GC28-0600-2

Systems

OS/VS2 Planning and Use Guide

VS2 Release 1

IBM

Second Edition (September, 1972)

This is a major revision of, and obsoletes, GC28-0600-1. Changes or additions to the text and illustrations are indicated by a vertical bar to the left of the change.

This edition applies to release 1 of OS/VS2 and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest **IBM System/360 and System/370 Bibliography,** GA22-6822, and the current **SRL Newsletter,** GN20-0360, 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 Corporation, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, N.Y. 12602. Comments become the property of IBM.

Preface

This publication describes OS/VS2. The purpose of the publication is to introduce VS2 concepts and provide planning information for users preparing to have VS2 installed. The publication assumes basic knowledge of OS/MVT. (MVT is described in **IBM System/360 Operating System: MVT Guide,** GC28-6720.)

This publication is divided into the following chapters:

- Introduction -- This chapter presents an overview of VS2. The chapter also discusses the advantages of a virtual storage system, the new functions provided by VS2, and the basic configuration and supported devices of the system.
- System Control Program -- This chapter presents an overview of the five major components of the VS2 system control program: job management, task management, input/output supervision, data management, and recovery management. For each item discussed, a summary of the major changes from MVT is provided.
- Standard Support Programs -- This chapter presents an overview of VS2 standard support programs (such as utilities, service aids, and linkage editor) that interact with the system control program. For each item discussed, a summary of the major changes from MVT is provided.
- Options -- This chapter presents an overview of the optional facilities that can be added to the VS2 system during and after system generation. For each item discussed, a summary of the major changes from MVT is provided.
- Compatibility -- This chapter describes the differences and incompatibilities that exist between VS2 and MVT. Where applicable, differences and incompatibilities that apply to MFT and VS1 are also discussed. The chapter provides basic information that the user needs in converting from MVT to VS2.
- Defining the System -- This chapter provides preliminary planning information that the user needs in defining his system. For example, the chapter describes the VS2 system generation procedures and macro instructions, and the VS2 system initialization procedures and parameters.
- Job Management and Supervisor Services for System Programmers -- This chapter describes how to modify, extend, or implement various facilities of the control program. It is designed primarily for system programmers responsible for maintaining, updating, and extending the system facilities.
- Supervisor Macro Instructions for System Programmers -- This chapter describes supervisor macro instructions and parameters that are restricted to authorized programs, programs executing in the supervisor state, or programs operating under protection key zero. It is designed primarily for system programmers responsible for maintaining, updating, and extending the system facilities.
- System Overview -- This chapter graphically and concisely explains the job, task, data, and recovery management components of the VS2 control program to both the new and the experienced user. This chapter refers to other IBM publications that contain detailed information suitable for the experienced user.
- Glossary -- This chapter defines new VS2 terms used in this publication. The chapter does not define data processing and MVT terms commonly in use.

The following items are described in this publication for planning purposes only, and are not available with the initial release of VS2:

- Dynamic support system (DSS)
- Virtual storage access method (VSAM)

Availability dates for support of the above items may be obtained from the local IBM branch office.

Related Publications

The following publications contain general information on OS/VS.

Introduction to Virtual Storage in System/370, GR20-4260

This publication provides an overview of the advantages and concepts of virtual storage systems.

IBM System/370 System Summary, GA22-7001

This publication provides an overview of the system concepts and features of System/370, including a discussion of the components comprising OS/VS.

IBM Data Processing Glossary, GC20-1699
This publication defines terms and concepts used in IBM publications.

OS/VS1 Planning and Use Guide, GC24-5090
This publication describes OS/VS1. It can assist installation personnel in selecting and evaluating VS1, and can assist system programmers in implementing, modifying, and extending the capabilities of the VS1 control program.

In this publication, the following publication are referred to for more information on specific subjects:

OS TCAM Concepts and Facilities, GC30-2022

OS/VS BTAM, GC27-6980

OS/VS Checkpoint/Restart, GC26-3784

OS/VS Data Management for System Programmers, GC28-0631

OS/VS Dynamic Support System, GC28-0640

OS/VS JCL Reference, GC28-0618

OS/VS Linkage Editor and Loader, GC26-3813

OS/VS Message Library: VS2 System Messages, GC38-1002

OS/VS OLTEP, GC28-0636

OS/VS RDE Guide, GC28-0642

OS/VS Service Aids, GC28-0633

OS/VS Supervisor Services and Macro Instructions, GC27-6979

OS/VS System Generation Introduction, GC26-3790

OS/VS System Management Facilities, GC35-0004

OS/VS SYS1.LOGREC Error Recording, GC28-0638

OS/VS TCAM Programmer's Guide, GC30-2034

OS/VS Virtual Storage Access Method (VSAM) Planning Guide, GC26-3799

OS/VS Utilities, GC35-0005

OS/VS and DOS/VS Assembler Language, GC33-4010

OS/VS2 Debugging Guide, GC28-0632

OS/VS2 System Generation Reference, GC26-3792

OS/VS2 TSO Command Language Reference, GC28-0646

OS/VS2 TSO Guide, GC28-0644

Operator's Library: OS/VS2 Reference, GC38-0210

System/370 Program Product Language and Sort Processors, GC28-8200

This publication also refers to the following logic manuals:

OS/VS Graphic Access Method Logic, SY27-7240

OS/VS I/O Supervisor Logic, SY24-3823

OS/VS Open/Close/EOV Logic, SY26-3785

OS/VS Recovery Management Support Logic, SY27-7252

OS/VS TCAM Logic, SY30-2039

OS/VS2 IPL and NIP Logic, SY27-7243

OS/VS2 Job Management Logic, SY28-0620

OS/VS2 Supervisor Logic, SY27-7244

OS/VS2 System Data Areas, SY28-0606

OS/VS2 TSO Control Program Logic, SY28-0649

Contents

Figures

Diagrams

Introduction

Operating System/Virtual Storage 2 (VS2) is a compatible extension of OS/MVT (multiprogramming with a variable number of tasks) that is specially modified to take advantage of relocation hardware and the extended control features of System/370.

Virtual Storage

One of the primary differences between VS2 and MVT is that VS2 provides 16,777,216 bytes of addressable space, called virtual storage. It is addressable space that appears to the user's program as real storage. When a user's instructions and data are ready for processing by the CPU, they are brought into real storage locations.

When instructions and data are said to be in "virtual storage", they are actually in a portion of auxiliary storage called external page storage. When they are needed for execution, the instructions and data are loaded into real storage through a process called paging. The data and instructions are contained in 4K-byte blocks of storage, called pages.

In VS2, the term real storage is used for what was called main storage in MVT; that is, real storage is the storage of System/370 from which the central processing unit directly obtains instructions and data, and to which it directly returns results. Instructions and data are brought into real storage from virtual storage only when they are actually required by an executing program, and altered instructions and data are returned to virtual storage when they are no longer being accessed. Therefore, at any given time, real storage contains only a portion of the total contents of virtual storage. The relationships between virtual storage, real storage, and external page storage are illustrated in Figure 1.

The map of virtual storage in VS2 resembles that of main storage in MVT, except that the addressable space is enlarged. Figure 2 shows the similarities. In both systems, the dynamic area (from which regions are allocated) occupies the middle of storage, while each end is used for programs established at IPL time. In VS2, the virtual storage areas that correspond to the fixed areas of main storage in MVT are called the nondynamic areas.

Theoretically, the size of virtual storage is limited only by the addressing scheme of the computing system. (In System/370, a 24-bit address is used, allowing up to 16,777,216 bytes of addressable storage.) In reality, the size of virtual storage is limited by the amount of auxiliary storage available in the system. This concept is illustrated in Figure 3.

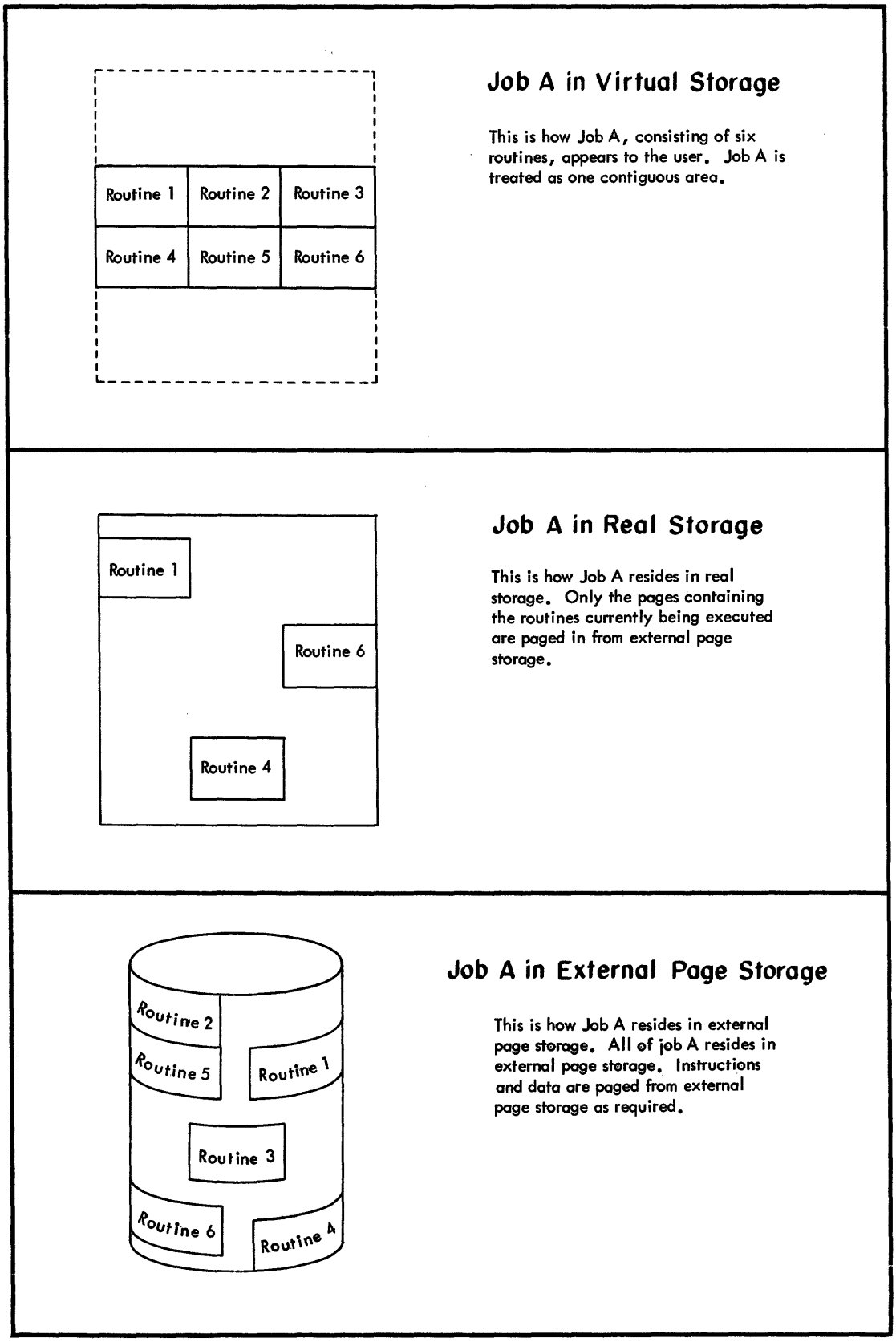


Figure 1. Relationship Between Virtual Storage, Real Storage, and External Page Storage

