertheless useful to hold special-purpose programs. This allows, for instance, a new version of a program to be tested while the original version remains in working order on the system core image library.

A private core image library should not be assigned to more than one partition at the same time if the linkage editor is being executed in one of these partitions. If this occurs, the linkage editor issues a message and terminates abnormally.

Output from the linkage editor is, therefore, placed in a private core image library only if it is uniquely assigned to the partition where the linkage editor is executed. When fetching or loading a phase, the system first searches the private core image library, if assigned, and if the phase is not found, the search is continued in the system core image library. For phases starting with $, first the system and then the assigned private core image library is searched. This library search sequence, particularly important in teleprocessing and VSAM applications, should be considered when determing names and library residence of programs.

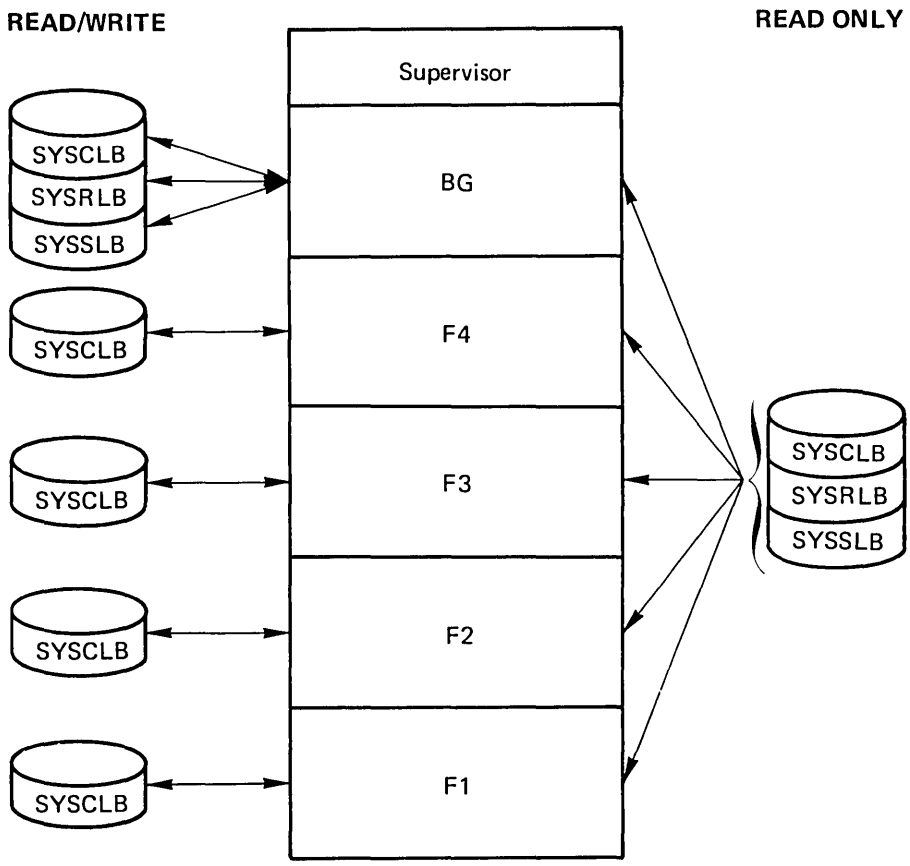


Figure 7.8. Possible Assignments of Private Libraries in a Multiprogramming System

The example assumes a five-partition system. For update access, a private relocatable or source statement library must be assigned to the background partition (see Figure 7.3 for ESERV and CORGZ exceptions).

Chapter 8: Using POWER/VS

The POWER/VS information has been removed from this manual and is now contained in: *DOS/VS POWER/VS Installation Guide and Reference*, GC33-6048.

Part III: Designing Programs

This section addresses the system programmer and application programmer. It gives some programming considerations for designing virtual-mode programs and shows how to use many of the macros and special features of DOS/VS. This section consists of two chapters:

Chapter 9: Designing Programs for Virtual-Mode Execution provides considerations for designing programs and using the macros especially provided for the virtual-mode environment. This chapter also describes how to code for the shared virtual area and the programming conventions for a POWER/VS user exit routine.

Chapter 10: Using the Facilities and Options of DOS/VS describes how user programs can communicate with one another and with the system. This chapter discusses how programs can take advantage of user exit routines, the time-of-day clock support, cancel and checkpoint services, and job accounting interface.

CAUTION: If a program (or routine) written in assembler language reads or manipulates DOS/VS control information through code other than IBM provided macros, this program may be executed successfully under the DOS/VS release with which the program was tested. The program may fail, however, when it is executed under a later DOS/VS release.

Chapter 9: Designing Programs for Virtual-Mode Execution

This chapter addresses system programmers and application programmers who are concerned with designing programs for the DOS/VS environment. This chapter contains information that may improve the efficiency of those programs that exceed the amount of real storage available to them at any one time. It is recommended that these techniques be considered as new programs are written and as old programs are revised. The chapter also contains information on the use of certain assembler language macro instructions that are provided especially for virtual storage. Programming conventions for the shared virtual area and a POWER/VS user exit routine are also discussed.

Programming Hints for Reducing Page Faults

It is desirable to spend some extra programming effort to *tune* virtual-mode programs that are used frequently or that require long periods of processing time so that they will cause fewer page faults during execution. Page faults generally occur when the size of the virtual-mode program exceeds the number of page frames available to it during execution. Efforts to reduce the number of page faults occuring in a program generally center around efforts to reduce the size of the *working set* of the program. The term *working set* is one that recurs often in discussions of virtual storage systems.

The working set of a program is the minimum number of pages (not specific pages) which must be in real storage in order for a program to execute efficiently. In other words, the working set of a program is the minimum number of page frames that the program requires for efficient execution. The supervisor determines which specific pages should be in real storage at any particular time.

What does *execute efficiently* mean? Essentially, this means that a program will not execute appreciably slower than if the entire program were in real storage during its entire execution.

Although the following section does not tell you how to determine the size of the working set, it does provide techniques for reducing its size.

General Hints for Reducing the Working Set

You should especially try to reduce the size of the working set of programs that you use frequently or that execute for long periods of time. Your programming efforts are more worthwhile for such programs than for relatively short and less frequently-used programs.

There are three general rules to keep in mind when working to reduce the working set. The first is *locality of reference,* that is, instructions and data used together should be in storage near each other. Second is *minimum real storage.* In other words, the amount of real storage

necessary for a program to do something should be kept as low as possible. Third is *validity of reference,* that is, references should be made only to data which will actually be used.

The chief means of achieving locality of reference is to make execution sequential whenever possible, by avoiding excessive branching.

A program that executes sequentially normally requires a partition larger than the same program when it does not execute sequentially. For example, the functions of a section of code repeat themselves several times throughout the logic of your program. You are tempted to write this code once and branch to it whenever necessary, but branching violates the principle of locality of reference. Branching may cause more page faults the program incurs than would coding the routine in line each time it is used. Also, it is easier for someone else to follow the logic of a program which is written to execute sequentially.

Locality of reference can be achieved only to a limited extent by programs written in a high-level language.

Elements in arrays in FORTRAN or PL/I can be referred to in the order in which they appear in storage. In FORTRAN, for example, arrays are ordered by columns. The elements of the array DIMENSION (2,2,2) are arranged as follows in contiguous virtual storage locations:

(1,1,1)
(2,1,1)
(1,2,1)
(2,2,1)
(1,1,2)
(2,1,2)
(1,2,2)
(2,2,2)

For array structures of other compilers, refer to the appropriate programming language reference manuals.

A routine which processes all the elements of such an array should refer to them in this order. If only certain elements of an array are processed, the elements should be arranged in the order in which they are to be processed. If arranging an array in a certain manner causes it to be processed advantageously one time, but disadvantageously another time, you should consider writing two arrays, even at the cost of additional virtual storage.

Another good practice to help reduce paging is to not initialize variables until just before they are to be used. For example in PL/I instead of the following:

```
DCL A FIXED INIT (10);
.
.
DO B=1 TO 100;
A=A+B;
END;
```

use:

```
DCL A FIXED;
.
.
A=10;
DO B=1 TO 100;
A=A+B;
END;
```

In the first method of coding, PL/I initializes the automatic variable at the beginning of execution. The second method of coding does not require the page containing A to be in real storage until just before A is used.

An important help in reducing the amount of real storage needed for execution is to remove coding which is used for errors or other unusual occurrences. If, for example, the main routine contains code for conditions that only occur 5% of the time, by removing this error code and making it into a separate section of code you can reduce the amount of real storage necessary for 95% of the processing.

Frequently-used subroutines should be loaded near each other. Because of their frequent use, these routines tend to be in real storage almost continuously. If they are scattered over several pages, each of these pages will need to be in real storage most of the time, thus increasing the size of the working set. By loading these routines near each other, you reduce the number of pages required in real storage at any one time.

Subroutines should be designed to do as much processing as possible whenever they are called. It is better to duplicate some code from the calling routine in the called routine in order to avoid switching back and forth between routines. One technique for accomplishing this is to have the calling program pass several parameters to the subroutine each time a call is made, rather than passing one parameter at a time and making several calls.

Data and Constants in Assembler Language Programs

You should keep frequently used data and constants near each other in storage, and near the instructions which use them. This contrasts with the traditional practice of having one area at the end of the program reserved for all the data areas and constants. By the same token, seldom used data should be separated from the frequently used data and placed with the routines which use it.

Avoid, if possible, using chains which must be searched each time a data item is required. If chains are unavoidable they should be kept in a compact area of storage. This may result in some wasted storage but will be better than searches of large areas of storage.

You should try to keep code that can be modified and code that cannot be modified in separate sections of a large program. This will reduce page traffic by reducing the number of pages that are changed. Also, try to prevent I/O buffers from crossing page boundaries unnecessarily. Check the assembler listing and the linkage editor map to determine where 2K boundaries occur in your programs.

Using Virtual Storage Macros

The macros designed for use by virtual-mode programs, which are discussed in this section, perform the following services:

- influence the paging mechanism in order to reduce the number of page faults, to minimize the page I/O activity, and to control the page traffic within a specific partition.
- 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).

In order to use these macros you must be programming in assembler language or, if your program is written in a high-level language, you must write an assembler subroutine to accommodate them. Refer to *DOS/VS Supervisor and I/O Macros* for a complete description of the formats of these macros.

Fixing Pages in Real Storage

In DOS/VS parts of virtual-mode programs must be in real storage only at certain times. These parts include not only the instructions and data being processed at any one moment by the CPU, but also data areas for use by channel programs. Instructions and data are always in real storage when being used. Because of the nature of I/O operations, the data areas for these operations could be paged out during the I/O operation if something were not done to keep them in real storage during the entire operation. The DOS/VS supervisor fixes I/O areas in real storage for the duration of the I/O operation.

There are other parts of a program, however, which cannot tolerate paging, and these parts are not necessarily kept in storage by the system. For instance, I/O appendages and programs that control time-dependent I/O operations cannot tolerate paging. A familiar example of the latter is a MICR (Magnetic Ink Character Reader) stacker select routine. If a page fault were to occur during the execution of one of these programs, the results would be unpredictable. A page fault in one of these programs can be avoided by fixing the affected pages in real storage (using the PFIX macro).

The supervisor fixes pages for I/O operations temporarily anywhere in the page pool. The pages that you fix by the PFIX macro, however, are fixed in the storage allocated to the corresponding real partition. Only as many pages may be fixed by a program at any one time as there are page frames in the corresponding real partition. This is done to prevent a loop in one program from fixing all the pages in the system, and to enable other programs to issue a PFIX macro concurrently.

The PFIX macro fixes the pages in real storage, regardless of whether these pages are stored in contiguous page frames or not. The supervisor keeps a count of the number of times a page has been fixed without being freed. A page that is fixed more than once without having been freed (via the PFREE macro) is not brought in a second time and given another page frame. Instead, the counter for that page is just increased by one and the page remains in the same page frame. If more than 255 PFIX requests were issued for the same page (without having issued PFREE requests in the meantime), the issuing task is canceled.

The PFREE macro does not directly free a page for paging out, but each time it is issued, the counter of fixes is reduced by one. As soon as the counter for a page reaches zero, the page can be paged out. At the end of a job step, all pages that have been fixed during the job step are freed. The PFREE macro should be used as soon as possible to make the page frames available to all programs running in virtual mode.

Figure 9.1 is an example using the PFIX and PFREE macros. After the execution of a PFIX macro, a return code is given in register 15. The meanings of the return codes are:

0 - The pages were fixed successfully.

4 - You requested more page frames than can be contained in a real partition of the size you are working in.