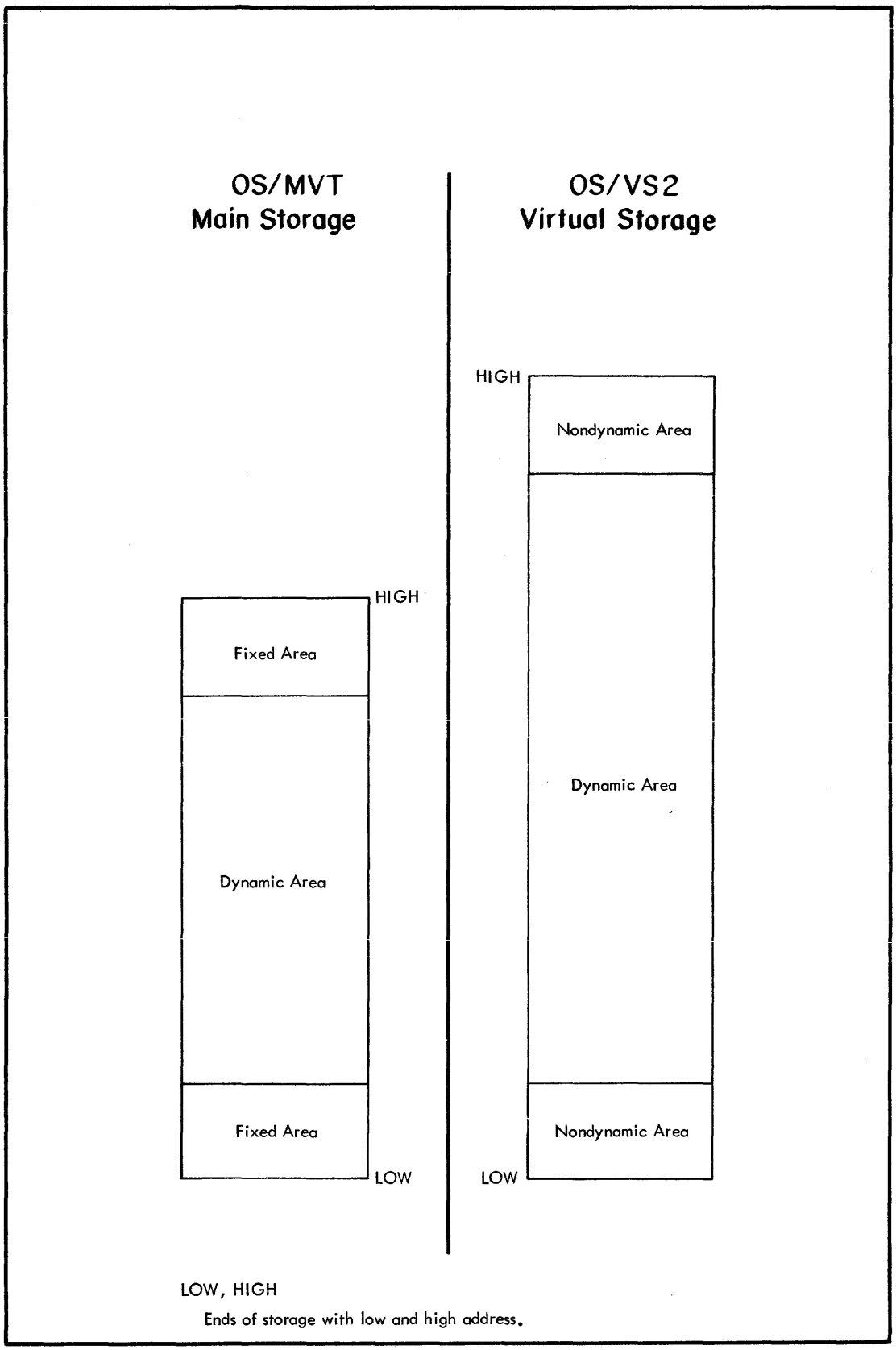


Figure 2. MVT and VS2 Storage Overviews

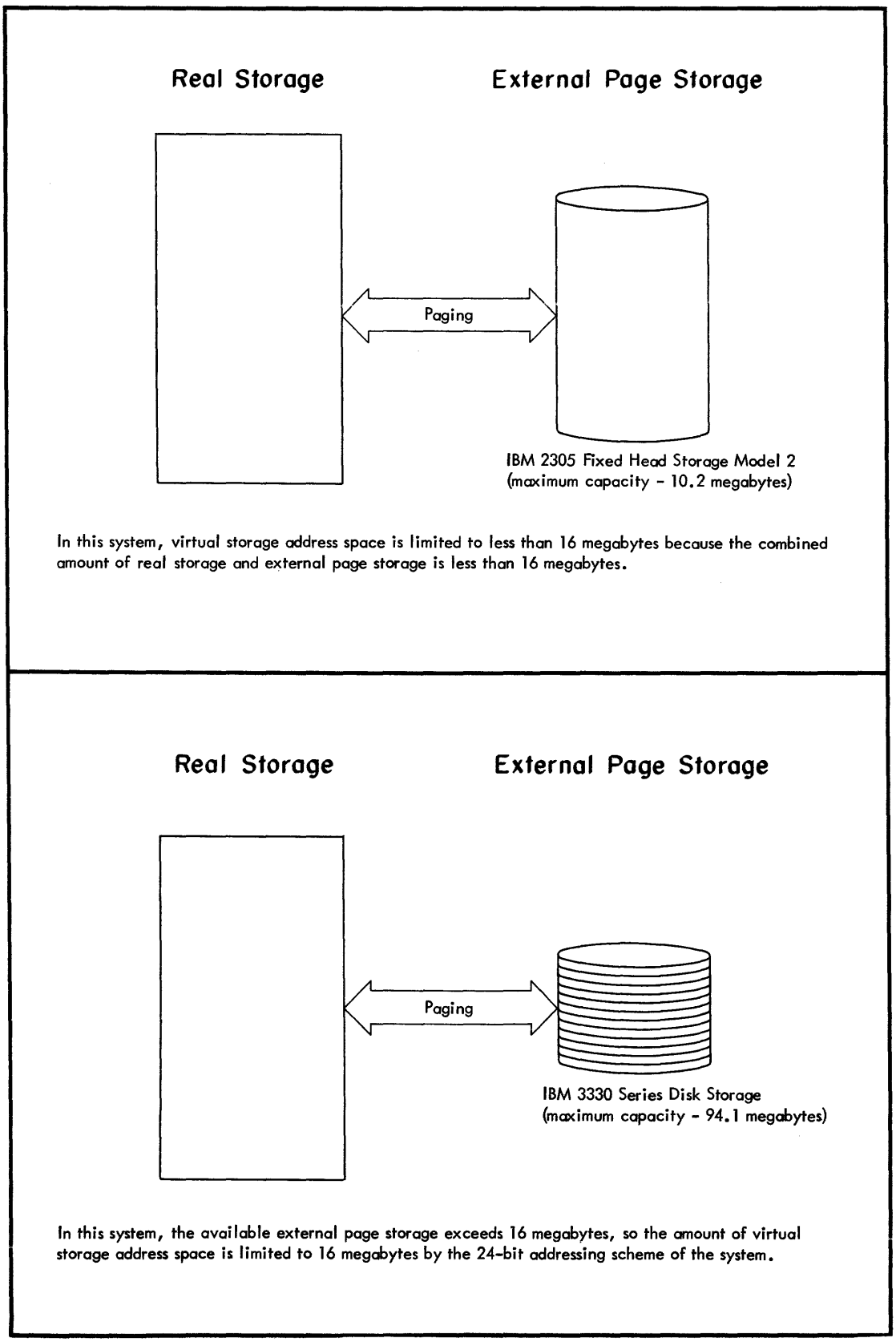


Figure 3. Relationship Between Virtual Storage Address Space and Available External Page Storage

New Functions

In addition to the benefits of using a virtual storage system (described in the following section), VS2 provides new functional enhancements over MVT. They include:

- Simplified system generation in which IBM performs various functions (including installation verification procedures) previously performed by the user.
- Additional standard facilities in VS2 that were previously optional:

  Basic direct access method (BDAM)
  Channel check handler (CCH)
  Checkpoint/restart
  Dynamic device reconfiguration (DDR)
  Hardcopy log
  Job step timing
  Machine check handler (MCH)
  Missing interruption checker
  Multiple console support (MCS)
  Online test executive program (OLTEP)
  PCI fetch
  System log
  System management facilities (SMF)
  Volume statistics
- Additional parameter lists that can be specified during system generation and used at initialization time.
- New debugging tool for authorized maintenance personnel (dynamic support system) to help correct causes of software failures.
- New timing facilities (CPU timer and clock comparator).
- Enhanced I/O load balancing.
- New dispatching facility to provide more efficient use of CPU and I/O resources (automatic priority group).
- Improved system security and integrity (authorized program facility, fetch protection. data extent block validity checking, local system queue area, missing interruption checker).
- Enlarged link pack area (fixed and pageable) that contains reenterable routines used concurrently by all tasks in the system.
- Operator capability to cancel a job waiting for a region or a data set.
- New optional access method (virtual storage access method).
- Enhanced support programs (VS2 assembler, linkage editor, IEBCOPY utility program).
- Enhanced system management facilities (SMF).

Compatibility

VS2 is an extension of MVT; VS1 is an extension of MFT. Although the major thrust for compatiblility discussed in this publication is between VS2 and MVT, VS2 is also an extension of VS1. Compatibility between MVT, MFT, VS2, and VS1 is discussed in detail in the chapter "Compatibility".

The main compatibility objective of VS2 is that all problem programs that run under MVT also run under VS2. In VS2, a part of virtual storage -- the nonpageable dynamic area -- was established to accommodate those programs that can not be executed in a virtual storage environment. Another objective is that capabilities added to VS2 not affect currently existing interfaces between problem programs and the operating system.

The known incompatibilities that do exist, and the modifications necessary to compensate for them, are described later in this publication. The following list briefly states the incompatibilities:

- MVT operator commands contain some new parameters, and some old parameters that should not be specified because they specify nonsupported functions. In several cases, the output generated by the commands has changed.
- TSO operator commands START and MODIFY contain both new parameters and changes to the existing parameters; also, all operator parameters now contain unique abbreviations.
- Job control language (JCL) is basically unchanged, and all jobs that run in MVT can be executed in VS2 without modification to the JCL.
- All problem programs that execute under Release 21 of MVT also execute, with no modifications, under VS2.
- Reassembly/recompilation of programs is unnecessary except for users of TCAM message control programs and users of programs which have unique system dependencies.
- System data sets, except for three new data sets (SYS1.LPALIB, SYS1.PAGE, and SYS1.DSSVM), are the same as their MVT counterparts.
- Certain major functions in MVT are not supported in VS2. See Figure 4.

MVT Function Not Supported	Comparable VS2 Function
Automatic SYSIN batching (ASB) reader	---
Conversational remote job entry (CRJE)	Time sharing option (TSO)
Direct system output (DSO) writer	---
Graphic job processor (GJP)	---
IEBUPDAT utility program	IEBUPDTE utility program
IEHIOSUP utility program	Pageable link pack area
Main storage hierarchy support	---
Multiprocessing (MP65)	---
Queued telecommunications access method (QTAM)	Telecommunications access method (TCAM)
Remote job entry (RJE)	---
Rollout/rollin	Paging
Satellite graphic job processor (SGJP)	---
Scatter load	Paging
System environment recording routines (SER0 and SER1)	Recovery management support
TESTRAN program	---
Transient areas	Pageable link pack area

Figure 4. Major MVT Functions Not Supported in VS2

Planning aids for defining a system, optimizing the performance of a system, and modifying a system are vital to successfully meeting the needs of an installation.

In VS2, the process of defining the system has been made easier by the improved system generation and system initialization facilities. These processes are discussed in the chapter "Defining the System".

The performance of a system depends to a large extent on the specifications of job classes, job priorities, and system output classes. In VS2, these facilities are the same as their counterparts in MVT.

Modification of a system, including maintenance and updating, is primarily the responsibility of the system programmers at an installation. Except for those MVT facilities that are not supported in VS2 or are no longer needed, the external implementation of the MVT facilities is

the same for VS2. The facilities that fall into this category are listed below, and are described later in this publication.

- Cataloged procedures.
- System output writers.
- Job queue formatting.
- Output separation.
- Message routing exit routines.
- PRESRES volume characteristics list.
- Must complete function.
- Shared direct access storage devices.
- Time slicing.

Advantages of VS2

This chapter has already briefly described the functional enhancements VS2 provides over MVT. However, the most important new feature of VS2 is its support of a virtual storage environment. The advantages that can result from using a virtual storage system are described below.

System Flexibility

In scheduling, installations strive to match work to be done against available real storage and input/output devices. In VS2, virtual storage can make this installation preplanning easier. The result is:

- Less planning to prevent storage fragmentation. In systems without virtual storage, storage fragmentation was a primary consideration when selecting jobs to be multiprogrammed. Detailed planning of job mixes to avoid storage fragmentation in real storage is no longer necessary. (In virtual storage, some fragmentation will exist, but its effect will be minimal.)
- More flexible backup. Since a small CPU can be used to execute programs designed to run on a larger CPU, storage size is now less of a consideration when planning for a backup CPU.
- More efficient use of system resources. Since real storage is obtained and released dynamically, only the real storage that is needed is tied to a job currently executing. In systems without virtual storage, all real storage belonging to a job could not be reassigned by the system until the end of the job.
- Easier handling of priority jobs. In systems without virtual storage, a high priority job would have to to wait to be executed if there was insufficient real storage to contain the job (for example, if large production jobs were occupying most of storage). In VS2, virtual storage space increases the probability that high priority jobs can begin execution without waiting for completion of the large, low priority jobs.

System Throughput

Throughput is the total volume of work performed by a computing system over a given period of time. In VS2, virtual storage management has considerable influence on the system throughput. As a result, the following enhancements may result in improved system throughput:

- More system functions that can be shared by all users are resident in virtual storage. Resident reenterable functions eliminate duplicate copies of code, reduce the need to access system libraries, and save the time required to load these functions with each program.

- More job steps can be initiated, thus achieving a higher degree of multiprogramming. Because of virtual storage and demand paging, entire jobs do not occupy real storage at any one time.
- Better balance in the use of the CPU and the input/output devices is achieved. In VS2, the optional dynamic dispatching facility automatically allocates varying slices of time to jobs having the same dynamic dispatching priority, giving preference to I/O-bound jobs over CPU-bound jobs.
- More regions can be started because of the larger addressable space. Thus, if the time sharing option is included in the system, the added regions can be used to serve more terminals.

Programmer Productivity

The additional virtual storage available with VS2 helps to reduce many of the former constraints of program design, thus allowing the programmer to concentrate on applications. As a result:

- More addressable space is available for programs. Since jobs requiring more real storage than is actually available can run in VS2, storage constraints have been eased for the programmer. In most cases, programming practices such as overlays have been made unnecessary.
- Programs with large storage requirements can be tested on machines with small real storage. Since programs are loaded into virtual storage and only parts of the programs are paged on demand into real storage, applications intended for large machines can be tested on smaller machines.
- Programs can be changed more easily. Since storage size is not a primary concern of the programmer, he can avoid intricate coding when changing a program.

System Integrity

Integrity is preservation of data or programs for their intended purpose. In VS2, system integrity has been improved in the following ways:

- The authorized program facility (APF) limits the use of sensitive system and (optionally) user services and resources to authorized system and user programs. The authorization consists of a code that is used with programs residing in the password protected SYS1.LINKLIB and SYS1.SVCLIB data sets, and in the link pack area. The SYS1.LINKLIB, SYS1.LPALIB, and SYS1.SVCLIB data sets are the only data sets in which an authorized program can reside. For a complete description of APF, see the chapter "Job Management and Supervisor Services for System Programmers".

  A new system macro instruction, TESTAUTH, has been defined to support APF. The macro instruction tests the authorization of the caller and informs the caller if it is authorized to perform a particular function. For a complete description of the TESTAUTH macro instruction see the chapter "Supervisor Macro Instructions for System Programmers".
- The data extent block (DEB) validity checking facility prevents a user from unauthorized access to data on an external device and alleviates several possibilities of transferring control in the supervisor state to an unauthorized routine.

  A new system macro instruction, DEBCHK, has been defined to support DEB validity checking. The macro instruction is used to verify that a DEB is valid. For a complete description of the DEBCHK macro instruction, see *OS/VS Data Management for System Programmers,* GC28-0631.

- The local system queue area (LSQA) prevents excessive use of system queue area (SQA) space. The LSQA consists of one or more segments that are associated with each virtual storage region and contain job-related system control blocks. The isolation of job-related system control blocks makes it less likely for job failures to cause system failures. The LSQA is protected via read-only access from the problem program.
- The storage protection feature, consisting of both store and fetch protection, prevents unauthorized or unintentional access to virtual or real storage by other than the intended user. (See the discussion of storage protection later in this chapter.)

VS2 Overview

The functional capability of VS2 consists of the MVT functional base which has been extended to take advantage of virtual storage. The virtual storage concept is implemented through facilities called address translation and paging.

Address Translation

In VS2, references in a program to virtual storage locations must be translated into references to real storage locations. The process of changing the address of a data item or an instruction to its real storage address is called address translation. In VS2, the dynamic address translation (DAT) facility of the System/370 machines performs this function.

During execution of a program, each virtual storage address is translated as the instruction is executed. Translation occurs only when the central processing unit is operating in translation mode; that is, bit 5 of the extended control (EC) mode program status word (PSW) is 1. Storage addresses referenced by channels for input/output operations are not translated by the DAT feature; they are translated by the I/O supervisor.

New program interruptions indicate when the translation process cannot be completed. They are:

- Page translation exception, which occurs when a virtual address cannot be translated by the hardware because the invalid bit in the page table entry for that address is set. The page table indicates whether a page is in real storage and correlates virtual addresses with real storage addresses.
- Segment translation exception, which occurs when a virtual address cannot be translated by the hardware because the invalid bit in the segment table entry for that address is set. The segment table is used to control user access to virtual storage segments.
- Translation specification exception, which occurs when a page table entry, segment table entry, or the control register pointing to the segment table contains information in an invalid format.

Paging

A program can be executing even though some of its pages are not in real storage. The specific pages needed in real storage at any given time depend upon the particular program being executed. The process of transferring pages between real storage and external page storage (the portion of auxiliary storage used to contain pages) is called paging.

When an instruction is executed and addresses are translated, an interruption occurs if a page is referred to and it is not in real storage. The paging supervisor brings the page into real storage from an external page storage device. (The paging supervisor is discussed in the chapter "System Control Program".)

When real storage is needed for a page being paged in from external page storage the real storage will be made available by the paging supervisor. If a page in real storage was modified during execution, it is written out to an external storage device to make room for the required page. If a page in real storage was not modified and an exact copy of the page already exists on an external storage device, then the page need not be written out; the page awaiting real storage space is loaded into real storage overlaying the unmodified page.

The system sometimes commits more real storage that can be supported efficiently, and excessive paging occurs in an effort to satisfy that commitment. If the paging rate becomes excessive, there is a constant request queue at the paging device, and active programs resident in storage become idle, waiting for service to their paging requests. This condition, called thrashing, results in little useful work being done. To control thrashing, the dispatcher selects and marks tasks nondispatchable. When the paging rate subsides, the tasks that were marked nondispatchable are reset to dispatchable status.

Some programs cannot be paged. For example, programs that modify channel programs while they are executing cannot be paged since the I/O supervisor duplicates and uses a copy of the channel command words (CCWs) as part of its translation process; any changes made to the original CCW during execution would not be known to the I/O supervisor.

Programs that are highly time dependent (such as the magnetic ink character recognition programs) cannot be paged because the time required to translate the channel programs cannot be handled in the amount of time available. Such programs must be run in an area of virtual storage that has the same range of addresses as real storage, called nonpageable dynamic storage (or virtual equals real or V=R storage).

Storage Maps

Figure 5 shows a map of virtual storage. Figure 6 shows a map of real storage. The maps are divided into the following areas:

System Queue Area - The system queue area is an area of virtual storage reserved for control blocks not related to jobs and job steps and tables (such as the segment tables) maintained by the control program. The amount of fixed real storage required to back up the system queue area varies with the demands of the system.

Pageable Link Pack Area - The pageable link pack area is an area of virtual storage that contains reenterable routines that can be used concurrently by all tasks in the system. As needed, parts of this area are paged into real storage. The fixed link pack area, described below, is an extension of this area.

Pageable BLDL Table - The BLDL table can be either fixed or paged, but not both. (The fixed BLDL table is described below.) The pageable BLDL table occupies an area in the upper portion of virtual storage. It consists of the list of entries for the SYS1.LINKLIB data set that are to be paged into real storage as needed.

Master Scheduler Region - The master scheduler region is an area of virtual storage that contains the master scheduler routine. As needed, parts of this area are paged into real storage.

Master Scheduler Local System Queue Area - The master scheduler LSQA is an area of virtual storage that contains system control blocks (such as the task control block) for the master scheduler. At least one page of this area always remains in real storage.