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.

```
            .
            .
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
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 9.1. PFIX and PFREE Example

Determining the Execution Mode of a Program

You may have a program that must do different processing depending upon what its execution mode is. It may be impractical to have two separate programs cataloged in the core image library, one program for real mode and another program for virtual mode. The RUNMODE macro can be issued during the execution of the program to inquire which mode of execution is being used. A return code is issued to the program in register 1.

Releasing Pages

With the RELPAG macro, you inform the page management routines that the contents of one or more pages is no longer required and need not be saved on the page data set. Thus, page frames occupied by these released pages can be claimed for use by other pages, and page I/O activity is reduced.

Forcing Page-out

The FCEPGOUT macro is used to inform the page management routines that one or more pages will not be needed until a later stage of processing. The pages are given the highest page-out priority, with the result that other pages, which may be needed immediately, are kept in storage. Except when the RELPAG macro is in operation, the contents of any pages written out are saved.

Advancing Page-in

The PAGEIN macro allows you to request that one or more pages be paged in in advance, in order to avoid page faults when the specified pages are needed in real storage. If the specified pages are already in real storage when the macro is issued, they are given the lowest priority for page-out.

Balancing Teleprocessing

The TPIN macro signals the DOS/VS supervisor that an immediate demand for system resources is to be made by the teleprocessing application, for instance, when a message has arrived. After processing is completed, TPOUT informs DOS/VS that the teleprocessing application has no further processing to do for the time being, and that the system resources that were exclusively used for teleprocessing should be released. Failure to issue the TPOUT macro can cause serious performance degradation in batch processing.

It is not recommended that you use TPIN/TPOUT macros in your teleprocessing application programs. Use them instead in the telecommunications access methods and data base/data communication interface programs such as the IBM program product CICS/VS. The latter, when running under DOS/VS, supports the TPIN/TPOUT interface with the supervisor. Refer to *DOS/VS Supervisor and I/O Macros* for further details.

Coding for the Shared Virtual Area

Besides accommodating the system directory list (SDL), and perhaps the VSAM phases with their associated GETVIS work area, the shared virtual area (SVA) contains phases that can be used concurrently by more than one partition. The SVA phases must be fully reenterable and relocatable; code that modifies itself will cause a protection check when executed from the SVA. This section presents some advice on coding phases to use SVA facilities and suggests some standards for base-register usage.

The basic assumptions for coding an SVA phase are:

- The reenterable code must not modify any storage within its own storage area.
- The phase can modify registers only if it saves and restores them for each user.
- A user-specified work area (within the calling partition) must be provided for storing registers and for any storage modifications.

Suggested register conventions:

- Use register 12 as the base register in both the main routine and the reenterable code.
- Use register 13 as base for the working storage area. It is the responsibility of the main routine to provide addressability to the work area by loading register 13; the reenterable routine must not modify register 13. The easiest way to address the working storage area in the reenterable code is by a DSECT that defines the fields of the work area and a USING DSECTNAME,13. In this way symbolic addressing can be used.
- Use CALL, SAVE, and RETURN macros. Since register 13 is the base register, SAVE (14,12) and RETURN (14,12) result. Use register notation for CALL, for example, CALL (15) .... Before issuing the CALL, load register 15 with the transfer address. Register 14 will always contain the return address. The standard is thus established of register 15 for calling and register 14 for returning.
- Switches, and other areas that may be modified, can be placed in the working storage area using base register 13.

Figure 9.2 illustrates the suggested conventions: MASTER is the main routine, SLAVE is the SVA phase contained in the SDL.

```
MASTER    CSECT
          BALR   BASE,0
          USING  *,BASE
          LA     13,SAVE
          LOAD   SLAVE,WORKAREA+2      CANCELS IF SLAVE NOT IN CIL
          LR     15,1
          CALL   (15),(SWITCH,TECB,FIELDA,FIELDB,WORKAREA)
          .
          .
          EOJ
SAVE      DS     9D
WORKAREA  DS     200D
SWITCH    DC     XL1'00'
TECB      DS     CL4
FIELDA    DS     CL15
FIELDB    DS     CL11
          END

SLAVE     CSECT            MUST BE SEPARATE ASSEMBLY
          SAVE   (14,12)
          BALR   BASE,0
          USING  *,BASE
          USING  WORKAREA,6
          LM     2,6,0(1)
          MVC    0(15,4),DATA1
          MVC    0(11,5),DATA2
          CLI    0(2),X'FF'
          BE     EXIT
          SETIME 3,(3)          SETIME ALTERS THE TECB
          WAIT   (3)
          .
          .

EXIT      XI     0(2),X'FF'
          RETURN (14,12)
DATA1     DC     CL15'THIS IS FIELDA'
DATA2     DC     CL11'THIS IS FIELDB'
          LTORG
WORKAREA  DSECT
FIELDC    DS     3D
FIELDD    DS     3D
          END
```

Figure 9.2. Example of Conventions for SVA Coding

Chapter 10: Using the Facilities and Options of DOS/VS

DOS/VS provides a variety of standard and optional services for programs to communicate with each other, with one or more systems, and with the operator. The most prominent of these are:

- Direct linkage between programs
- Timing features
- Linkages to user exit routines
- Checkpointing facility
- Job accounting interface feature
- Storage dump facility
- DASD switching

Judicious use of these services enhances the benefits to be obtained from computer operations.

Direct Linkage between Programs

Any user phase or routine can communicate with another phase or routine in the same partition by direct linkage, and, in multitasking (asynchronous processing), the main task and subtasks within a partition can communicate with each other.

For efficient virtual mode processing under DOS/VS with multiprogramming support, a modular program structure is recommended. Ideally, within a module the instructions should be sequential.

Sequential execution of instructions moderates paging activity necessary for the programs to proceed and thus promotes system throughput.

Interlanguage Communications

Every programming language provides for communicating and passing control between modules written in the same language or in Assembler language. Communication is also possible between any modules written in languages that use compatible linkage conventions. Transferring data between high-level languages is usually difficult, however, because of differences in data formats and storage allocations.

The PL/1 optimizing compiler (an IBM program product) provides for communication between programs written in PL/1 and others written in COBOL or FORTRAN.

User Program Switch Indicators (UPSI)

A user program switch in the partition communication region of the supervisor can be used to execute a special routine in a program, or to cause a module to call another module for special processing. A typical application enables a program that regularly processes certain standard data to do some special processing periodically. The special processing routine

can be entered by using a program switch that is set by the UPSI job control statement as illustrated by the assembler language example in Figure 10.1.

```
// UPSI     00000001       SET SWITCH
   .
   .
   .
   COMRG                   GET COMRG ADDRESS INTO REG 1
   TM       23(R1),X'FF'   TEST UPSI FOR ANY BIT SET ON
   BNZ      SPECIAL        IF NO BIT ON NORMAL PROCESSING
   .
   .
SPECIAL
```

Figure 10.1. Setting and Testing UPSI

Note that the UPSI job control statement is included only when special processing is required. For optimal processing efficiency, the type of routine entered at the label SPECIAL depends on the amount of special processing and on what options the system supports. It could be the special processing routine directly or it could be a routine to load and enter a new phase or, in multitasking, a routine to attach a subtask.

Also, in this example, without the UPSI job control statement the special routine will never be entered because the UPSI byte is set to all zeros when a JOB or / & statement is encountered, but the special routine will always be entered when *any* UPSI bit is set to 1 by an UPSI statement.

Timing Features

DOS/VS provides optional timing facilities which use hardware features that are standard in any System/370 CPU. (Note):

1. The time-of-day (TOD) clock is used to determine the current time.
2. The interval timer (IT) which enables a time interval in seconds on 1/300ths of a second to be preset so that a program can be notified when the time interval has expired.
3. The task timer (TT) which allows a timer interval in milliseconds to be preset by a task so that the task can give control to an exit routine when the specified time interval has elapsed.

Note: *For hardware feature requirements, refer to Timer Services in chapter 3.*

Using the Time of Day Clock

The time-of-day (TOD) clock is a standard high-resolution System/370 hardware feature. Any program executing under DOS/VS can obtain the time of day. Two methods are available, the first of which requires the optional supervisor support for the GETIME macro (TOD=YES specified in the FOPT macro at system generation time). The methods are:

1. Issue a GETIME macro. This returns the time of day in hours, minutes, and seconds, or as a binary integer value in seconds, or as a binary integer in units of 1/300 seconds, depending on the optional operand specified. For details of this method, refer to *DOS/VS Supervisor and I/O Macros*.

2. Issue a STCK instruction. This stores the high-resolution time of day value at a specified address in the program's partition. A very accurate real-time interval measurement is facilitated by issuing this instruction at the beginning and again at the end of a routine with all pages of the routine (including the STCK instructions), and all pages containing referenced addresses, being previously fixed in real storage. Any interrupt that occurs during an interval is included in the measurement.

Figure 10.2 illustrates the use of the STCK instruction and a typical routine to calculate the time interval.

```
         STCK  START         STORE THE STARTING TIME
BEGIN    (Routine to be timed)
         .
         .
         .
         STCK  FINISH        STORE THE FINISHING TIME
         BR    R14           RETURN TO NORMAL PROCESSING

* TIMER ROUTINE
TIME     LM    R2,R3,FINISH GET FINISHING TIME
         SL    R3,START+4    SUBTRACT RIGHT-HAND HALVES
         BC    3,SUBLEFT     BRANCH IF CARRY
         BCTR  R2,0          SUBTRACT ONE
SUBLEFT  S     R2,START      SUBTRACT LEFT-HAND HALVES
         SRDL  R2,12         SHIFT TO GET MICROSECONDS
         STM   R2,R3,TIMEINT  SAVE THE TIME INTERVAL
         .
         .
         .
END      EOJ
START    DS    D
FINISH   DS    D
TIMEINT  DS    D
         END
```

Figure 10.2. Method for Accurate Measurement of a Real Time Interval

Interval Timer

Interval timer support may be generated optionally for all programs (including subtasks if multitasking is supported) in *all* partitions.

Any program (or task) can set a real time interval, in seconds, or 1/300ths of a second, by issuing a SETIME macro. Expiration of the specified interval causes an external interrupt. The maximum valid interval is 55918 seconds (15 hours, 31 minutes, and 58 seconds). When the interrupt occurs, the program that issued the SETIME macro may continue processing, another task may be given control if it was waiting on the same event and has higher priority, or a special user routine may be entered if linkage has been established by a STXIT IT macro. If no task is waiting on the event and no linkage has been established, the interrupt is ignored.

Waiting for a Time Interval to Elapse

When processing is dependent on the expiration of a time interval, a WAIT macro will suspend processing until the interval set by a SETIME macro has elapsed.

The SETIME macro passes to the supervisor the name of the timer event control block (TECB) to be posted when the specified interval has elapsed. The WAIT macro specifies the same TECB and passes control to the supervisor which, in a multiprogramming environment, allows a task in another partition to execute in the meantime. When the timer interrupt occurs, the event bit in the TECB is turned on and any task that has issued a WAIT macro specifying this same TECB is made ready to proceed; if more than one task, then the task having the highest priority is dispatched. Figure 10.3 illustrates a program that waits for a time interval to expire.

```
         START 0
         .
         .
TECB1    TECB
         .
         .
STIMER   SETIME 30,TECB1      START 30 SECOND INTERVAL
         .
         .
(normal processing not time-dependent)
         .
         .
         WAIT TECB1           WAIT FOR TIMER END
         .
(time-dependent processing)
         .
         .
         END
```

Figure 10.3. Skeleton Example of a Program in which a 30-second Interval Must Elapse before Special Processing is Performed

Getting the Unexpired Time

After a SETIME macro has been issued, any program or task executing in the same partition can obtain the unexpended part of the interval by issuing a TTIMER macro. This macro returns the residual time (seconds) without disturbing the interval timer function.

If the TTIMER macro includes the operand CANCEL, a previously issued SETIME macro is canceled.

Task Timer

The task timer support can be generated only for the main task of a specific partition.

The main task sets the desired time interval by specifying it, in milliseconds, in the operand of the SETT macro; or by putting the desired interval, in milliseconds, in binary, in the register specified as the operand

of the SETT macro. The maximum valid interval is 21,474,836 milliseconds.

When the specified time interval has elapsed, the task timer routine supplied in the STXIT TT macro is entered. If a routine was not supplied to the supervisor by the time the interrupt occurs, the interrupt is ignored.

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

Obtaining or Canceling the Time Remaining

The task using the task timer can issue a TESTT macro to test how much time remains in the time interval set by an associated SETT macro. The time remaining in the interval is returned, expressed in hundredths of milliseconds - in binary, in register 0.

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

Linkages to User Exit Routines*

Through the STXIT macro instruction, linkage can be established to one or more user routines if the appropriate FOPT macro parameter was specified to generate the support in the supervisor.

The first operand of a STXIT macro instruction informs the supervisor where to store the special routine entry point address that is specified by the second operand. When the specific condition arises, the supervisor passes control by entering the routine at that address. The conditions, STXIT macro first operands, and the special user-written routines entered, are shown in the following table:

Condition	STXIT Operand	User Routine
Interval Timer External Interrupt	IT	Interval Timer Exit
Task Timer Interrupt	TT	Task Timer Exit
Abnormal Termination of Problem Program	AB	Abnormal Termination Exit
Program Check Interrupt	PC	Program Check Exit
Operator Communications Interrupt	OC	Operator Communications Exit

Interval Timer User Exit Routine

If special processing is required when a specified time interval has elapsed, the STXIT IT macro can be used to establish linkage to the appropriate

---

* The IPL user exit and the job control user exit are described separately later in this chapter.

routine and subsequently, when this routine completes the special processing, an EXIT macro to return to the next sequential instruction in the main routine.

Note: *If the program issuing the STXIT IT macro is an application program, under VTAM, the exit will not be taken while VTAM is processing any request on behalf of the application program. The exit will be taken when VTAM has completed the program's request.*

Figure 10.4 shows the application of a STXIT IT macro to enter a checkpoint routine every half hour during processing. Notice that in this example the user's interval timer exit routine need not be fixed in real storage; since there is no real-time dependency, the results cannot be influenced by paging activity.

Multitasking Considerations

When the supervisor includes interval timer support, the main task and/or any subtask in a partition may issue a SETIME macro. Each may also issue a STXIT macro to establish linkage to a common user routine provided that the routine is reenterable and that each task has its own unique save area. Figure 10.5 illustrates this principle.

Task Timer User Exit

The linkage from the supervisor to a routine for processing a task timer interrupt is established and terminated by the STXIT TT macro. This linkage must be established before an interrupt occurs, or the interrupt will be ignored.

No linkage is established if the macro contains an error or is incomplete. This macro can only be issued by the main task of the partition owning the task timer.

The task timer exit routine has to return control to the supervisor by issuing an EXIT TT macro. When the EXIT TT macro is processed, the interrupt status information and registers from the save area are restored. It is important, therefore, that the contents of the save area specified in the associated STXIT TT macro not be destroyed.

Abnormal Termination User Exit Routine

The STXIT AB macro establishes linkage to a user routine that is entered whenever the issuing program is to be terminated for any reason other than a normal end-of-job. In this routine, you can do any necessary housekeeping such as closing LIOCS files and writing messages before the program is terminated. If the exit routine is associated with a subtask, you cannot recover from an error and it must end with a CANCEL, DETACH, DUMP, JDUMP, or EOJ macro. However, if the exit routine is associated with a main task, it may be preferable to continue processing. For this purpose the EXIT AB macro can be used. For further details see the *DOS/VS Supervisor and I/O Macros* manual.

```
TIMECHK  START 0
         STXIT IT,TIMINTR,TIMSA  SET UP LINK TO TIMER RTN
         MVI   STATSW,X'80'      SET SW FIRST TIME THROUGH
         SETIME 1800             TAKE CHCKPNTS EVERY 30 MIN
         .
         .
PROCESS  (perform normal processing)
         .
         .
         CLI   STATSW,X'40'      CHECK FOR TIMER INTERRUPT
         BNE   PROCESS           IF NOT CONT PROCESSING
         B     CHKPTR            IF SO TAKE CHECKPOINT

* TIMER INTERRUPT ROUTINE
TIMINTR  MVI   STATSW,X'40'      SHOW INTERRUPT
         EXIT  IT                RETURN TO INTERRUPTED PNT

* CHECKPOINT ROUTINE
CHKPTR   (do necessary processing before taking checkpnt)
         .
         .
         CHKPT SYSC01,RSTRTR,,,,DSKFLE  TAKE CHECKPOINT
         LTR   R0,R0             CHECK IF CHECKPOINT OK
         BE    ERROR             GO TO ERROR RTN IF NOT
         ST    R0,CHKPTNR        PUT CHKPT NUMBER IN MSG
         LA    R1,MSG1           GET ADDRESS OF RIGHT MSG
         STCM  R1,7,OPCCW+1      PUT MSG ADDR IN CCW
         LA    R1,OPCCB          MESSAGE CCB
         EXCP  (1)               WRITE MESSAGE TO OPERATOR
         WAIT  (1)               WAIT FOR COMPLETION
         MVI   STATSW,X'80'      RESET CHECKPOINT SWITCH
         SETIME 1800             RESET TIMER
         B     PROCESS           RESUME PROCESSING

* RESTART ROUTINE
RSTRTR   STXIT IT,TIMINTR,TIMSA  RESTORE TIMER INTERR LINK
         SETIME 1800             SET TIMER
         .
         .
         (restore everything saved in checkpoint)
         .
         .
         B     PROCESS           START PROCESSING

* MESSAGE ROUTINE FOR INVALID CHECKPOINT
ERROR    LA    R1,MSG2           GET ADDRESS OF ERR MSG
         STCM  R1,7,OPCCW+1      PUT MSG ADDR IN CCW
         LA    R1,OPCCB          LOAD MESSAGE CCB
         EXCP  (1)               WRITE MESSAGE TO OPERATOR
         WAIT  (1)               WAIT FOR COMPLETION
         CANCEL ALL              CANCEL PROGRAM
         .
         .
END      EOJ
```

Figure 10.4 Example of Using the Interval Timer for Taking a Checkpoint Every Half-hour (Part 1 of 2)

```
* CONSTANTS
TIMSA    DS   9D
OPCCB    CCB  SYSLOG,OPCCW
OPCCW    CCW  X'09',MSG1,X'20',80
MSG1     DC   CL16'CHECKPOINT NR'
CHKPTNR  DS   F
         DC   CL60'HAS BEEN TAKEN'
MSG2     DC   CL80'CHECKPOINT FAILED JOB IS CANCELED'
STATSW   DS   X
         END
```

Figure 10.4. Example of Using the Interval Timer for Taking a Checkpoint Every Half-hour (Part 2 of 2)

```
MAINTASK START 0
         .
         .              (set up addressability)
         .
         STXIT IT,STRTER,MTSKSA
         SETIME 300               MAIN TASK TIMER TO 5 MINS
         ATTACH SUBTASK1,SAVE=SAV1
         ATTACH SUBTASK2,SAVE=SAV2
         .
         .
* IT USER EXIT ROUTINE
STRTER   (reenterable routine)
         .
         .
         EXIT IT

SUBTASK1 STXIT IT,STRTER,STSK1SA USE SAME EXIT ROUTINE
         SETTIME 400              SET TIME INTERVAL
         .
         .
         DETACH

SUBTASK2 STXIT IT,STRTER,STSK2SA USE SAME EXIT ROUTINE
         SETIME 500               SET TIME INTERVAL
         .
         .
         TTIMER CANCEL            CNCL INTRVL THIS TSK ONLY
         .
         .
         DETACH

MTSKSA   DS   9D
STSK1SA  DS   9D
STSK2SA  DS   9D
SAV1     DS  16D
SAV2     DS  16D
```

Figure 10.5. Skeleton Example of Multitask Linkage to a Common IT Exit Routine

Program Check User Exit Routine

The linkage established by the STXIT PC macro instruction provides entry to a user routine for handling any program check interrupt that is not caused by a page fault (page or segment translation exception or a

translation specification exception). 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, this can be corrected or ignored (depending on the severity of the error) and control returned to the interrupted program, or termination of the program may be requested.

Note: *As with the interval timer exit, the program check exit is not taken if the program check occurs while VTAM is processing a VTAM request issued by the program. When VTAM has completed processing the request, the exit will be taken.*

```
DIVTEST  CSECT
         .
         .                          (set up addressability)
         .
         STXIT PC,PCRTN,PCSAV       SET UP PROGRAM CHECK LINK
         .
         .
         LM    R2,R3,DIVIDEND       LOAD FOR DIVIDING
         D     R2,DIVISOR           DIVIDE
         .
         .
* USER'S PROGRAM CHECK ROUTINE
PCRTN    SR    R5,R5                CLEAR REGISTER 5
         CL    R5,DIVISOR           CHECK FOR ZERO DIVISOR
         BNE   CANCELR              IF NOT CLEAR FILES & CNCL
         .
         (special recovery routine)
         .
         .
         EXIT  PC                   RETURN TO NORMAL PROC
CANCELR  PDUMP PCSAV,PCSAV+71       DUMP SAVE AREA
         .
         (close files and do other housekeeping)
         (equates and storage definitions)
         .
         CANCEL ALL
```

Figure 10.6. Skeleton Example of a Routine for Processing a Program Check Caused by Zero Division

Supervisor support for entering a user's program check routine is useful when it is known that one or more programs may be checked by processing errors that are insignificant to the results or can easily be corrected. Figure 10.6 shows a routine for recovering from a program check caused by attempting to divide by zero. In this example, any other causative errors result in the user save area being dumped before the job is terminated.

Operator Communications User Exit

A direct communications link between the operator and a program can be established by issuing a STXIT OC macro instruction. In a multitasking environment, the STXIT OC macro instruction may be issued only by the

main task in any partition. The operator procedure to initiate communication depends, however, on whether the program executes in the background or in a foreground partition.

For a program in the background partition, the operator initiates communication by pressing the external interrupt key. This activates the attention task which sets the linkage to the user's operator communications routine. This routine is then entered instead of returning to the program that issued the STXIT OC macro instruction.

For a program in a foreground partition, the operator presses the request key. This initiates an I/O interrupt. When the attention routine identifier AR appears, the operator enters *MSG* followed by the partition identifier (such as F1 or F4) which sets the linkage to the user's operator communications routine. This routine is then entered instead of returning to the program that issued the STXIT OC macro instruction.

The operator communications routine may perform any special processing, a typical application being the taking of a checkpoint record in a program that has to be canceled in order to start a high-priority job that has just been handed in; the checkpointed program can then be restarted later on.

Writing an IPL User Exit Routine

The IPL Exit allows you to do some processing at the end of IPL and prior to execution of the job control program. For example, you may want to open a job accounting file and accumulate and/or store IPL values and statistics. Data may be read from and written to the operator console.

Before you actually start coding your $SYSOPEN routine, take account of any system requirements that should be met at the time the routine is to be executed. For instance, labeled files that are to be opened need device assignments and label information in the specific label area. Any routines called by your routine must be present in the system core image library.

Moreover, the following conventions must be followed:

- Register 15 is to contain the entry point of the routine.
- Register 14 is to be loaded with the return address to job control.
- The format of the phase card must be as follows:

```
PHASE  SYSOPEN,+0[NOAUTO]
```

- The phase is to be self-relocating.

Use EXCP macros to perform all I/O operations within your routine; any use of LIOCS or of a DTFPH will destroy the job control program. After IPL job control executes the exit routine as an overlay phase. In your exit routine you can issue SVCs and perform I/O operations in user-written $$B-transient routines. While the routine is being executed job control is unable to read any JCL statements. Therefore, if your routine is written to perform some level of OPEN function for a labeled device, make sure that labels are present in the standard label area, the partition label area, or the user label area. Likewise, assignments for the specific physical devices must

have been made. Code your routine as an overlay of an existing program phase. A slot of 4K bytes has been reserved for the exit routine.

Phase $SYSOPEN will be executed with a storage protect key of zero. If the phase is abnormally terminated, the job control program will be loaded for execution.

Figure 10.7 illustrates a user-written routine that can be entered once each time the IPL procedure is performed.