Pageable Dynamic Area - The pageable dynamic area is an area of virtual storage whose virtual storage addresses are not necessarily identical to real storage addresses. It is used for programs that can be paged during execution. Reader/interpreter regions, system output writer regions, initiator/terminator regions, regions for pageable problem programs, and LSQAs for each of these types of regions are all allocated from the pageable dynamic area.

Nonpageable Dynamic Area - The nonpageable dynamic area (also called virtual equals real or V=R storage) is an area of virtual storage whose virtual storage addresses are identical to real storage addresses. It is used for programs that are not to be paged during execution. If the virtual storage in this area is not assigned for nonpageable programs, the real storage with the same storage addresses is available for paging. The minimum size of this area is 64K bytes. (Considerations for using this area are described in the chapter "Job Management and Supervisor Services for System Programmers".)

Fixed BLDL Table - The BLDL table can be either fixed or paged, but not both. The fixed BLDL table is present only if the user specifies it to occupy an area in the lower portion of virtual storage. It consists of the list of entries for the SYS1.LINKLIB data set that are to remain in real storage.

Fixed Link Pack Area - The fixed link pack area is present only if the user specifies it at IPL time. It is an extension of the pageable link pack area and occupies an area in the lower portion of virtual storage. It is used for frequently used reenterable routines which are to remain in real storage.

Nucleus - The nucleus, mapped into virtual storage at location zero, consists of those system programs which must remain in real storage while the system is in use.

Virtual storage is divided into 256 segments of 64K bytes each. Each segment is divided into 16 pages of 4K bytes each. Regions in the pageable area of virtual storage are allocated in multiples of 64K bytes (segments); regions in the nonpageable dynamic area of virtual storage are allocated in multiples of 4K bytes (pages).

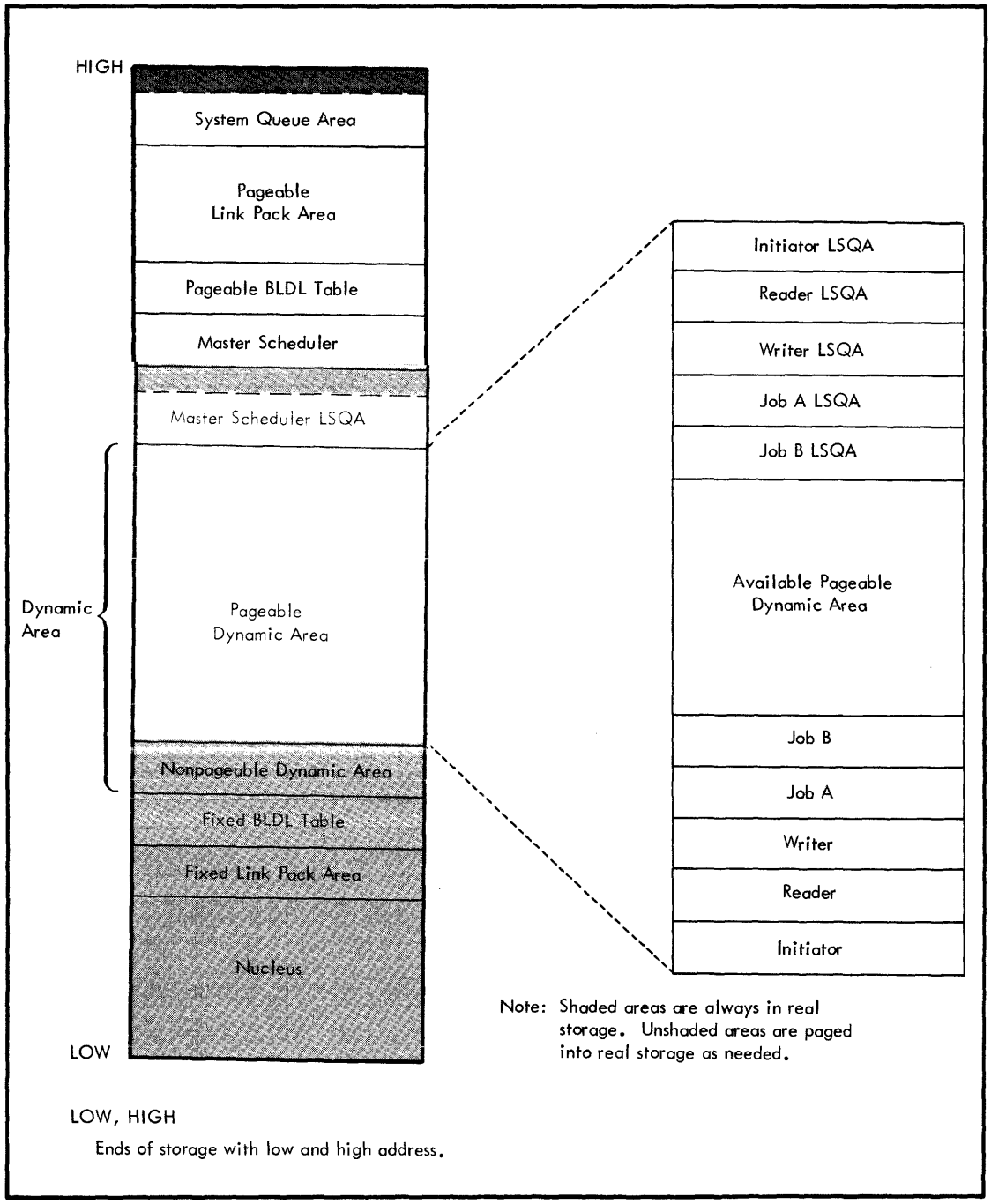


Figure 5. VS2 Virtual Storage Map

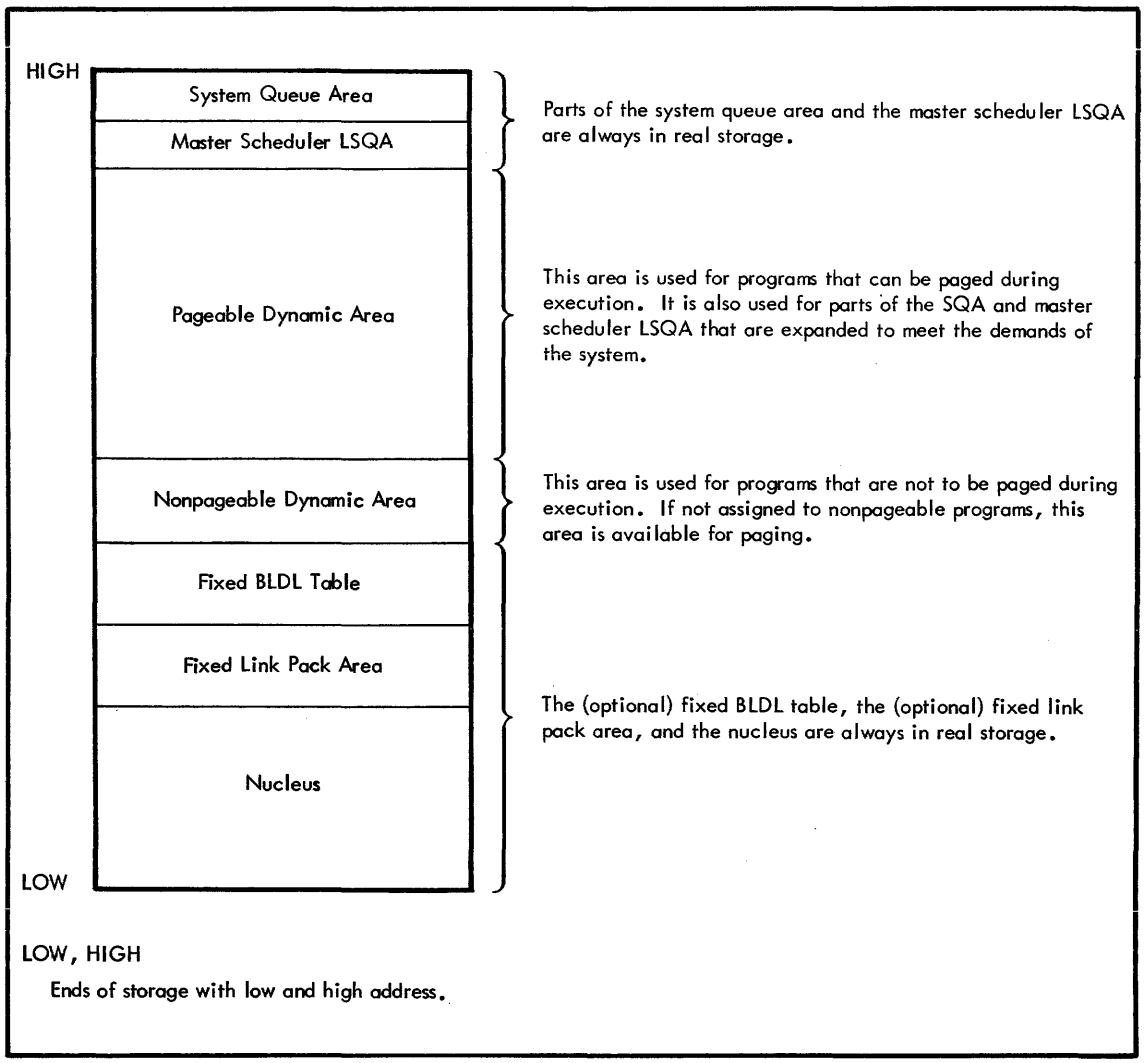


Figure 6. VS2 Real Storage Map

Storage Protection

Storage protection is a feature that prevents unauthorized or unintentional access to virtual storage by other than the intended user. Because of the larger addressable space that can be used to start more regions in a virtual storage system, storage protection is extended to allow for the additional number of regions that have to be protected. In VS2, storage protection limits both store and fetch access to storage. (Store protection prevents the contents of storage from being altered by storage addressing errors in programs or input from I/O devices. Fetch protection prevents the unauthorized retrieval of data and instructions from storage.)

Programs executing in pageable dynamic storage have read-only access to the areas of the system that are always in real storage (for example, nucleus and system queue area), and full access to only their own regions. All programs executing in pageable dynamic storage are assigned the same non-zero protection key.

Programs executing in nonpageable dynamic storage have the same access as programs executing in pageable dynamic storage. However, these programs each have unique non-zero protection keys.

System tasks have access to all allocated areas of virtual storage. These tasks have a protection key of zero.

Programs executing in pageable dynamic storage are protected from each other by the valid/invalid bits in the segment table entries; these bits control the address range the

programs can reference. Programs executing in nonpageable dynamic storage are protected from each other through their protection keys. The protection keys are also used to provide protection between programs executing in pageable and nonpageable dynamic storage.

Configuration

OS/VS2 supports the System/370 models listed below. The Model 145 is available with the initial release of OS/VS2; the local IBM branch office should be contacted for availability of the other models.

Model 145
Model 155II
Model 158
Model 165II
Model 168

A batch and TSO system will operate in 512K bytes of real storage.[1] For a detailed description of storage requirements, see *OS/VS2 Storage Estimates,* GC28-0604.