```
          ISEQ   73,80
IPLEXIT   START  0
          USING  *,R15                SET BASE
BEGIN     ST     R14,RETURN           SAVE RETURN ADDRESS
          L      R1,20                GET COMREG. ADDRESS
          MVC    SYSDATE(2),79(R1)    GET DAY
          MVC    SYSDATE+3(2),81(R1)  GET MONTH
          MVC    SYSDATE+6(2),83(R1)  GET YEAR
          MVC    SYSDATE+9(3),85(R1)  GET CURRENT DAY OF YEAR
          LA     R1,LOGCCB            GET LOGCCB ADDRESS
          LA     R0,LOGCCW            GET LOGCCW ADDRESS
          ST     R0,LOGCCB+8          AND STORE IT IN CCB
INQUIRYD  LA     R8,SYSCODE           GET SYSTEM DATE ADDRESS
          ST     R8,LOGCCW            AND STORE IT IN CCW
          MVI    LOGCCW+7,X'11'       SET LENGTH
          BAL    R14,OUTLOG           WRITE MESSAGE
          LA     R0,PARM              LOAD PARAMETER REGISTER
          LA     R1,PHASNAME          LOAD PHASE NAME
          SVC    2                    OPEN ACCOUNTING
          L      R14,RETURN           LOAD RETURN ADDRESS
          BR     R14                  RETURN TO CALLER
          DC     0F'0'                ALIGNMENT
PARM      DC     C'OPEN'              SET ID.
          DC     X'80000000'
PHASNAME  DC     C'$$BACSEE'          PHASE NAME
OUTLOG    ST     R14,OUTSAVE          SAVE RETURN ADDRESS
          MVI    LOGCCW,X'09'         SET WRITE COMMAND
          SVC    0                    EXCP
          TM     2(R1),X'80'          COMPLETE?
          BO     *+6                  YES
          SVC    7                    WAIT
          MVC    MSGAREA,BLANKS       CLEAR MESSAGE AREA
          L      R14,OUTSAVE          LOAD RETURN ADDRESS
          BR     R14                  RETURN TO CALLER
OUTSAVE   DC     F'0'                 RETURN ADDRESS

INLOG     ST     R14,INSAVE           SAVE RETURN ADDRESS
```

Figure 10.7. IPL User Exit Example (Part 1 of 2)

```
INLOG1   MVI    LOGCCW,X'0A'          SET READ COMMAND
         SVC    0                     EXCP
         TM     2(R1),X'80'           COMPLETE?
         BO     *+6                   YES
         SVC    7                     WAIT
         TM     LOGCCB+4,X'01'        WAS MESSAGE CANCELED?
         BNZ    INLOG1                YES READ AGAIN
         OC     MSGAREA,BLANKS        CONVERT TO UPPER CASE
         L      R14,INSAVE            LOAD RETURN ADDRESS
         BR     R14                   RETURN TO CALLER
INSAVE   DC     F'0'                  RETURN ADDRESS
LOGCCB   CCB    SYSLOG,LOGCCW
* SUPVR COMMN MACROS - CCB - 5745-SC-SUP - REL. 28.0
LOGCCB   DC     XL2'0'                RESIDUAL COUNT
         DC     XL2'0'                COMMUNICATIONS BYTES
         DC     XL2'0'                CSW STATUS BYTES
         DC     AL1(0)                LOGICAL UNIT CLASS
         DC     AL1(4)                LOGICAL UNIT
         DC     XL1'0'
         DC     AL3(LOGCCW)           CCW ADDRESS
         DC     B'00000000'           STATUS BYTE
         DC     AL3(0)                CSW CCW ADDRESS
LOGCCW   CCW    X'00',*,X'20',0
RETURN   DC     F'0'
MSGAREA  DC     CL60' '
SYSCODE  DC     C'DATE='
SYSDATE  DC     CL12'  .  .  /   '
BLANKS   DC     CL60' '
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
         END    BEGIN
```

Figure 10.7. IPL User Exit Example (Part 2 of 2)

Writing a Job Control User Exit Routine

In your routine you are free to modify the operands of the job control statement and to add comments. You must not, however, modify the operation field of the statement. For example, // EXEC IBM can be modified to // EXEC USER; the operation field (EXEC) cannot be modified. In your exit routine do not perform any I/O operations, do not issue any SVCs, or request the system to cancel the job step.

The phase card must be coded as follows:

```
PHASE $JOBEXIT,S[,NOAUTO],SVA[,PBDY]
```

Your routine must be coded reenterable, SVA eligible, and must reside in the SVA. SVA residence may be achieved at IPL time as shown below:

If message

```
1I00A READY FOR COMMUNICATIONS
```

is displayed, enter

```
set sdl=create
  .
  .
$JOBEXIT,SVA
  .
  .
/*
```

or, if message

```
1T00A WARM START COPY OF SVA FOUND
```

is displayed, press END/ENTER. Phase $JOBEXIT, if previously loaded into the SVA, is contained in the warm start copy.

Phase $JOBEXIT is executed with a storage protection key of zero. The code is shared between partitions.

When your routine is entered, the following registers contain:

Register Number:	Contents of Register:
0	System identification characters 'SDOS'
1	Address of partition communication region
2	Address of system communication region
3	Address of job control vector table*
4	Address of buffer into which the job control statement is loaded
14	Return address to job control
15	Entry point to $JOBEXIT; at completion of the routine it contains the return code for job control.

Before taking the exit to your routine, job control saves the contents of all general-purpose registers. These registers will be restored when job control regains control.

Prior to returning control to job control, your routine must store a return code value into register 15:

a zero value - requests job control to continue processing the current statement as normal.

a non-zero value - requests job control to process the statement as if it were a comment.

The vector table shows which job control statement will be processed by job control. You must not modify its contents. Use it for comparison only. The size of the buffer into which the job control statement is loaded (left-justified) is 120 bytes, the first 71 bytes of which are printed on the console printer. The full length of 120 bytes is printed on the printer assigned to SYSLST. The / & and End-of-job statements are not displayed. In the buffer, you may modify the first byte following the operation field through byte 71. Bytes 72-80 could contain a statement identification, such as for procedure overwrites, and therefore should not be modified. After

the return code has been set, control is passed back to job control.

* **Vector Table Layout**

Operation field	7 bytes (Name of job control statement)
Condition switches	1 bytes
Branch displacement	1 byte
Phase ID.	1 byte
Total	10 bytes

Do not attempt to modify the table or modify the operation field in the buffer.

Note: *Care must be taken to ensure that your exit routine is free of errors that could cause abnormal termination in a production environment.*

Figure 10.8 illustrates a job control user exit routine.

```
// JOB EXIT ROUTINE
// OPTION CATAL,NODECK
 PHASE $JOBEXIT,S,NOAUTO,SVA,PBDY
// EXEC ASSEMBLY
          EJECT
*********************************************************************
*         THIS PROGRAM, PHASE $JOBEXIT, EXAMINES ALL EXEC CONTROL STATE-  *
*         MENTS AND EXEC COMMANDS WHETHER THEY WANT TO EXECUTE A PROGRAM  *
*         NAMED: IBM. THIS PROGRAM IS ASSUMED TO BE RESTRICTED FOR        *
*         GENERAL USE AND THE STATEMENT:                                  *
*    [//] EXEC IBM                                                        *
*         IS CHANGED TO:                                                  *
*    [//] EXEC USER                                                       *
*         MESSAGE, 'PROG. IBM RESTRICTED FOR ALL USERS',IS PLACED         *
*         INTO THE EXEC CARD AND PRINTED ON SYSLOG (IF LOG IS             *
*         ON) AND SYSLST.                                                 *
*                                                                         *
*                                                                         *
*         THE PHASE NAMED USER MUST BE CATALOGED IN THE CIL               *
*                                                                         *
*         $JOBEXIT IS REENTRANT AND SVA ELIGIBLE AND MUST BE LOADED INTO  *
*         THE SVA.                                                        *
*                                                                         *
*********************************************************************
          EJECT
JOBEXIT   START 0
          BALR   R12,0               ESTABLISH
          USING  *,R12               ADRESSABILITY
```

Figure 10.8 Job Control User Exit Example (Part 1 of 2)

```
*
*          CHECK FOR EXEC STATMENT
*          REG.3 POINTS TO JOB CONTROL VECTOR TABLE
*
           CLC     EXECNAM,0(R3)          IS IT AN EXEC STATEMENT?
           BNE     RETURN                 IF NOT RETURN
*
*          EXAMINE THE STATEMENT
*          REG.4 POINTS TO STATEMENT BUFFER
*
           L       R6,=F'1'               INCREMENT VALUE FOR SEARCH LOOP
           L       R7,=F'67'              COUNT MAXIMUM FOR SEARCH LOOP
           SR      R5,R5                  CLEAR R5, USED AS INDEXING REG.
*
*          FIND POSITION OF EXEC STATEMENT
*
SEARCHE    EQU     *
           LA      R8,0(R5,R4)            POINT TO INDEXED POS. IN STMNT. BUF
           CLC     EXECNAM,0(R8)          DETERMINE POSITION OF EXEC
           BE      EXFOUND                FOUND THE STATEMENT
           BXLE    R5,R6,SEARCHE          INCREMENT INDEX AND LOOP
           LA      R15,8                  NO EXEC FOUND, RETURN CODE=8
           BR      R14                    RETURN TO CALLER
EXFOUND    EQU     *
           LA      R5,5(R5)               SKIP OVER EXEC TO PROGNAME
SEARCHP    EQU     *
           LA      R8,0(R5,R4)            POINT TO INDEXED POS. IN STMNT. BUF
           CLC     PROGNAM,0(R8)          LOOK FOR PROGRAM-NAME IBM
           BE      PFOUND                 PROGRAM-NAME FOUND
           BXLE    R5,R6,SEARCHP          INCREMENT INDEX AND LOOP
           B       RETURN                 IF ANY OTHER OR NO PROG.-NAME RETURN
*
*          PROGRAM-NAME-IBM-FOUND PROCESSING
*
PFOUND     EQU     *
           LA      R4,0(R5,R4)            POINT TO PROG.-NAME IN BUFFER
           MVC     0(L'USERTXT,R4),USERTXT MOVE USERTXT TO BUFFER
*
*          PREVIOUS MVC CHANGED PROGRAM-NAME IBM INTO PROGRAM-NAME USER
*          AN ADDITIONAL MESSAGE IS MOVED INTO THE BUFFER
*
RETURN     EQU     *
           SR      R15,R15                RETURNCODE ZERO TO REG.15
           BR      R14                    RETURN TO CALLER
EXECNAM    DC      C'EXEC'
PROGNAM    DC      C'IBM'
USERTXT    DC      C'USER *** PROG. IBM RESTRICTED FOR ALL USERS'
R3         EQU     3
R4         EQU     4
R5         EQU     5
R6         EQU     6
R7         EQU     7
R8         EQU     8
R12        EQU     12
R14        EQU     14
R15        EQU     15
           END     JOBEXIT
/*
// EXEC LNKEDT
/&
```

Figure 10.8. Job Control User Exit Example (Part 2 of 2)

Checkpointing Facility

The progress of a program that performs considerable processing in one job step should be protected against destruction in case the program is canceled. DOS/VS provides support for taking up to 9999 checkpoint records in a job. Through this facility, information can be preserved at regular intervals and in sufficient quantity to allow restarting a program at an intermediate point.

The CHKPT macro stores the checkpoint record on a magnetic tape or disk. For full details regarding the use and restrictions of this macro, refer to *DOS/VS Supervisor and I/O Macros.*

The RSTRT job control statement restarts the program from the last or any specified checkpoint taken before cancelation. For full details on using this statement, see *DOS/VS System Control Statements.*

Choosing a Checkpoint

The most important criterion for a checkpoint decision is a minimum of necessary housekeeping before the checkpoint record can be taken. The possibility of an error occurring either in the checkpoint routine or at restart is then also minimal. Checkpoints cannot be taken by a subtask or by a main task with subtasks attached. Therefore, when multitasking, checkpoints should be avoided where a number of subtasks must first be detached.

A successful checkpoint record taken immediately after opening files indicates that processing can safely proceed. If such a checkpoint record is invalid, however, then the program should be canceled.

Other checkpoint records may be taken at logical breaks in data, such as at the end of a reel of magnetic tape.

After a CHKPT macro is successfully executed, register 0 contains the checkpoint number; if CHKPT macro execution is unsuccessful register 0 contains zero, and the reason for failure is printed on SYSLOG.

Timing the Entry to the Checkpoint Routine

Having decided where a program can conveniently be checkpointed, it may be useful to enter the checkpoint routine only if a certain time interval has elapsed since the previous checkpoint record was taken.

By issuing a SETIME macro after a STXIT IT macro has established linkage to a user routine that sets a switch and returns, the main program can test this switch and then branch to the checkpoint routine or continue processing according to whether the switch is set or not. An example of this technique can be found in Figure 10.4.

By issuing a STXIT OC macro instruction, it is also possible to have checkpoint records taken at convenient points on command from the operator. This method is illustrated by Figure 10.9.

```
CHKPTRTN CSECT
         (set up addressability)
         STXIT  OC,OCMSG,OCSAV SET UP LINKAGE FOR OC MSG
         .
         .
         MVI    SW1,X'40'      SET CHECKPOINT SWITCH
         OPENR  (RDISKOUT),(RCHKPTF)  OPEN FILES
* DTF ADDRESSES FROM MAIN ROUTINE ARE USED
         BAL    RLINK,CHECKPT  TAKE TEST CHECKPOINT
         .
         .
START    (normal processing)
         .
         .
         CLI    SW1,X'40'      SEE IF OPER HAS SENT MSG
         BE     START          CONTINUE IF NOT

* THE FOLLOWING IS THE CHECKPOINT ROUTINE ENTERED ON
* A SIGNAL FROM THE OPERATOR
         STD    F0,REG0        SAVE FLOATING POINT REGS
         STD    F2,REG2
         STD    F4,REG4
         STD    F6,REG6
         CHKPT  SYS011,(RSTRTR),,,,(RCKPTF)  TAKE CHKPTS
         LTR    R0,R0          TEST IF SUCCESSFUL
         BZ     CANCEL         CANCEL IF NOT
         MVI    SW1,X'40'      RESET CHECKPOINT SWITCH
         B      START          RETURN TO NORMAL PROCESSING
         .
         .
         (equates)
OCMSG    MVI    SW1,X'80'      SET CHECKPOINT SWITCH
         EXIT   OC             RETURN TO POINT OF INTERR
CHECKPT  CHKPT  SYS011,(RSTRTR),,,,(RCHKPTF)
         LTR    R0,R0          SEE IF CHECKPNT SUCCESSFUL
         BNZ    0(RLINK)       RETURN IF TAKEN
CANCEL   CANCEL ALL            CANCEL IF CHECKPOINT FAILED
STRTR    STXIT  OC,OCMSG,OCSAV RESTORE LINKAGE
         LD     F0,REG0        RESTORE FLOATING POINT REGS
         LD     F2,REG2
         LD     F4,REG4
         LD     F6,REG6
         B      START          RESTART PROGRAM
END      EOJ
REG0     DS     D
REG2     DS     D
REG4     DS     D
REG6     DS     D
OCSAV    DS     9D
SW1      DS     X
         .
         .
         (equates)
         .
         .
         end
```

Figure 10.9. Skeleton Example of a Routine for Checkpointing a Program on Operator Command

Saving Data for Restart

Besides the information stored by the CHKPT macro, certain data must usually be saved by the user's checkpoint routine in order to facilitate a

successful restart. This may include the contents of floating point registers, any linkage that was established by a STXIT or a SETPFA macro, the interval value for a SETIME macro, and the program mask in the problem program PSW.

For the repositioning of I/O files so that they point to the next record to be read or written, refer to *DOS/VS Supervisor and I/O Macros.*

Restarting a Checkpointed Program

A checkpointed program can be restarted only in the same partition. The virtual partition (or real partition if a real mode program) must start at the same location as when the program was checkpointed and its end address must not be lower than at that time unless a lower end address was specified in the CHKPT macro instruction. Unless the user resets all linkages to SVA phases himself, the contents and location of the modules in the SVA when restarting must also be the same as when the program was checkpointed. The SDL need not be identical.

If any pages of a virtual mode program were fixed when the checkpoint record was taken, then the real partition must also start at the same location and its end address must be at least as high as at that time. The pages that were fixed are refixed by the supervisor when the program is restarted.

The appropriate job control statements for restarting a checkpointed program on disk are illustrated in Figure 10.10.

```
// JOB   CHECKPOINT   (the JOBNAME must be the same as before)
// ASSGN ...          (all ASSGNs must be renewed)
// ASSGN ...          (new assignments may be made)
// ASSGN ...
// RSTRT SYS001,1111,CHKPTF
```

Figure 10.10. Example of Job Control Statements for Restarting a Checkpointed Job from Checkpoint 1111

Job Accounting Interface Feature

A DOS/VS supervisor generation option provides job accounting interface support for all partitions in the system. At the end of each job step or job, accounting information is accumulated in a table for that partition and can be processed by a user routine, which must be either relocatable or self-relocating. This user routine can extract data for such purposes as charging system usage, supervising system operation, or for planning new applications or changing the system configuration.

Since the processing of the information is an overhead element, the user routine should be efficient and avoid unnecessary reduction or reformatting of data.

If your system also supports POWER/VS job accounting, you do not need such a user routine. Refer to *DOS/VS POWER/VS Installation Guide and Reference.*

Basic Job Accounting Information

When support is generated for basic job accounting, the supervisor includes for each partition in the system a job accounting table comprising fourteen fields. At the end of each job step and job, information is stored as shown in Figure 10.11, fields 1 to 14 inclusive.

Job accounting automatically includes support for the interval timer.

I/O Accounting Information

Additional support can be provided at system generation time to include the number of SIO (Start I/O) instructions issued per device for each job step and job. The job accounting table for each partition is then extended to contain the additional fields 15 and 16 shown in Figure 10.11.

SIO accounting is performed for the number of devices specified to be supported by the feature for each partition. The maximum is 255 and has no relation to the number of devices specified for the system. If more devices are accessed than the number specified, SIOs on the excess devices will not be counted.

Save Area for the User's Routine

The address of a save area that can be used by the user's routine is passed in general register 13. This save area is 16 bytes long unless a greater length (up to 1024 bytes), to save DTF information for LIOCS, was specified at system generation time. However, CCBs and executable CCWs must not be included.

User's Area for LIOCS Label Processing

If the user's routine uses LIOCS for processing such items as standard tape labels, DTFDA, or DTFPH with MOUNTED=ALL, then an alternative label area must be specified at system generation time. The length of this label area should normally be the number of bytes that would be allocated by a given parameter of the LBLTYP statement. For information on determining the number of bytes, see *DOS/VS System Control Statements.*

Programming Considerations

The user program to process the information entered by the supervisor in the Job Accounting Table must be cataloged in a core image library with the name $JOBACCT. If the supervisor supports relocating load, then the user program must be relocatable, otherwise it must be self-relocating in a multiprogramming environment.

Field	Displacement	Byte Length	Contents
1	0 - 7	8	Job name. 8-byte character string taken from JOB card.
2	8 - 23	16	User Information. 16 characters of information taken from the JOB card.
3	24 - 25	2	Partition ID, BG, F4, F3, F2, or F1.
4	26	1	Cancel Code. Refer to *DOS/VS Serviceability Aids and Debugging Procedures, GC33-5380.*
5	27	1	Type of Record. S = job step; L = last step of job.
6	28 - 35	8	Date. mm/dd/yy or dd/mm/yy depending on supervisor option.
7	36 - 39	4	Job Step Start Time. 0hhmmssF, where h hours, m minutes, s seconds, F is a sign (in packed decimal format).
8	40 - 43	4	Job Step Stop Time (in same format as start time).
9	44 - 47	4	Reserved.
10	48 - 55	8	Phase Name. 8-byte character string taken from the EXEC card.
11	56 - 59	4	Real Mode Processing: High storage address of partition. If the SIZE parameter is used in the EXEC statement, this field reflects the value of the parameter. Virtual Mode Processing: Simulated high storage address. Calculated by multiplying the number of pages referenced in the partition by 2K and adding the result to the start address of the virtual partition.
12	60 - 63	4	CPU Time. 4 binary bytes given in 300ths of a second. Time is calculated from exit of the user-written routine called during job control to next entry of the routine. Time used by the user-written output routine is charged to overhead of the next record.
13	64 - 67	4	Overhead Time. 4 binary bytes given in 300th of a second. Includes time taken by functions that cannot be charged readily to one partition (such as attention routine and error recovery). System overhead time is divided by the number of active batch partitions and recorded in each accounting table.
	68 - 71	4	All Bound Time. 4 binary bytes in 300th of a second. This is the time the system is in the wait state divided by the number of partitions running.
15	72 -		SIO Tables. Variable number of bytes. Six bytes are reserved for each device specified in the JA parameter. First two bytes are X.0cuu', next four are hex count of SIOs for job step. Unused entries contain X'10' followed by five bytes of zeros. Stacker select commands for MICR devices are not counted. Error recovery SIOs are not charged to the JOB Accounting Table. Devices are added to the table as they are used.
16		1	Overflow. Normally X'20'. Set to X'30' if more devices are used than set by the JA parameter at system generation time.

Note: *The difference between Start and Stop times will not necessarily equal the sum of CPU, All Bound, and Overhead times. All Bound and Overhead times will vary, depending on the number of active partitions and the type of partition activity. CPU time is accurate for each partition, but it may not be reproducible. That is, the same job being executed under different system conditions (varying number of active partitions, logical transient available, etc.) may show differences in CPU time.*

Figure 10.11. Job Accounting Table

If physical IOCS is used for printing, a skip must be issued to prevent overwriting of job control statements.

For efficiency, an overlay structure should be avoided and the length of the program should preferably not exceed one core image library block.

If the job accounting program is canceled as the result of an error condition, the current information cannot be retrieved. If the job accounting program is canceled, the job accounting information for the current job step is unreliable. However, provision is made that the job accounting information for any subsequent job steps will be correct, provided the cancellation was not caused by an error in the $JOBACCT routine itself. If there was an error in the $JOBACCT routine, it must be corrected first. If not, the routine will be re-entered for every subsequent job step and will continue to produce wrong job accounting information.

In order to avoid accidental cancellation of the job accounting program by the operator, the operator should issue the MAP command and check the job name for the running partition. If the job name is 'JOB ACCT', the job accounting routine is active; the CANCEL command should not be issued until the original job name is displayed after another MAP command.

Register Usage

Important data for the user's job accounting routine are passed in the following general registers:

12 Base address for $JOBACCT
15 Address of the job accounting table
11 Length of the job accounting table
13 Address of the user save area
14 Return address to job control

If $JOBACCT uses LIOCS, the contents of general registers 14 and 15 must be saved (also registers 0 and 1 if necessary) because LIOCS uses these registers.

Tailoring the Program

The requirements of the program may be simply to record the accounting information as part of the SYSLST output for each job step or job, or it may be to accumulate information to be used for equitably allocating the costs of a computing center.

If data is to be written out on a disk or tape, the save area can be used for communicating between job steps. Such information as the disk address for the next record or an indication that tape labels have been successfully processed, or even the DTF used to control the output, may be stored in the save area.

Figure 10.12 illustrates a job accounting program that writes records to disk without additional processing.

```
JAACT    CSECT
         USING *,R12
         USING JASAVE,R13          JOB ACCT SAVE AREA
         LA    R0,JABROUT          AB ROUTINE
         LA    R1,JABSAVE          AB SAVE AREA
         STXIT AB,(0),(1)          SET ABNRML TERM EXIT
         TM    JASTATSW,X'C0'      TEST STATUS
         BO    JARET               DISK AREA FULL
         BM    JAOPEN              SAVE AREA INITIALIZED
* PERFORM LABEL PROCESSING AND INITIALIZE SAVE AREA
         OPENR JADTF               OPEN FILE (see Note)
         MVC   JACCB,JADTF         MOVE CCB TO SAVE AREA
         MVC   JASEEK,JADTF+58     EXTENT LOWER LIMIT
         MVI   JAR,X'01'           FIRST RECORD
         MVC   JAHIGH,JADTF+54     HIGH EXTENT LIMIT
* RELOCATE CCWS
         MVC   JASKCCW(32),JAMODCCW PUT MOD CCWS IN SVE AREA
         LA    R10,JASEEK          SEEK ADDRESS
         STCM  R10,7,JASKCCW+1     PUT ADDRESS IN CCW
         LA    R10,JASRCH          SEARCH ADDRESS
         STCM  R10,7,JASRCCW+1     PUT ADDRESS IN CCW
         LA    R10,JASRCCW         SEARCH CCW ADDRESS
         STCM  R10,7,JATIC+1       PUT ADDRESS IN CCW
         LA    R10,JASKCCW         CHANNEL PROGRAM ADDR
         STCM  R10,7,JACCB+9       PUT ADDRESS IN CCB
         MVI   JASTATSW,X'80'      IND SAVE AREA INIT
* WRITE JOB ACCOUNTING TABLE TO DISK
JAOPEN   STCM  R15,7,JADATA+1      PUT ADDR OF TBL IN CCW
         MVC   JADTF(16),JACCB
         LA    R1,JADTF            POINT TO CCB
         EXCP  (1)                 WRITE DATA
         WAIT  (1)                 WAIT FOR COMPLETION
* UPDATE SEEK ADDRESS
         TR    JAR,JARECTAB        RECORD
         CLI   JAR,X'01'           NEW TRACK
         BNE   JARET               NO
         TR    JAHEAD+1(1),JAHDTAB HEAD
         CLI   JAHEAD+1,X'00'      NEW CYLINDER
         BNE   JAHTST              NO
         LH    R10,JACYL           CYLINDER ADDRESS
         LA    R10,1(R10)          INCREMENT BY ONE
         STH   R10,JACYL           REPLACE IN SEEK ADDR
JAHTST   CLC   JAHIGH,JASRCH       BEYOND UPPER LIMIT
         BH    JARET               NO
         LA    R1,JACCBL           CONSOLE CCB
         LA    R2,JAMSG1           ERROR MESSAGE
         STCM  R2,7,9(R1)          PUT ADDRESS IN CCB
         LA    R3,JAERR1           DATA ADDRESS
         STCM  R3,7,1(R2)          PLACE IN CCW
```

Note: *If the supervisor does not support relocating load, the self-relocating form of the OPEN macro (OPENR) should be used in a multiprogramming environment; otherwise OPEN may be used instead.*

Figure 10.12. Job Accounting Routine Example (Part 1 of 2)