Basic Configuration

The minimum configuration required for VS2 is:

- One IBM System/370 of the type listed above, with the minimum storage as indicated above. (The clock comparator and CPU timer feature (#2001) is required on the Model 145.)
- One multiplexer channel.
- One selector or block multiplexer channel.
- Three IBM 3330 Series Disk Storage devices, or four IBM 2314 Direct Access Storage Facility or IBM 2319 Disk Storage devices.
- One card reader and punch.
- One line printer.
- One system console.

If IBM 3330 Series Disk Storage devices are used for system generation, a total of four devices is necessary. A 9-track magnetic tape is required to restore the OS/VS2 system from magnetic tape to a disk drive for system generation and maintenance.

External Page Storage

External page storage is the portion of auxiliary storage that is used to contain pages. There should be sufficient external page storage in the system for all of virtual storage except that part of virtual storage (essentially, the nucleus and other nonpageable storage) that permanently resides in real storage. If the space is inadequate, a user may have to wait for external page storage when his job is initiated.

If the time sharing option is included in the system, external page storage must also accomodate swapping requirements. Although only 16,777,216 bytes are addressable, several TSO users can share the same region and therefore external page storage in excess of 16,777,216 bytes may be needed for backup.

In VS2, a primary paging device is an auxiliary storage device that is used in preference to secondary paging devices for paging operations; portions of a primary paging device can be used for purposes other than paging operations. A secondary paging device is an auxiliary storage device that is not used for paging operations until the available space on primary

---

[1] A minimum OS/VS2 system will operate in 384K bytes of real storage, batch and reader/writer operating concurrently.

paging devices falls below a specified minimum; portions of a secondary paging device can be used for purposes other than paging operations.

Figure 7 shows the devices that may be used for external page storage, and their capacities. If both fixed-head and moveable-head devices are used, the fixed-head devices should compose the primary external page storage. External page storage may be allocated on as many as 16 devices of the types eligible.

When the primary paging device or devices become full, page migration occurs. Page migration is the transfer of pages from the primary paging device to the secondary paging device to make more space available on the primary device. The pages transferred are assoicated with the region in the pageable dynamic area with the lowest priority.

Device	Maximum number of pages	Maximum capacity in megabytes
IBM 2305 Model 1, Fixed Head Storage	1,146	4.7
IBM 2305 Model 2, Fixed Head Storage	2,483	10.2
IBM 2314 Direct Access Storage Facility	6,392	26.2
IBM 2319 Disk Storage	6,392	26.2
IBM 3330 Series Disk Storage	22,968	94.1

Figure 7. External Page Storage Devices

Input/Output Devices

The following input/output devices will be supported in VS2. (Any changes to the following list of supported devices are available from the local IBM branch office.)

Unless otherwise noted in this list, terminals are supported in both TCAM and BTAM. Availability of TCAM support for those terminals indicated by asterisk (*) may be obtained from the local IBM branch office.

Note: Terminals which are equivalent to those explicitly supported may also function satisfactorily. The customer is responsible for establishing equivalency. IBM assumes no responsibility for the impact that change to the IBM-supplied products or programs may have on such terminals.

Direct Access Storage Devices

IBM 2305 Fixed Head Storage Model 1 (Models 165II and 168 only)
IBM 2305 Fixed Head Storage Model 2
IBM 2314 Direct Access Storage Facility
IBM 2319 Disk Storage
IBM 3330 Series Disk Storage

Note: All of the above devices are supported as system residence, input/output, paging, and spooling devices.

Direct Access Storage Control Units

IBM 2835 Storage Control Model 1 (Models 165II and 168 only)
IBM 2835 Storage Control Model 2
IBM 2844 Auxiliary Storage Control
IBM 3345 Storage and Control Frame Models 3, 4, and 5 (Model 145 only)
IBM 3830 Storage Control Models 1 and 2 (including 2 channel switch additional #8171)
IBM Integrated File Adapter (IFA) Feature #4650 (Model 145 only)
IBM Integrated Storage Controls (ISC) Feature #4650 (Models 158 and 168 only)

Magnetic Tape Devices

IBM 2401 Magnetic Tape Unit
IBM 2420 Magnetic Tape Unit

IBM 2495 Tape Cartridge Reader
IBM 3410 Magnetic Tape Unit (Models 145, 155II, and 158 only)
IBM 3411 Magnetic Tape Unit and Control (Models 145, 155II, and 158 only)
IBM 3420 Magnetic Tape Unit

Control Units

IBM 2803 Tape Control
IBM 2804 Tape Control
IBM 3803 Tape Control

Tape Switch

IBM 2816 Switching Unit

Paper Tape Devices

IBM 2671 Paper Tape Reader

Printers

IBM 1403 Printer Models 2,7, and N1
IBM 1443 Printer Model N1
IBM 3211 Printer

Printer Control Units

IBM 2821 Control Unit Model 2 and 3
IBM 3811 Printer Control Unit

Card Readers and Punches

IBM 2501 Card Reader Models B1 and B2
IBM 2520 Card Read Punch
IBM 2540 Card Read Punch
IBM 3505 Card Reader
IBM 3525 Card Punch

Reader and Punch Control Units

IBM 2821 Control Unit Models 1, 5, and 6

Optical Character Recognition (OCR)/ Magnetic Ink Character Recognition (MICR) Devices

IBM 1287 Optical Reader
IBM 1288 Optical Page Reader
IBM 1419 Magnetic Character Reader (Dual Address Adapter #7730 and Expanded Capability #3800 features required)

Consoles

IBM 2150 Console/IBM 1052 Printer-Keyboard Model 7
IBM 2250 Display Unit Models 1 and 3
IBM 2260 Display Station Model 1
IBM 2740 Communication Terminal Model 1
IBM 3066 System Console (Models 165II and 168 only)
IBM Model 158 Display Console
IBM 3210 Console Printer-Keyboard
IBM 3215 Console Printer-Keyboard
IBM 3270 Information Display System
IBM 3213 Printer (Model 158 only)

Start/Stop Terminals

IBM 1030 Data Collection System
IBM 1050 Data Communication System
IBM 1060 Data Communication System
IBM 2260 Display Station Models 1 and 2
IBM 2265 Display Station

IBM 2740 Communication Terminal Models 1 and 2
IBM 2741 Communication Terminal
IBM 2760 Optical Image Unit
IBM System/7 (as an IBM 2740 Communication Terminal Model 1 with checking)
AT & T Model 83B3 Selective Calling Stations
Teletype[1] Models 33 and 35 (Paper tape is not supported with Teletype.[1])
Western Union Plan 115A Outstations
IBM World Trade Telegraph Terminals

Binary Synchronous Terminals

IBM 1130 Computing System Processor Station
IBM 1800 Data Acquisition and Control System Processor Station (BTAM support only)
IBM 2770 Data Communication System
IBM 2780 Data Transmission Terminal
*IBM 2790 Data Communication System (BTAM support only)/2715 Transmission Control Unit Model 2
IBM 2972 General Banking System Models 8 and 11 (BTAM suport only)
*IBM 3270 Information Display System
*IBM 3670 Brokerage Communication System (TCAM support only)
IBM 3735 Programmable Buffered Terminal
IBM System/3 Processor Station
IBM System/360 Processor Station (including Model 25 Integrated Communications Adapter)
IBM System/360 Model 20 Processor Station
IBM System/370 Processor Station (including Model 135 Integrated Communications Adapter)

Locally - Attached Terminals

IBM 2250 Display Unit Models 1 and 3 (GAM and GSP support only)
IBM 2260 Display Station Models 1 and 2 (GAM, GSP, and TCAM support only)
*IBM 3270 Information Display System

Telecommunication Control Units

IBM 2701 Data Adapter Unit
IBM 2702 Transmission Control
IBM 2703 Transmission Control
*IBM 2715 Transmission Control Unit Model 1
*IBM 3705 Communications Controller (Emulation program support only) (TCAM support only)
IBM 7770 Audio Response Unit Model 3 (IBM 2721 Portable Audio Terminal and IBM 2730 Transaction Validation Terminal Model 1 support only) (TCAM support only)

Devices Supported by World Trade Only

IBM 1275 Optical Reader Sorter (Dual Address Adapter and Expanded Capability feature required)
IBM 1419 Magnetic Character Reader Models 31 and 32 (Dual Address Adapter and Expanded Capability feature required)

---

[1]Trademark of Teletype Corporation, Skokie, Illinois.

System Control Program

The VS2 system control program consists of the following major components:

- Job management, which schedules all work done by the computing system.
- Task management, which allocates system resources and controls system execution.
- Input/output supervision, which performs all I/O operations.
- Data management, which coordinates data flow.
- Recovery management, which attempts to recover from system malfunctions.

Job Management

Job management is a term that is generally used to describe the functions of the master scheduler and the job scheduler. The master scheduler and the job scheduler are major parts of the operating system that control the processing of jobs. The master scheduler initializes the system and responds to operator commands by initiating the requested actions. The job scheduler reads and interprets job definitions, schedules the jobs for processing, initiates and terminates the processing of jobs and job steps, and records job output data.

Figure 8 provides a summary of the VS2 changes to the job management functions.

Facility	Change
Master scheduler	Changes to operator commands to reflect virtual storage and nonsupport of some MVT parameters.
Job scheduler	ASB reader, DSO writer, and column binary not supported; new I/O load balancing facility; new maximum number (63) of initiators that can be started; ability of operator to cancel a job waiting for a region or a data set.
Multiple console support (MCS)	Now standard (previously optional).
System log	Now standard (previously optional).
Hardcopy log	Now standard (previously optional).
Job step timing	Now standard (previously optional).
Checkpoint/restart	Now standard (previously optional); only 2K block size supported.
System management facilities (SMF)	Now standard (previously optional); tape and OUTLIM facility not supported; new exit for system output; new accounting information to record virtual storage usage.
Automatic volume recognition (AVR)	None. (See the chapter "Options".)
Time slicing	None. (See the chapter "Options".)
Device independent display operator console support (DIDOCS)/status display support (SDS)	None. (See the chapter "Options".)
Track stacking	None. (See the chapter "Options".)

Figure 8. VS2 Changes to Job Management

Master Scheduler

The master scheduler is one of the system tasks established when the system is loaded. Its functions can be divided into two categories: initialization and command processing.

Initialization

Master scheduler initialization consists of the following functions:

- Initializing the communications task to handle all communication with the operator console.
- Scheduling execution of the initial SET command.

- Initializing the time-of-day clock.
- Initializing the job queue (SYS1.SYSJOBQE) data set or performing restart processing.
- Setting volume attributes of all volumes listed in PRESRES.
- Scheduling execution of the automatic START commands.
- Scheduling execution of the SEND command.
- Initializing the system log.
- Initializing the system management facilities (SMF).
- Initializing the missing interruption checker task.

Major Changes from MVT: In VS2, initializing of the I/O load balancing function is performed.

Command Processing

Command processing is the reading, scheduling, and executing of operator commands issued via either a console device or an input job stream.

The reading of commands entered via a console device is performed by routines operating under a console communication task; the reading of commands entered via an input job stream is performed by routines operating under a reader task associated with that input job stream.

The scheduling of a command is the storing of the command and the readying of a task to continue processing the command. A command scheduling routine operates under either the console communications task (when the command was issued via a console device), or the reader task (when the command was issued via an input job stream).

The executing of a command is the performance of the function specified in the command. The functions are performed either as new tasks established by the master scheduler or as parts of existing system tasks. Commands are classified, accordingly, as task-creating commands and existing-task commands. In VS2, the task-creating commands START and MOUNT are attached and run in a separate region.

Major Changes from MVT: In VS2, the size of the master scheduler region can be specified at IPL time. Also, some operator commands have been changed; these changes are discussed in the chapter "Compatibility".

Job Scheduler

The job scheduler is divided into three major parts: the reader/interpreter, the initiator/terminator, and the output writer. Each part is a separate task and can thus be executed concurrently with and independently of the others.

Reader/Interpreter

The reader/interpreter reads job and step definitions from an input job stream, analyzes the definitions, and builds control blocks and tables that are used during initation and execution of the job steps. The reader/interpreter also reads and analyzes commands encountered in the input stream. When the reader/interpreter encounters data in the input stream, it writes the data on a direct access device.

The control blocks and tables constructed by the reader/interpreter contain the following information:

- Job attributes.
- Job step attributes.
- Information needed to assign devices to data sets.
- Data set attributes.

Major Changes from MVT: In VS2, the automatic SYSIN batching (ASB) reader is not supported. Therefore, column binary is also not supported.

Initiator/Terminator

The initiator/terminator selects jobs and job steps to be executed. It analyzes the I/O device requirements of the job steps, allocates devices to them, creates tasks for them, and at completion of the jobs, supplies control information for writing job output on a system output unit.

After selecting an interpreted job to be executed, the initiator/terminator examines the types of regions requested for the job. If a job or job step requires nonpageable dynamic storage for execution, the initator/terminator reserves a unique non-zero protection key for the job. (See the discussion of storage protection in the chapter "Introduction".)

During allocation, in order to reduce contention for I/O devices, a new algorithm for I/O load balancing is used. The new algorithm allocates devices for data sets that have nonspecific device requests. Rather than basing the algorithm on a count of allocated data sets on a device (as in MVT), in VS2 the actual number of I/O requests to a tape or direct access device will be monitored to get a more accurate picture of I/O load. The device determined to be the best candidate for allocation to a given data set is then selected.

The following factors can contribute to the efficient allocation of I/O devices and thus help the user realize benefit from I/O load balancing:

- Devices not dedicated to specific applications should be distributed evenly across channels; within channels, they should be distributed evenly across control units.
- DD statements entered at the time of job initiation should be sequenced in the order of expected activity.

Major Changes from MVT: In MVT, a maximum of 15 initiators could be started; in VS2, a maximum of 63 initiators can be started. In VS2, the operator has the ability to cancel a job waiting for a region or a data set. VS2 provides a more accurate algorithm for I/O load balancing.

Output Writer

The output writer transfers system messages and system output data sets from the direct access volume on which they were initially written by the system to a specified output device.

Output data sets can be directed to a class of devices, and references to the data are then placed on an output work queue. Because the queue is maintained in priority sequence, the system output writers can select jobs in the output work queue on a priority basis.

Major Changes from MVT: In VS2, the direct system output (DSO) writer is not supported.

Facilities

The major facilities provided by job management are:

- Multiple console support.
- System log.
- Hardcopy log.
- Checkpoint/restart.
- System management facilities.
- Job step timing.

Multiple Console Support (MCS)

Multiple console support (MCS) allows one operating system to use many operator consoles. Each console in a multiple console configuration is defined by specifying:

- The operator commands the system will accept from that console.
- A console to act as an alternate if a failure occurs.
- The types of messages the console will receive.

In a system with MCS, one console acts as the master console and the rest (up to thirty-one) are secondary consoles. The master console is the basic console required for operator-system communication; it alone can accept all possible operator commands, change the status of the hardcopy log and the messages to be recorded on it, switch to a different master console, and receive all messages not specifically assigned to any other console. A secondary console is any console other than the master console; it handles one or more functions assigned to it (for example, it might handle tape activity).

Major Changes from MVT: In VS2, MCS is a standard facility; in MVT, MCS was optional.

System Log

The system log consists of data sets on which the communication between problem programs, operators, and the system is recorded. It may contain the following kinds of information:

- Job time, job step time, and data from the JOB and EXEC statements of a job that has ended.
- Operating data entered by problem programs using a write-to-log (WTL) macro instruction.
- Descriptions of unusual events that occurred during a shift.
- Write-to-operator (WTO) and write-to-operator with reply (WTOR) messages.
- Accepted replies to WTOR messages.
- Commands issued through operator's consoles and the input stream, and commands issued by the operating system.

Major Changes from MVT: In VS2, the system log is a standard facility; in MVT, it was optional.

Hardcopy Log

The hardcopy log is a permanent record of system activity that is mandatory for systems with an active graphic console or multiple active consoles; for other systems, the primary console device serves as the hardcopy log.

Since multiple console support allows more than one console in a system, an installation might find it helpful to record all the messages issued by and to a system. The hardcopy log is a place to collect these messages, and therefore an installation can review system activity by reviewing message activity.

Major Changes from MVT: In VS2, the hardcopy log is a standard facility; in MVT, it was optional.

Checkpoint/Restart

If a job step is terminated before successful completion, checkpoint/restart can make it possible to resume execution from the beginning of the step or from a place within the step. Either way, the restart can be made to occur after resubmission of the job by the programmer or it can be made to occur automatically when the failure occurs.

The CHKPT macro instruction is coded in the user's program at a checkpoint to be taken. A checkpoint is the point at which information about the status of a job can be recorded so that the job step can be later restarted.

Checkpoint/restart includes a checkpoint routine and several restart routines.

The checkpoint routine gathers and records on a checkpoint data set enough information about the status of the job step and its related control blocks to allow a restart from the place where the checkpoint is taken.

The restart routines can be invoked when a job step is resubmitted for restart, or they can be invoked automatically when a failure occurs. The functions performed by restart routines depend upon the type of restart that is requested.

If the restart is to be made from the beginning of a job step, the RESTART parameter of the JOB statement must contain the name of the step to be restarted, and routines of the ready task simply bypass preceding steps and begin processing with the named step.

If a step is to be restarted from the beginning, automatically, then restart processing begins during step termination. The step termination routine of job management invokes routines to verify that a restart can be performed and requests the operator to authorize the restart.

If a step is to be restarted from a place where a checkpoint was taken and the job is resubmitted, the RESTART parameter of the JOB statement must identify the step and checkpoint identifier, while a SYSCHK DD statement must describe the checkpoint data set.

If a step is to be restarted automatically from a place where a checkpoint was taken, the step termination routine invokes routines to ensure that all data sets for the step are kept.

For a detailed description of checkpoint/restart, see *OS/VS Checkpoint Restart,* GC26-3784.

Major Changes from MVT: In VS2, checkpoint/restart is a standard facility; in MVT, it was optional. Also, in VS2, only 2K block size data sets are supported for checkpoint/restart data sets.

System Management Facilities (SMF)

System management facilities (SMF) collect and record system information. The information obtained can be used in management information reports that describe system efficiency, performance, and usage. The SMF records contain such data as:

- System configuration.
- Job and job step identification.
- CPU wait time.
- CPU and input/output device usage.
- Temporary and non-temporary data set usage and status.
- Virtual and real storage usage.
- Status of removable direct access volumes.
- Allocation recovery records.
- Paging statistics.

SMF provides exits to installation-supplied routines that can monitor the operation of a job or job step and generate the installation's own SMF records. The exit routines can cancel jobs, write records to the SMF data set, open and close user-defined data sets, suppress the writing of certain SMF records, and enforce installation standards (such as identification of users). Dummy routines are automatically provided for all unused exits.

For a detailed description of SMF, see *OS/VS System Management Facilities,* GC35-0004.

Major Changes from MVT: SMF has been changed in the following ways:

- SMF is a standard facility of VS2.
- SMF records in VS2 contain additional accounting information to record new system environmental characteristics.

- SMF in VS2 provides one new exit from the system control program that receives control each time an SMF logical record has been formatted and is ready to be written out. The exit may prevent the record from being written.
- In VS2, SMF does not support tape for recording SMF data.
- In VS2, the OUTLIM facility is not supported.

Job Step Timing

Each job step can be timed and the time limits enforced. The amount of time used is recorded after a job step is finished. In addition, the following are included in this facility: the ability to request the date plus the time of day, to change the time at midnight, and to request, check, and cancel intervals of time.

Major Changes from MVT: In VS2, the job step timing facility is standard; in MVT, it was optional.

Task Management

Task management is a major function of the operating system that coordinates the use of resources and maintains the flow of central processing unit (CPU) operations. It assigns resources to perform tasks, keeps track of all such assignments, and ensures that the resources are freed upon task completion. VS2 changes to task management functions are summarized in Figure 9.

Facility	Change
Interruption supervision	New timer facilities (CPU timer and clock comparator); new program interruptions (translation exceptions, program event recording, set system mask, and monitor call); new extended control (EC) mode architecture.
Paging supervision	New support for virtual storage; support for TSO swapping function.
Task supervision	New dispatching facility (automatic priority group).
Contents supervision	Scatter load, TESTRAN, storage hierarchies, and transient areas not supported; PCI fetch now standard (previously optional).
Virtual storage supervision	Support for virtual storage (similar to MVT main storage supervision).
Timer supervision	New timer facilities (CPU timer and clock comparator); location 80 timer not supported.

Figure 9. VS2 Changes to Task Management

Interruption Supervision

All supervisory activity begins with an interruption (a break in the normal sequence of instruction execution). An interruption can occur either because a program has requested a control program service or because an event has occurred which requires supervisory processing. There are five types of interruptions:

- Supervisor call (SVC) interruptions occur when a supervisor call (SVC) instruction is executed.
- Input/output interruptions occur when an I/O device is readied or an I/O operation terminates, or during an I/O operation such as a program controlled interruption (PCI).
- Timer/external interruptions occur when a specified time interval expires or when the interruption key of the system control panel is pressed.

- Program interruptions occur when a program attempts an invalid action, when a data error is detected, or where a page fault occurs.
- Machine check interruptions occur when the CPU detects hardware malfunctions.

Any interruption causes the current program status word (PSW) to be replaced by a new PSW. The new PSW causes an appropriate interruption handler to receive control, depending on the type of interruption. The interruption handler saves critical information (such as register contents and PSW information) necessary to return control to the interrupted program after the interruption is processed. In most cases, the interruption handler analyzes the interruption and passes control to a special purpose routine for processing the interruption.