```
         EXCP  (1)                    INFORM OPERATOR
         WAIT  (1)                    WAIT FOR COMPLETION
         OI    JASTATSW,X'40'         INDICATE DISK FULL
JARET    STXIT AB                     RESET EXIT LINKAGE
         BR    R14                    RETURN TO SUPERVISOR
JABROUT  BALR  R10,0                  BASE REGISTER
         USING *,R10                  ESTABL ADDRESSABILITY
         LA    R1,JACCBL              CONSOLE CCB
         LA    R2,JAMSG2              ERROR MESSAGE
         STCM  R2,7,9(R1)             PUT ADDRESS IN CCB
         LA    R3,JAERR2              DATA ADDRESS
         STCM  R3,7,1(R2)             PLACE IN CCW
         EXCP  (1)                    INFORM OPERATOR
         WAIT  (1)                    WAIT FOR COMPLETION
         EOJ
JAMODCCW CCW   X'07',*,X'60',6
         CCW   X'31',*,X'60',5
         CCW   X'08',*,X'00',1
         CCW   X'05',*,X'20',246
JACCBL   CCB   SYSLOG,*
JADTF    DTFPH TYPEFLE=INPUT,         MEANS CHECK LABELS  *
               DEVICE=2314,                               *
               MOUNTED=SINGLE
         ORG   JADTF
         DC    X'00000B00'            SET CCB OPTION BITS
         ORG
JAMSG1   CCW   X'09',JAERR1,X'20',L'JAERR1
JAMSG2   CCW   X'09',JAERR2,X'20',L'JAERR2
JAERR1   DC    C'JOB ACCOUNTING DISK FULL'
JAERR2   DC    C'JOB ACCOUNTING ROUTINE CANCELED'
JARECTAB DC    X'000203040506070809 0A0B0C0D0E0F101112131401'
JAHDTAB  DC    X'010203040506070809 0A0B0C0D0E0F1011121300'
JABSAVE  DS    9D
         LTORG USED IF LITERALS ARE PRESENT
JASAVE   DSECT
JASEEK   DS    0XL6                   SEEK ADDRESS BBCCHH
JABB     DS    XL2                    BB
JASRCH   DS    0XL5                   SEARCH ADDRESS CCHHR
JACYL    DS    XL2                    CC
JAHEAD   DS    XL2                    HH
JAR      DS    X                      R
JASTATSW DS    X
JACCB    DS    XL16                   COMMAND CONTROL BLOCK
JAHIGH   DS    XL4                    HIGH EXTENT LIMIT
         DS    XL4
JASKCCW  CCW   X'07',JASEEK,X'60',6   SEEK CCW
JASRCCW  CCW   X'31',JASRCH,X'60',5   SEARCH CCW
JATIC    CCW   X'08',JASRCCW,X'00',1 TIC CCW
JADATA   CCW   X'05',*,X'20',246      WRITE DATA ASSUMING 29
*                                     SIO DEVICES TRACED
         CSECT
JABROUTE EQU *                        YOUR AB ROUTINE
         :
         (equates)
         :
         END
```

Note: *The DSECT labeled JASAVE through JADATA defines the layout of the Job Accounting user save area, which resides within the supervisor. The address of this area is passed, in register 13, to your Job Accounting phase. When generating your supervisor you must specify the desired length of this save area by substituting a value for s, the first operand of the JALIOCS parameter of the FOPT macro. If the operand is omitted or if JALIOCS=NO is specified the length of the user save area is set to 16 bytes by default.*

Figure 10.12. Job Accounting Routine Example (Part 2 of 2)

Storage Dump Facility

Whenever a program is to be terminated by the system for any 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. Facilities for obtaining such a printout are provided by most high-level languages and are described in the various language manuals. For guidance on reading and interpreting the printout, see *DOS/VS Serviceability Aids and Debugging Procedures*.

The DOS/VS supervisor supports several macro instructions that dump the contents of real or virtual partitions to SYSLST, which may be assigned to a printer, a disk, or a tape unit. These macro instructions, details of which are given in *DOS/VS Supervisor and I/O Macros,* may be used, for example, at the end of a user's routine for handling an abnormal termination condition.

The following is a summary of the functions of supervisor macros that provide storage dumps:

DUMP — The DUMP macro instruction dumps, in hexadecimal format, the contents of the supervisor area, or the contents of the supervisor control blocks, depending on the parameter specified in the STDJC macro during system generation or on the // OPTION job control statement in a specific job step. (For details, see *DOS/VS Serviceability Aids and Debugging Procedures.*) In addition, the DUMP macro instruction dumps the entire real or virtual partition of the issuing program, and all the general registers. The job step is always terminated if multitasking is not supported; with multitasking, 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 instruction causes the same areas to be dumped as for a DUMP macro, but terminates the entire job.

PDUMP — A PDUMP macro instruction furnishes a dynamic hexadecimal dump of the general registers and of the virtual or real storage area between the addresses specified by two operands. After execution of this macro instruction, processing continues at the next sequential instruction.

A PDUMP macro instruction may therefore be issued several times in a program to provide dumps of selected storage fields for examination at different stages of the program's execution.

DASD Switching under DOS/VS

The standard I/O interface between an I/O device and the CPU is a channel and a control unit.

Normally, this interface provides one, and only one, path by which a

CPU communicates with an I/O device. However, it may be desirable to access a device, especially a DASD device, by more than one path. For example, a second CPU may be required to back-up the host CPU such that should the host CPU become inoperable, the attached DASD devices may be switched immediately to (made accessible to) the back-up CPU. Multiple CPUs may also need to access the same data base.

A single CPU may require back-up channels and control units, providing alternate paths to the same DASD devices.

In order to do this device sharing the hardware provides a two-level switching mechanism that allows you to connect one or more DASDs either dynamically or manually to different I/O paths. This mechanism is known as channel switching and string switching.

Channel Switching

Channel switching provides the switching mechanism at the control unit level. The channel switch allows you to connect the control unit to up to four channels, which may belong to the same or different CPUs thus providing up to four distinct I/O paths. A maximum of two channels may connect to one CPU. The connection of any channel can be manually enabled or disabled. When enabled, the switch is dynamically controlled by the hardware.

String Switching

In the case of string switching, the switching mechanism is at the DASD string level. String switching allows you to connect a string of DASDs to two distinct control units, or integrated disk attachments. The two I/O paths may be connected to a single or two different CPUs.

Using DASD Switching

In both types of this hardware-supported switching, a desired I/O path may be selected in one of two ways. In the first case, connection is made dynamically when an I/O command is issued for a device. Provided that the control unit (in channel switching) and the DASD string (in string switching) are free for connection, the target DASD device can be accessed by the requesting CPU. Once a connection is estabilshed by one CPU, the other CPU receives device busy status if attempting to access a device on the string.

In the second case, the operator may manually switch the sharable devices to the desired CPU (via the Enable/Disable toggle switches). It should be noted that in this case an entire string of DASD is disconnected from the other CPU.

If, at your installation, a DASD switching feature is being used, it is your responsibility to resolve conflicting CPU references to shared devices (or files) and thus ensure data integrity. Following are two ways of preventing potential conflicts.

First, through scheduling of CPU file referencing, ensure that only one CPU that is updating the file is connected to the shared DASD. The operator needs only to switch the manual control to the updating CPU for that period of time.

Secondly, through scheduling and the use of the operator commands DVCUP and DVCDN (as described below), devices may be reserved for use by one CPU for a particular period of time.

An individual device can be excluded from use by a particular CPU by entering a DVCDN command for that device via the operator console. The other system then has exclusive access to that device. The device can be made available again by issuing a DVCUP command for the device. However, the other system should then issue a DVCDN command for that device. To avoid conflicts both system operators have to inform each other about the status of the reserved devices. It is therefore recommended that a job, which requires exclusive access to a file or device, notifies the operator when the device has to be reserved, and when it may be released.

Note that the DVCUP/DVCDN commands reserve the DASD at the device level, although the programmer may only be interested in reserving only one file on that particular device.

DVCUP and DVCDN commands can only be entered via the console.

Further hardware details on channel or string switching may be found in the appropriate hardware manuals for the IBM 370/115, 370/125, 3340, 3330, 3350, and 3830 Storage Control Unit Model 2.

When using DASD Switching, in order to facilitate the diagnosis of hardware failures, the inclusion of Recovery Management Support (RMS) is required. RMS may be included during DOS/VS system generation by specifying RMS=YES in the SUPVR macro.

Appendix A: System Layout on Disk

Figure 11.1 illustrates how DOS/VS is organized on the volume containing the system residence file, which is called SYSRES. The device itself can be any IBM DASD device except a 2321 data cell, or a 2311 disk. The organization of SYSRES is as follows:

IPL

This area contains the initial program load (IPL) bootstrap program, which causes the IPL retrieval program to be read from SYSRES and loaded into real storage.

System Volume Label

The volume label (VOL1 label) contains the address of the volume table of contents (VTOC) established when the pack was initialized. (The DOS/VS system utility program Initialize Disk is provided for this purpose). The VTOC can be located on any cylinder outside of the SYSRES file.

User Volume Label

The user volume label area is provided for any additional standard volume labels (VOL2-VOL8 labels). This area can extend from record 4 through the end of track 0.

System Directory

This area contains the system (master) directory. Record 1 contains the starting address of the core image directory and the address of the label information cylinder. Records 2, 3, and 4 contain the starting addresses of the relocatable directory, source statement directory, and procedure directory, respectively. Record 5 contains the IPL retrieval program, which reads the supervisor from the core image library into real storage.

Component		Starting Disk Address				Number of Tracks (Alloc.)	R=Required O=Optional
		BB	CC	HH	R		
IPL Bootstrap Record 1 ($$A$IPL1)		00	00	00	1	1	R
IPL Bootstrap Record 2 ($$A$IPLA)		00	00	00	2		R
System Volume Label		00	00	00	3		R
User Volume Label		00	00	00	4		O
System Directory	Record 1	00	00	01	1	1	R
	Record 2	00	00	01	2		R
	Record 3	00	00	01	3		R
	Record 4	00	00	01	4		R
IPL Retrieval Program ($$A$IPL2)		00	00	01	5		R
Core Image Directory	Cataloged phases						
	Linked Phase	00	00	02		*	R
Core Image Library		00	End of CI Directory: X	End of CI Directory: Y+1	1	*	R
Relocatable Directory		00	End of CI Library: Z+1	End of CI Library: 00	1	*	O
Relocatable Library		00	End of RL Directory: X	End of RL Directory: Y+1	1	*	O
Source Statement Directory		00	End of RL Library: Z+1	End of RL Library: 00	1	*	O
Source Statement Library		00	End of SS Directory: X	End of SS Directory: Y+1	1	*	O
Procedure Directory		00	End of SS Library: Z+1	End of SS Library: 00	1	*	O
Procedure Library		00	End of P Directory: X	End of P Directory: Y+1	1	*	O
Label Information Cylinder(s)		00	End of P Library: Z+1	End of P Library: 00	1	2314/2319:20 3330/3333:19 3340/3344:24 3350:30	R

* Allocation Dependent on User Requirements
X=Ending CC of the Preceding Directory
Y=Ending HH of the Preceding Directory
Z=Ending CC of the Preceding Library

Figure 11.1. System Residence Organization

Core Image Directory

This directory consists of two or more tracks, depending on the allocation specified by the user. The directory is in two parts: the first is the directory of cataloged phases; the second is the directory of linked phases.

Note: *As the directory for linked phases begins at the first unused track of the core image directory, the total directory allocation must allow for this.*

Each directory entry describes one phase in the core image library and contains such information as the phase name, loading address, number of blocks, type of phase, entry point, starting disk address in the core image library, and the number of text bytes in the last block. The entries are sorted in alphameric sequence.

The first entry in the directory is called the library descriptor entry. This contains such information as the number of directory tracks, library cylinders, active phases, directory blocks available, and library blocks available.

Thereafter, the entries have a length varying from 18 bytes to 34 bytes (depending on the specifications in the PHASE statement). Entries are grouped in blocks of 256 bytes, plus an 8-byte key for the highest phase name in the block. The number of blocks per track for the 2314/2319, 3330/3333, 3340 and 3350 is 17, 28, 16, and 36, respectively. As the size of an entry can vary from 18 to 34 bytes, one block can have a maximum of 18 entries. The maximum number of entries per track again depends on the device.

Core Image Library

The core image library consists of one or more complete cylinders, depending on the allocation specified by the user. Each block is 1024 bytes. For the 2314/2319, each track contains six blocks. For the 3333/3330 each track contains eleven blocks. For the 3340, each track contains seven blocks. For the 3350, each track contains fifteen blocks. The number of phases and the size of each program dictates the number of cylinders that must be allocated. Each program starts with a new block.

Relocatable Directory

This directory consists of one or more tracks, depending on the allocation specified by the user. It contains two types of information:

1. System directory information for the relocatable directory and library. This information occupies the first five entries of the first record in the relocatable directory.
2. An entry that describes each module (the output of a complete language translator run) in the relocatable library and contains: the module name, total number of text-record blocks required to contain this module, starting disk address of the first text-record of this module, and change level identification.

Relocatable Library

The relocatable library consists of one or more complete cylinders, depending on the allocation specified by the user. The number of modules and the size of each module to be contained in this library dictate the number of tracks that must be allocated. Each allocated track contains 16 blocks (2314/2319), or 28 blocks (3333/3330), 17 blocks (3340), or 37 blocks (3350), and block has a fixed length of 322 bytes. Each module starts with a new block but not necessarily a new track.

Source Statement Directory

This directory consists of one or more tracks, depending on the allocation specified by the user. It contains two types of information:

1. System directory information for the source statement directory and library. This information occupies the first five entries of the first record in the source statement directory.
2. An entry that describes each book (a sequence of source language statements in a compressed card image format, accessed by a single name) in the source statement library and contains: a sublibrary prefix, the book name, starting disk address of the first block of this book, total number of blocks required to contain this book in the source statement library, and change level information.

Source Statement Library

The source statement library consists of one or more complete cylinders, depending on the allocation specified by the user. The number of blocks and the size of each book to be contained in this library dictates the number of tracks that must be allocated. Each track contains 27 blocks (2314/2319) or 44 blocks (3333/3330) or 26 blocks (3340) or 55 blocks (3350). Each block has a fixed length of 160 bytes. Each book starts with a new block but not necessarily on a new track.

Procedure Directory

This directory consists of one or more tracks depending on the allocation specified by the user. It contains two types of information:

1. System directory information for the procedure directory and procedure library. This information occupies the first five entries of the first record in the procedure library.
2. An entry that describes each procedure (a set of control statements in card image format) cataloged in the procedure library and contains: the name of the procedure, the starting disk address of the procedure, the number of blocks occupied in the procedure library, and a version and modification level.

The blocksize of the directory is 160 bytes, and the length of each entry is 16 bytes.

Procedure Library

The procedure library consists of one or more complete cylinders, depending on the allocation specified by the user. Each procedure consists of one or more consecutive 80-byte blocks, containing control statements (one card image per block).

Label Information Cylinder(s)

The label information cylinder(s) contains standard, partition standard, and user label information for background and foreground partitions. This area

is allocated 2 cylinders on the 3340 or 1 cylinder on other DASD. Job control stores label information found in job control statements here. The label information cylinder(s) is/are the last cylinder on the SYSRES file.

The LSERV program can be executed to print the contents of the label information cylinders on SYSLST. Secured data files are not listed. Information on the LSERV program can be found in *DOS/VS Serviceability Aids and Debugging Procedures.*

Volume Table Of Contents

Following the label information cylinder(s), the use of the remaining areas on the disk pack is left to the user's discretion. However, the volume table of contents (VTOC) must be contained on the same physical disk pack as the SYSRES file. (A VTOC is required on every disk pack, and is created by the Initialize Disk utility.) The VTOC is most frequently the last cylinder before the alternate track area for SYSRES. For non-SYSRES packs, the standard location is cylinder 0, track 0, record 4 to the end of cylinder 0. The location and length of the VTOC are determined when the pack is initialized. (The DOS/VS system utility program, Initialize Disk, is provided for this purpose.) The DOS/VS system utility program VTOC Display can be used to obtain a formatted listing of the VTOC. (Refer to *DOS/VS System Utilities.*)

The VTOC is a file describing the organization of the disk pack. It contains the VTOC identifier (format 4 label) that contains the starting and ending addresses of the VTOC, a format 5 label that is not used by DOS/VS, and format 1, 2, and 3 labels that identify and describe all files on the pack. More specific information on label formats is contained in the *DOS/VS DASD Labels.*

Alternate SYSRES Layout

In Figure 11.1 the relocatable library, the source statement library, and the procedure library are shown as optional areas of the SYSRES file, because these libraries are not essential for system operation. If desired, the relocatable and source statement libraries can be defined as private libraries; a private library for the procedure library is not supported. A private core image library can also be defined, but the system core image library must always be included on the SYSRES file. Planning information concerning private libraries is contained in the section *Planning theLibraries* in *Chapter 3: Planning the System.*

Glossary

This glossary defines the terms proper to this manual. If you do not find the term you are looking for, refer to the *IBM Data Processing Glossary,* GC20-1699.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard Vocabulary for Information Processing (Copyright © 1970 by American National Standards Institute, Incorporated), which was prepared by Subcommittee X3K5 on Terminology and Glossary of American National Standards Committee X3. American National Standard Definitions are marked with an asterisk (*).

* **access method:** A technique for moving data between virtual storage and input/output devices.

access method services: A multifunction service program that defines VSAM files and allocates space for them, converts indexed-sequential files to key-sequenced files with indexes, modifies file attributes in the catalog, reorganizes files, facilitates data portability between operating systems, creates backup copies of files and indexes, helps make inaccessible files accessible, and lists the records of the files and catalogs.

address: (1) An identification, as represented by a name, label, or number, for a register, location in storage, or any other data source or destination such as the location of a station in a communication network. (2) Loosely, any part of an instruction that specifies the location of an operand for the instruction.

address translation: The process of changing the address of an item of data or an instruction from its virtual address to its real storage address. See also dynamic address translation.

alternate track: One of a number of tracks set aside on a disk pack for use as alternatives to any defective tracks found elsewhere on the disk pack.

application program: A program written by a user that applies to his own work.

assembler language: A source language that includes symbolic machine language statements in which there is a one-to-one correspondence with the instruction formats and data formats of the computer.

attach: (1) To create a task and present it to the supervisor. (2) A macro instruction that causes the control program to create a new task and indicates the entry point in the program to be given control when the new task becomes active.

auxiliary storage: Data storage other than real storage; for example, storage on magnetic tape or disk. Synonymous with external storage, secondary storage.

blocking: Combining two or more logical records into one block.

blocking factor: The number of logical records combined into one physical record or block.

book: A group of source statements written in any of the languages supported by DOS/VS and stored in a source statement library.

buffer: An area of storage that is temporarily reserved for use in performing an input/output operation, into which data is read or from which data is written. Synonymous with I/O area.

byte: A sequence of eight adjacent binary digits that are operated upon as a unit and that constitute the smallest addressable unit of the system.

card punch: A device to record information in cards by punching holes in the cards to represent letters, digits, and special characters.

card reader: A device which senses and translates into machine code the holes in punched cards.

cardless system: A System/370 Model 115/125 configured without a card reader or card punch, but with an IBM 3540 Diskette Input/Output Unit.

catalog: To enter a phase, module, book, or procedure into one of the system or private libraries.

* **central processing unit:** A unit of a computer that includes the circuits controlling the interpretation and execution of instructions. Abbreviated CPU.

channel: (1) * A path along which signals can be sent, for example, data channel, output channel. (2) A hardware device that connects the CPU and real storage with the I/O control units.

channel program translation: In a copy of a channel program, replacement, by software, of virtual addresses with real addresses.

compile: To prepare a machine language program from a computer program written in a high-level language by making use of the overall logic structure of the program, or generating more than one machine instruction for each symbolic statement, or both, as well as performing the function of an assembler.

compiler: A program that translates high-level language statements into machine language instructions.

configuration: The group of machines, devices, etc., which make up a data processing system.

control area: A group of control intervals used as a unit for formatting a file before adding records to it. Also, in a key-sequenced file, the set of control intervals covered by an index record; used by VSAM for distributing free space and for placing a low-level index adjacent to its data.

control interval: A fixed-length area of auxiliary storage space in which VSAM stores records and distributes free space, also, in a key-sequenced file, the set of records pointed to by an entry in the index record. It is the unit of information transmitted to or from auxiliary storage by VSAM, independent of blocksize.

control program: A program that is designed to schedule and supervise the performance of data processing work by a computing system.

control registers: A set of registers used for operating system control of relocation, priority interruption, program event recording, error recovery, and masking operations.

control section: That part of a program specified by the programmer to be a relocatable unit.

control unit: A device that controls the reading, writing, or display of data at one or more input/output devices.

core image library: A library of phases that have been produced as output from link-editing. The phases in the core image library are in a format that is executable either directly or after processing by the relocating loader in the supervisor.

CPU busy time: The amount of time devoted by the central processing unit to the execution of instructions.

data file: A collection of related data records organized in a specific manner. For example, a payroll file (one record for each employee, showing his rate of pay, deductions, etc., or an inventory item, showing the cost, selling price, number in stock, etc.). See also file.

data integrity: See integrity.

data management: A major function of DOS/VS that involves organizing, storing, locating, retrieving, and maintaining data.

data security: See security.

deblocking: The action of making the first and each subsequent logical record of a block available for processing one record at a time.

default value: The choice among exclusive alternatives made by the system when no explicit choice is specified by the user.

deletion of an I/O Device: Removal of the I/O unit from the supervisor configuration tables.

diagnostic routine: A program that facilitates computer maintenance by detection and isolation of malfunctions or mistakes.

dial-up terminal: A terminal on a switched teleprocessing line.

direct access: (1) Retrieval or storage of data by a reference to its location on a volume, other than relative to the previously retrieved or stored data. (2) * Pertaining to the process of obtaining data from, or placing data into, storage where the time required for such access is independent of the location of the data most recently obtained or placed in storage. (3) * Pertaining to a storage device in which the access time is effectively independent of the location of the data. Synonymous with random access.

direct organization: Direct file organization implies that for purposes of storage and retrieval there is a direct relationship between the contents of the records and their addresses on disk storage.

directory: An index that is used by the system control and service programs to locate one or more sequential blocks of program information that are stored on direct access storage.

diskette: A flexible magnetic-oxide coated disk suitable for data storage and retrieval. Data may be stored and retrieved via such devices as the IBM 3740 Data Entry Unit and the IBM 3540 Diskette Input/Output Unit. Diskettes are also used to contain microprograms for some central processing units.

disk pack: A direct access storage volume containing magnetic disks on which data is stored. Disk packs are mounted on a disk storage drive, such as the IBM 3330 Disk Storage Drive.

distributed free space: Space reserved within the control intervals of a key-sequenced file for inserting new records into the file in key sequence; also, whole control intervals reserved in a control area for the same purpose.

dump: (1) To copy the contents of all or part of virtual storage. (2) The data resulting from the process as in (1).

dynamic address translation (DAT): (1) The change of a virtual storage address to an address in real storage during execution of an instruction. (2) A hardware function that performs the translation.

entry sequence: The order in which data records are physically arranged in auxiliary storage, without respect to their contents (contrast with key sequence).

entry-sequenced file: A VSAM file whose records are loaded without respect to their contents, and whose relative byte addresses cannot change. Records are retrieved and stored by addressed access, and new records are added to the end of the file.

error message: The communication that an error has been detected.

error recovery procedures: Procedures designed to help isolate, and, when possible, to recover from errors in equipment. The procedures are often used in conjunction with programs that record the statistics of machine malfunctions.

extent: A continuous space on a direct access storage device, occupied by or reserved for a particular file.

* **file:** A collection of related records treated as a unit. For example, one line of an invoice may form an item, a complete invoice may form a record, the complete set of such records may form a file, the collection of inventory control files may form a library, and the libraries used by an organization are known as its data bank.

fixed page: A page in real storage that is not to be paged out.

hard copy: A printed copy of machine output in a visually readable form, for example, printed reports, listings, documents, and summaries.

hard wait state: In general, a wait state is the condition of a CPU when all operations are suspended. System recovery from a hard wait state requires that the user performs a new IPL (initial program load) procedure.

* **hardware:** Physical equipment, as opposed to the computer program or method of use, for example, mechanical, magnetic, electrical, or electronic devices. Contrast with software.

* **idle time:** That part of available time during which the hardware is not being used.

index: (1) * An ordered reference list of the contents of a file or document, together with keys or reference notations for identification or location of those contents. (2) A table used to locate the records of an indexed sequential file.

indexed-sequential organization: The records of an indexed sequential file are arranged in logical sequence by key. Indexes to these keys permit direct access to individual records. All or part of the file can be processed sequentially.

Initial Program Load (IPL): The intialization procedure that causes DOS/VS to commence operation.

integrity: Preservation of data or programs for their intended purpose.

* **interface:** A shared boundary. An interface might be a hardware component to link two devices or it might be a portion of storage or registers accessed by two or more computer programs.

* **I/O:** An abbreviation for input/output.

ISAM interface program: A set of routines that allow a processing program coded to use ISAM to gain access to a VSAM key-sequenced file with an index.

job: (1) * A specified group of tasks prescribed as a unit of work for a computer. By extension, a job usually includes all necessary computer programs, linkages, files, and instructions to the operating system. (2) A collection of related problem programs, identified in the input stream by a JOB statement followed by one or more EXEC statements.

job accounting interface: A function that accumulates, for each job step, accounting information that can be used for charging usage of the system, planning new applications, and supervising system operation more efficiently.

job control: A program that is called into a virtual partition to prepare each job or job step to be run. Some of its functions are to assign I/O devices to certain symbolic names, set switches for program use, log (or print) job control statements, and fetch the first program phase of each job step.

job (JOB) statement: The job control statement that identifies the beginning of a job. It contains the name of the job.

job step: The execution of a single processing program.

K: 1024.

* **key:** One or more characters associated within an item of data that are used to identify it or control its use.

key sequence: The collating sequence of data records, determined by the value of the key field in each of the data records. May be the same as, or different from, the entry sequence of the records.

key-sequenced file: A file whose records are loaded in key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in the file in key sequence by means of distributed free space. Relative byte addresses of records can change.

label: identification record for a tape, diskette, or disk file.