Major Changes from MVT: In VS2, the interruption handlers will support the new timer facilities and react to the new program interruptions (translation exceptions, program event recording, set system mask, and monitor call). All of the interruption handlers have been modified to support the new extended control (EC) mode architecture; for a discussion of EC mode, see "Problem Programs" in the chapter "Compatibility".

Paging Supervision

The paging supervisor allocates and releases real storage space for pages, and transfers pages between real storage and external page storage. Whenever a virtual address in a page is referred to and that page is not in real storage, an interruption occurs and control is given to the paging supervisor.

First, page reclamation is attempted in order to satisfy the request if the page is already in storage. In this case, the page may be available (previously released but not yet paged out), in page-out processing (being transferred from real storage to external page storage), or in page-in processing (being transferred from external page storage to real storage). If the required page is found, appropriate action is taken to use the page. Otherwise, to move pages into real storage, the paging supervisor maintains a list of page frames that are not receiving a high reference rate by the programs that are currently using them.

When the number of available page frames falls below a predetermined value specified at NIP time, the paging routines will select the page frames of the least-used pages and make them available for use. The selection process is primarily based on the change and reference bit settings in the storage key associated with each page frame.

If a page in a selected frame has been changed, the changed page is moved to external page storage before the page frame is made available. If a page in a selected frame has not been changed, it is not moved to external page storage if a copy already exists in external page storage.

The selecting of page frames continues until a predetermined maximum number of available frames is reached. During this process, page frames that contain unreferenced pages are selected before page frames that contain referenced pages. Also, page frames that contain unchanged pages are selected before page frames that contain changed pages.

The paging process utilizes an efficient slot sorting technique to optimize paging operations by minimizing page data set seek or rotational delay time, or both. A slot is a continuous area on a paging device in which a page can be stored. The capability to access the next slot sequentially forms the basis of the slot sorting algorithm.

In summary, the paging supervisor:

- Recognizes when a page must be transferred to real storage.
- Selects a page frame in which to place the page.
- Maintains a supply of page frames.
- Saves pages that have been changed in real storage.
- Recognizes when a page must be made non-transferable to external page storage.

Major Changes from MVT: The paging supervisor is necessary to support virtual storage, and was not needed in MVT. When the time sharing option (TSO) is included in the system, the paging supervisor performs the swapping function performed by the TSO supervisor in MVT.

Task Supervision

The task supervisor performs services requested by tasks and allocates CPU time among competing tasks. A task is described to the operating system by a task control block (TCB) and many services provided by task supervision relate to the TCB. The services supplied are listed below. Where applicable, the macro instructions used to request the services are shown in parentheses.

- Attaching/detaching a subtask (ATTACH/DETACH).
- Changing the dispatching priority of a task (CHAP).
- Extracting information from a TCB (EXTRACT).
- Specifying a user program interruption exit routine (SPIE).
- Synchronizing a program with one or more events (WAIT,POST).
- Serializing the use of one or more resources (ENQ, DEQ, RESERVE).
- Scheduling asynchronous exit routines.
- Testing the authorization of a user to perform a given function (TESTAUTH).
- Modifying information contained in the program status word (MODESET).
- Returning control by the dispatcher.

Major Changes from MVT: In VS2, a single task priority level may be identified at system generation or system initialization time for the dynamic dispatcher. Tasks within this priority level will be dispatched in a way that makes better use of the system's CPU and I/O resources. (The priority level cannot be identified also as a time-slicing group).

Contents Supervision

The contents supervisor locates requested programs, fetches the programs to virtual storage if necessary, and schedules their execution. The major contents supervision functions are requested by the LINK, LOAD, DELETE, XCTL, and ATTACH macro instructions. The functions are:

- Maintaining directories which describe all modules located in various portions of virtual storage.
- Searching directories for requested modules.
- Testing the status of modules to determine if they are available for use.
- Deferring requests for unavailable modules and restarting the deferred requests when the modules become available.
- Requesting the use of modules that are not in virtual storage.
- Scheduling execution of modules.

Program controlled interruption (PCI) permits the program to cause an I/O interruption during execution of an I/O operation. PCI provides an means of alerting the program of the progress of chaining during an I/O operation. It also permits programmed dynamic storage allocation.

PCI fetch is able to bring a program into storage with only one seek of the disk if:

- A buffer is always available for relocation dictionaries.
- No errors occur during the I/O operation.
- No cylinders are crossed while bringing in the program.

- The speed of the central processing unit allows PCI to modify the channel command word before it reaches the channel.

An additional WAIT and seek are required each time a buffer is not available. A seek is required each time an error occurs or a cylinder is crossed. If the speed of the central processing unit does not allow PCI to perform its function in time, the number of seeks needed by the standard fetch are required.

Major Changes from MVT: VS2 does not support scatter loading, TESTRAN attributes, or hierarchy support; these functions will be ignored if requested. Also, in VS2, transient areas are no longer needed because the pageable link pack area contains all those modules that formerly executed out of transient areas. PCI fetch is standard in VS2; in MVT, it was optional.

Virtual Storage Supervision

The virtual storage supervisor allocates address space within virtual storage. The supervisor routines service two macro instructions: GETMAIN (used to allocate virtual space) and FREEMAIN (used to free allocated virtual space):

- The GETMAIN routines service requests for virtual storage space, including requests for a region, space within a region, space within a local system queue area, and space within the system queue area.
- The FREEMAIN routines service all requests for making space available for reallocation, including requests for an entire region, space within a region, space within a local system queue area, and space within the system queue area.

Various supervisor services use areas in the system queue area and local system queue areas called quickcells. During execution of GETMAIN requests, these areas reduce the time required to allocate space for control blocks, and thus service the requests more quickly.

Major Changes from MVT: The virtual storage supervisor in VS2 perform the same function as the main storage supervisor in MVT. Basically, only internal changes have occurred as a result of the use of virtual storage. However, GETMAIN has been extended to service requests for virtual storage beginning on a page boundary. Also, subpools in VS2 are allocated in 4K-byte blocks.

Timer Supervision

The timer supervisor uses three timers, each with one microsecond precision, to service requests of programmers:

- The time-of-day clock is used to calculate the time of day and maintain the current date. This clock can only be set at IPL time.
- The clock comparator is used to measure elapsed time and schedule activity for specific times of the day. A value is placed in the clock comparator and when the time-of-day clock reaches that value, an external interruption occurs.
- The CPU timer is used to time tasks. A value is placed in the timer and when the timer is decremented to zero, an external interruption occurs.

Timer supervision is performed by four major routines:

- The TIME routine supplies the current date and time of day.
- The STIMER routine processes requests for interval timing based on task execution or real time.
- The TTIMER routine supplies the time remaining in a previously requested interval or it cancels previous timing requests.
- The timer second-level interruption handler processes timer interruptions.

Major Changes from MVT: In VS2, timer supervision routines support the CPU timer and the clock comparator, which are new features. The location 80 timer of MVT is not supported since the time-of-day clock is used instead.

Input/Output Supervision

The input/output (I/O) supervisor starts, terminates, and (where necessary) restarts activity on input/output devices. Its overall objective is to ensure that requested I/O operations are performed. The VS2 changes to I/O supervision are summarized in Figure 10.

Facility	Change
Starting I/O	Translation of virtual channel programs to real channel programs; page fixing.
Terminating I/O	None.
Restarting I/O	Translation of virtual channel programs to real channel programs; page fixing.

Figure 10. VS2 Changes to Input/Output Supervisor

Starting I/O Operations

Two parts of the control program are normally involved in starting I/O operations: access method routines and the I/O supervisor.

Each access method routine prepares information required by the I/O supervisor to start I/O operations. It gathers information used to initiate the I/O operations and places the information in control blocks. It then issues an EXCP macro instruction, causing entry to the I/O supervisor.

The I/O supervisor determines if the I/O device associated with the operation is not busy, and if so, whether any channel associated with the device is not busy. When both the device and an associated channel are not busy, the I/O supervisor prepares for the execution of channel programs and issues a START I/O instruction to initiate the operation.

The I/O supervisor must distinguish between two classes of channel programs:

- Channel programs that will run without address translation. These channel programs are submitted for execution from programs executing in nonpageable dynamic storage.
- Channel programs that will not run without address translation. These channel programs are submitted for execution from programs executing in pageable dynamic storage. The I/O supervisor provides the necessary translation by building another copy of the channel program. This copy contains real storage addresses instead of virtual storage addresses, and takes into account discontiguous pages frames in real storage. In addition, the I/O supervisor fixes all required pages for the duration of the I/O operation. (Translation of virtual storage channel programs to real storage channel programs is required because channels do not use the segment and page tables for translation.)

Major Changes from MVT: The I/O supervisor in VS2 translates virtual storage channel programs to real storage channel programs; this feature was not needed in MVT. Also, the I/O supervisor performs page fixing.

Terminating I/O Operations

I/O operations terminate either normally because the operation is completed, or abnormally because an error is detected. When an I/O operation terminates, an I/O interruption occurs, causing CPU control to be passed first to the I/O interruption handler and then to the I/O interruption supervisor portion of the I/O supervisor to process the interruption.

The I/O supervisor posts the completion of the I/O operation, schedules error routines when the operation terminated abnormally, and, if possible, starts another I/O operation on the channel. Then the I/O interruption supervisor returns control to the interruption handler.

Major Changes from MVT: None.

Restarting I/O Operations

When an error occurs during I/O activity, the I/O supervisor invokes an I/O error routine which records the error on the SYS1.LOGREC data set and begins a cycle of restarts. The cycle continues either until the error is corrected or until it is declared to be permanent (uncorrectable).

An error is considered to be corrected by an I/O error routine when no errors occur during a retry of the I/O activity. When an error is corrected, the I/O supervisor continues processing normally as if no error had been found.

I/O error routines count the number of retries and indicate that the error is permanent when the maximum allowable number of retries has been reached. When a permanent error is found none of the related requests for the involved data set are started.

Major Changes from MVT: The I/O supervisor in VS2 translates virtual storage channel programs to real storage channel programs; this feature was not needed in MVT. Also, the I/O supervisor performs page fixing.

Data Management

Data management is a major function of the operating system that involves organizing, cataloging, storing, retrieving, and maintaining data. The data management routines are primarily responsible for moving information between virtual storage and external storage. Figure 11 provides a summary of the VS2 changes to the data management facilities.

Facility	Change
Access methods	Standard support of QSAM and BDAM (previously optional); optional support of new VSAM (see the chapter "Options"); QTAM not supported; support of chained scheduling only in nonpageable storage (BSAM and QSAM).
Catalog management	None.
Direct access device space management (DADSM)	None.
I/O support	DEB validity checking performed.
Shared direct access storage devices (shared DASD)	None. (See the chapter "Options".)
Direct access volume serial number verification	Now standard (previously optional).

Figure 11. VS2 Changes to Data Management

Standard Access Methods

Access methods are techniques for moving data between virtual storage and input/output devices. In VS2, there are four standard access methods, each of which pairs a data set organization with a retrieval technique. The choice of which access method to use depends upon which best suits a particular application or installation.