label information cylinder(s): Under DOS/VS, the last one or two cylinder(s) of the system residence file that stores label information read from job control statements or commands. Synonymous with label cylinder(s).

language translator: A general term for any assembler, compiler, or other routine that accepts statements in one language and procedures equivalent statements in another language.

leased facility: A circuit of the public telephone network made available for the exclusive use of one subscriber.

librarian: The set of programs that maintains, services, and organizes the system and private libraries.

library: A collection of files or programs, each element of which has a unique name, that are related by some common characteristics. For example, all phases in the core image library have been processed by the linkage editor.

linkage editor: A processing program that prepares the output of language translators for execution. It combines separately produced object modules; resolves symbolic cross references among them, and generates overlay structure on request; and produces executable code (a phase) that is ready to be fetched or loaded into virtual storage.

load: (1) * In programming, to enter instructions or data into storage or working registers. (2) In DOS/VS, to bring a program phase from a core image library into virtual storage for execution.

main page pool: The set of all page frames in real storage not assigned to the supervisor or one of the real partitions.

message: See error message, operator message.

microprogramming: A method of working of the CPU in which each complete instruction starts the execution of a sequence of instructions, called microinstructions, which are generally at a more elementary level.

multiprogramming system: A system that controls more than one program simultaneously by interleaving their execution.

multitasking: The concurrent execution of one main task and one or more subtasks in the same partition.

object code: Output from a compiler or assembler which is suitable for processing by the linkage editor to produce executable machine code.

* **object module:** A module that is the output of an assembler or compiler and is input to a linkage editor.

object program: A fully compiled or assembled program. Contrast with source program.

* **online:** (1) Pertaining to equipment or devices under control of the central processing unit. (2) Pertaining to a user's ability to interact with a computer.

operand: (1) * That which is operated upon. An operand is usually identified by an address part of an instruction. (2) Information entered with a command name to define the data on which a command processor operates and to control the execution of the command processor.

operator command: A statement to the control program, issued via a console device, which causes the control program to provide requested information, alter normal operations, initiate new operations, or terminate existing operations.

operator message: A message from the operating system or a problem program directing the operator to perform a specific function, such as mounting a tape reel, or informing him of specific conditions within the system, such as an error condition.

* **overflow:** (1) That portion of the result of an operation that exceeds the capacity of the intended unit of storage. (2) Pertaining to the generation of overflow as in (1).

overlay: n. (1) One of the segments, which consists of one or more phases, of a program that is so structured that not all of the segments need be in virtual storage at any one time. v. (2) The process of replacing a previously retrieved program segment in virtual storage by another segment.

page: (1) In DOS/VS, a 2K block of instructions, data or both. (2) To transfer instructions, data, or both between real storage and the page data set.

page data set: An extent in auxiliary storage, in which pages are stored.

page fault: A program check interruption that occurs when a page that is marked *not in real storage* is referred to by an active page. Synonymous with page translation exception.

page fixing: Marking a page as nonpageable so that it remains in real storage.

page frame: A 2K block of real storage that can contain a page.

page in: The process of transferring a page from the page data set to real storage.

page out: The process of transferring a page from real storage to the page data set.

page pool: The set of all page frames that may contain pages of programs in virtual mode.

paging: The process of transferring pages between real storage and the page data set.

* **parameter:** A variable that is given a constant value for a specific purpose or process.

peripheral equipment: A term used to refer to card devices, magnetic tape and disk devices, diskettes, printers, and other equipment bearing a similar relation to the CPU.

phase: The smallest complete unit that can be referred to in the core image library.

POWER: *P*riority *O*utput *W*riters, *E*xecution Processors and Input *R*eaders.

printer: A device that expresses coded characters as hard copy.

priority: A rank assigned to a partition that determines its precedence in receiving CPU time.

private library: A user-owned library that is separate and distinct from the system library.

private second level directory: The private second level directory is a table located in the supervisor containing the highest phase names found on the corresponding directory tracks of the private core image library.

problem determination aid: A program that traces a specified event when it occurs during the operation of a program. Abbreviated PDAID.

problem program: Any program that is executed when the central processing unit is in the problem state; that is, any program that does not contain privileged instructions. This includes IBM-distributed programs, such as language translators and service programs, as well as programs written by a user.

processing program: (1) A general term for any program that is not a control program. (2) Synonymous with problem program.

processor storage: The general purpose storage of a computer. Processor storage can be accessed directly by the operating registers. Synonymous with real storage.

queue: (1) A waiting line or list formed by items in a system waiting for service; for example, tasks to be performed or messages to be transmitted in message switching system. (2) To arrange in, or form, a queue.

random processing: The treatment of data without respect to its location in auxiliary storage, and in an arbitrary sequence governed by the input against which it is to be processed.

real address; The address of a location in real storage.

real address area: In DOS/VS, the area of virtual storage where virtual addresses are equal to real addresses.

real mode: In DOS/VS, the mode of a program that cannot be paged.

real partition; In DOS/VS, a division of the real address area of virtual storage that may be allocated for programs that are not to be paged, or virtual programs that contain pages that are to be fixed.

real storage: The storage of a System/370 computing system from which the central processing unit can directly obtain instructions and data, and to which it can directly return results. Synonymous with processor storage.

reenterable: The attribute of a load module that allows the same copy of the load module to be used concurrently by two or more tasks.

relocatable: The attribute of a set of code whose address constants can be modified to compensate for a change in origin.

relocatable library: A library of relocatable object modules and IOCS modules required by various compilers. It allows the user to keep frequently used modules available for combination with other modules without recompilation.

restore: To return a data file created previously by a copy operation from cards, disk or magnetic tape to disk storage.

rotational position sensing (RPS): A standard or optional feature of most IBM disk storage devices. It permits these devices to disconnect from a block multiplexer channel (or its equivalent on Model 3115/3125 CPUs)

during rotational positioning operations, thereby allowing the channel to service other devices.

* **routine:** An ordered set of instructions that may have some general or frequent use.

secondary storage: Same as auxiliary storage.

second level directory: A table located in the supervisor containing the highest phase names found on the corresponding directory tracks of the system core image library.

security: Prevention of access to or use of data or programs without authorization.

sequential organization: Records of a sequential file are arranged in the order in which they will be processed.

service program: A program that assists in the use of a computing system, without contributing directly to the control of the system or the production of results.

shared virtual area: An area located in the highest addresses of virtual storage. It can contain a system directory list of highly used phases, resident programs that can be shared between partitions, and an area for system GETVIS support.

software: A set of programs, concerned with the operation of the hardware in a data processing system.

source: The statements written by the programmer in any programming language with the exception of actual machine language.

* **source program:** A computer program written in a source language. Contrast with object program.

source statement library: A collection of books (such as macro definitions) cataloged in the system by the librarian program.

spanned records: Records of varying length that may be longer than the currently used blocksize, and which may therefore be written in one or more continuous blocks. A spanned record may occupy more than 1 track of a disk device.

stand-alone dump: A program that displays the contents of the registers and part of the real address area and that runs independently and is not controlled by DOS/VS.

standard label: A fixed-format identification record for a tape, diskette, or disk file. Standard labels can be written and processed by DOS/VS.

storage protection: An arrangement for preventing access to storage.

supervisor: A component of the control program. It consists of routines to control the functions of program loading, machine interruptions, external interruptions, operator communications and physical IOCS requests and interruptions. The supervisor alone operates in the privileged (supervisor) state. It coexists in real storage with problem programs.

switched line: A communication line in which the connection between the computer and a remote station is established by dialing. Synonymous with dial line.

system directory list: A list containing directory entries of highly used phases and of all phases resident in the shared virtual area. This list is contained in the shared virtual area.

system residence device: The direct access device on which the system residence file is located.

system residence volume: The volume on which the basic system and all related supervisor code is located.

task: A unit of work for the central processing unit from the standpoint of the control program.

teleprocessing: The processing of data that is received from or sent to remote locations by way of telecommunication lines.

terminal: (1) * A point in a system or communication network at which data can either enter or leave. (2) Any device capable of sending and receiving information over a communication channel.

throughput: The total volume of work performed by a computing system over a given period of time.

track: The portion of a moving storage medium, such as a drum, tape, diskette, or disk, that is accessible to a given reading head position.

transient area: An area of real storage used for temporary storage of transient routines.

UCS: Universal character set.

unit record: A card containing one complete record; a punched card.

universal character set: A printer feature that permits the use of a variety of character arrays. Abbreviated UCS.

unrecoverable error: A hardware error which cannot be recovered from by the normal retry procedures.

user label: An identification record for a tape or disk file; the format and contents are defined by the user, who must also write the necessary processing routines.

utility program: A problem program designed to perform a routine task, such as transcribing data from one storage device to another.

virtual address: An address that refers to virtual storage and must, therefore, be translated into a real storage address when it is used.

virtual address area: In DOS/VS, the area of virtual storage whose addresses are greater than the highest address of the real address area.

virtual mode: In DOS/VS, the mode of execution of a program which may be paged.

virtual partition: In DOS/VS, a division of the virtual address area of virtual storage that is allocated for programs that may be paged.

virtual storage: Addressable space that appears to the user as real storage, from which instructions and data are mapped into real storage locations. The size of virtual storage is limited by the addressing scheme of the

computing system and by the capacity of the page data set, rather than by the actual number of real storage locations.

virtual storage access method (VSAM): VSAM is an access method for direct or sequential processing of fixed and variable length records on direct access devices. The records in a VSAM file can be organized either in logical sequence by a key field (key sequence) or in the physical sequence in which they are written on the file (entry-sequence). A key sequenced file has an index, an entry-sequenced file does not.

virtual telecommunications access method (VTAM): VTAM is an access method that supports communication between application programs and terminals in a telecommunications network.

volume: (1) That portion of a single unit of storage media which is accessible to a single read/write mechanism, for example, a drum, a diskette, a disk pack, or part of a disk storage module. (2) A recording medium that is mounted and dismounted as a unit, for example, a reel of magnetic tape, a disk pack, a diskette, or a data cell.

volume table of contents: A table on a direct access volume or diskette that describes each file on the volume. Abbreviated VTOC.

VSAM access method services: A multifunction utility program that defines VSAM files and allocates space for them, converts indexed sequential files to key-sequenced files with indexes, facilitates data portability between operating systems, creates backup copies of files and indexes, helps to make inaccessible files accessible, and lists file and catalog entries.

VSAM catalog: A key-sequenced file, with an index, containing extensive file and volume information that VSAM requires to locate files, to allocate and deallocate storage space, to verify the authorization of a program or operator to gain access to a file, and to accumulate usage statistics for files.

VTOC: See volume table of contents.

work file: A file on an auxiliary storage medium reserved for intermediate results during execution of the program.

working set: The set of pages of a user's virtual-mode program that must be in real storage in order to avoid excessive paging.

Index

D

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

GC33-5371-6

DOS/VS System Management Guide Printed in U.S.A. GC33-5371-6

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

DOS/VS
System Management Guide

GC33-5371-6

READER'S COMMENT FORM

This sheet is for comments and suggestions about this manual. We would appreciate *your* views, favorable or unfavorable, in order to aid us in improving *this* publication. This form will be sent directly to the author's department. Please include your name and address if you wish a reply. Contact your IBM branch office for answers to technical questions about the system or when requesting additional publications. Thank you.

Name
Address

What is your occupation?

How did you use this manual?
As a reference source
As a classroom text
As a self-study text

Your comments* and suggestions:

*** We would especially appreciate your comments on any of the following topics:**

Clarity of the text	Accuracy	Index	Illustrations	Appearance	Paper
Organization of the text	Cross-references	Tables	Examples	Printing	Binding

YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold

Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
1133 Westchester Avenue
White Plains, N.Y. 10604

Attention: Department 813 BP

Fold

Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

CUT ALONG THIS LINE

DOS/VS System Management Guide Printed in U.S.A. GC33-5371-6

GC33-5371-6

DOS/VS System Management Guide Printed in U.S.A. GC33-5371-6

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)