VS2 data sets can be organized in four ways:

- Sequential: Records are arranged in physical sequence, and are usually read or updated in the same order in which they appear. This organization is used for all magnetic tapes, but may also be selected for direct access devices. Punched tape, punched cards, and printed output are considered to be sequentially organized.

  In the sequential organization, individual records cannot be located quickly. Also, records usually cannot be deleted or added easily unless the entire data set is rewritten. This organization is generally used when most records are processed each time the data set is used.

- Indexed Sequential: Records are arranged in collating sequence on the tracks of a direct access volume according to a key that is part of every record. The location of each record is computed through the use of indexes maintained by the system. This organization permits direct as well as sequential access to any record.

  In this organization, since the system has control over the location of the individual records, the user needs to do very little input/output programming. Also, since a separate area of the data set is set aside for added records, the data set need not be rewritten to accommodate new records.

  In VS2, access methods using indexed sequential data sets are optional.

- Direct: Records are arranged in random order on a direct access volume. Each record is stored or retrieved directly with addressing as specified by the user. This organization is generally used for data sets whose characteristics do not permit the use of sequential or indexed sequential organization, or for data sets where the time required to locate individual records must be kept to an absolute minimum.

  In this organization, the user is largely responsible for the programming required to locate the records.

- Partitioned: This organization has characteristics of both the sequential and the indexed sequential organizations. Independent groups of sequentially organized data, called members, are in direct access storage. Each member has a unique name stored in a directory that is part of the data set and contains the location of the member's starting point.

Partitioned organization is used mainly to store programs, subroutines, and tables. As a result, partitioned data sets are often referred to as libraries.

VS2 data access techniques are divided into two categories:

- Queued: This technique provides GET and PUT macro instructions to handle individual records. It offers a maximum amount of automatic input/output facilities. It may be used only to retrieve records in a sequential order (for example, records on magnetic tape).

  The GET and PUT macro instructions cause automatic blocking and deblocking of the records stored and retrieved. Look-ahead buffering and synchronization of input and output operations with CPU processing are automatic features of this technique.

- Basic: This technique provides READ and WRITE macro instructions to handle blocks (not records). It places some of the responsibility for data handling on the programmer but gives him more control of input/output operations. It is used when the operating system cannot predict the sequence in which the records are to be processed, or when some or all of the automatic functions performed by the queued access technique are not desired.

  Since READ and WRITE macro instructions process blocks, the blocking and deblocking of records is the responsibility of the programmer. Although look-ahead buffering and synchronized scheduling are not automatically included in this technique, macro instructions are provided to perform these functions.

Following are brief descriptions of the four access methods that are standard in VS2; Figure 12 summarizes their characteristics. Optional access methods that are available in VS2 (telecommunications access method, virtual storage access method, basic telecommunication access method, basic indexed sequential access method, queued indexed sequential access method, and graphics access method) are described in the chapter "Options".

Organization / Access Method / Characteristics	Sequential		Partitioned	Direct
	QSAM	BSAM	BPAM	BDAM
Primary macro instructions	GET PUT PUTX	READ WRITE	READ WRITE FIND STOW	READ WRITE
Synchronization of program with I/O	Automatic	CHECK*	CHECK*	WAIT*
Record formats transmitted	Logical - F,V Block - V	Block - F,V,U	Block (part of member) - F,V,U	Block - F,V,U
Buffer creation and construction	BUILD*, GETPOOL*, Automatic	BUILD*, GETPOOL*, Automatic	BUILD*, GETPOOL*, Automatic	BUILD*, GETPOOL*, Automatic
Buffer techniques	Automatic, Simple, Exchange	GETBUF*, FREEBUF*	GETBUF*, FREEBUF*	GETBUF* FREEDBUF*, Dynamic FREEDBUF*
Transmittal modes (work area/buffer)	Move, Data, Locate, Substitute			

Note: * denotes a macro instruction.

Figure 12. Summary of Standard VS2 Access Method Characteristics

Basic Sequential Access Method (BSAM)

BSAM can be used for storing or retrieving data blocks arranged sequentially on sequential access or direct access devices. (See descriptions of basic and sequential above.)

Major Changes from MVT: In VS2 BSAM, chained scheduling is only available to programs excuting in nonpageable dynamic storage. Requests for chained scheduling from programs executing in pageable regions will be ignored, and normal scheduling will be substituted.

Queued Sequential Access Method (QSAM)

QSAM is an extended version of BSAM where a queue is formed of input data waiting processing or output data awaiting transfer to auxiliary storage or an output device. (See description of queued and sequential above.)

Major Changes from MVT: In VS2 QSAM, chained scheduling is only available to programs executing in nonpageable dynamic storage. Requests for chained scheduling from programs executing in pageable regions will be ignored, and normal scheduling will be substituted.

Basic Direct Access Method (BDAM)

BDAM is used to directly retrieve or update particular blocks of a data set on a direct access device. (See descriptions of basic and direct above.)

Major Changes from MVT: None.

Basic Partitioned Access Method (BPAM)

BPAM can be used to create program libraries in direct access storage for convenient storage and retrieval of programs. (See descriptions of basic and partitioned above.)

Major Changes from MVT: None.

Catalog Management

Catalog management routines maintain the collection of data set indexes (the catalog) that is used by the control program to locate volumes. The catalog management routines also locate the cataloged data sets.

The catalog, itself a data set (SYSCTLG), resides on one or more direct access volumes. It contains indexes that relate data set names to the serial numbers and device types of the volumes containing the data sets.

In maintaining the catalog, catalog management routines create and delete indexes, and add or remove entries. To locate a data set, catalog management routines search through the indexes for the index entry containing the last part of the qualified name of the data set.

The catalog management routines are used primarily by the job scheduler and the IEHPROGM utility program, although they can be used by any processing program:

- The scheduler invokes the catalog management routines during the initiation and termination of a job step. During initiation, a catalog management routine locates cataloged data sets. During termination, a catalog management routine may catalog or uncatalog data sets referred to during the job step and specified for the catalog.
- The IEHPROGM utility program invokes catalog management routines to create and delete indexes, and to add or remove entries. The IEHPROGM program does not locate data sets.
- Processing programs can invoke the catalog management routines via the CATALOG, INDEX, and LOCATE assembler language macro instructions. These macro instructions provide access to all the catalog management routines.

Major Changes from MVT: None.

Direct Access Device Space Management (DADSM)

Direct access device space management (DADSM) consists of routines that allocate space on direct access volumes to data sets. The routines are used primarily by job management routines during the initiating of job steps when space is obtained for output data sets. They are also used by other data management routines for increasing the space already assigned to a data set, and for releasing space no longer needed.

The DADSM routine controls allocation of space through the volume table of contents (VTOC). The VTOC is built when a volume is initialized by the direct access storage device initialization (IBCDASDI) utility program. The VTOC indicates the current usage of the space on the volume.

When space is needed on a volume, the DADSM routines check the VTOC for enough contiguous, available tracks to satisfy the request. If there are not enough contiguous tracks, the request is filled using up to five noncontiguous groups of free tracks.

Major Changes from MVT: None.

Input/Output Support

Input/output (I/O) support routines perform three functions associated with I/O operations:

- Opening a data control block before a data set is read or written.
- Closing a data control block after a data set has been read or written.
- Processing end-of-volume (EOV) conditions when an end-of-volume or end-of-data (EOD) set condition occurs during an I/O operation.

Open Processing

Before access can be gained to a data set, the data control block (DCB) for that data set must be opened by means of an OPEN macro instruction. When a processing program issues an OPEN macro instruction, the Open routine of the control program performs:

- Volume mounting and verification.
- Merging of data set attributes from the DD statement and the data set label into the control blocks.
- Determination of access method routines.

Major Changes from MVT: Open processing in VS2 performs DEB validity checking.

Close Processing

After processing has been completed for a data set, the data control block (DCB) for that data set must be closed by means of a CLOSE macro instruction. When a processing program issues a CLOSE macro instruction, the Close routine of the control program performs:

- Input and output label processing.
- Volume disposition.
- Restoration of data control block to its original condition by removing the information that was merged from the DD statement.

Major Changes from MVT: Close processing in VS2 performs DEB validity checking.

End-of-Volume Processing

End-of-volume (EOV) processing is performed when end-of-data set or end-of-volume conditions occur during I/O operations on sequentially organized data sets. When a routine of a sequential access method encounters a tape or file mark (end-of-data set) or an end-of-volume condition, the routine issues an SVC instruction to pass control to the EOV routine.

Major Changes from MVT: None.

Direct Access Volume Serial Number Verification

Direct access volume serial number verification is standard in VS2. The volume serial number of a direct access device is checked after an unsolicited device-end interruption condition has been corrected and the volume has been put back online again.

When an unsolicited device-end interruption is received from a direct access device, the I/O supervisor ensures that the volume serial number of the mounted volume agrees with the volume serial in the unit control block (UCB).

Major Changes from MVT: In VS2, direct access volume serial number verification is a standard facility; in MVT, it was optional.

Recovery Management

Recovery management facilities gather information about hardware reliability and allow retry of operations that fail because of CPU, I/O device, or channel errors. The facilities of VS2 that record the environment of the system at the time of a machine malfunction and attempt to recover from the malfunction are the machine check handler and the channel check handler; the facility that attempts recovery from various I/O errors is dynamic device reconfiguration. Figure 13 provides a summary of the VS2 changes to the recovery management programs.

Facility	Change
Machine check handler (MCH)	Now standard (previously optional); enhanced MODE command.
Channel check handler (CCH)	Now standard (previously optional).
Dynamic device reconfiguration (DDR)	Now standard (previously optional); system residence device and page data set not supported.
Alternate path retry (APR)	None. (See the chapter "Options".)

Figure 13. VS2 Changes to Recovery Management

Machine Check Handler (MCH)

Machine check handler (MCH) is a standard facility of VS2 that attempts to reduce lost computing time due to machine malfunctions.

Recovery from machine malfunctions is initially attempted by the hardware instruction retry (HIR) and error correction codes (ECC) facilities of the machine. If the machine recovery attempts are unsuccessful, a machine check interruption will occur. MCH then analyzes the data and attempts to keep the system as fully operational as possible. It may:

- Attempt to repair malfunctions and, if possible, resume operation, leaving no adverse effects on the system.
- Attempt to terminate affected tasks and resume operation.
- Isolate the failure to a page frame in real storage, and mark the page frame as invalid or unavailable for use by the paging supervisor.
- Place the system in a wait state.

In all cases, whether or not recovery is successful, MCH constructs records of the error environment on the SYS1.LOGREC data set and issues diagnostic messages to the operator.

Major Changes from MVT: In VS2, MCH is a standard facility; in MVT, it was optional. In VS2, there is an enhanced MODE command that is used for all machine models. The MCH facility in VS2 has been expanded to allow MCH to mark a page frame unavailable.

Channel Check Handler (CCH)

Channel check handler (CCH) is a standard facility of VS2 that allows the user to recover from errors in the execution of channel programs. CCH receives control from the I/O supervisor when a channel data check, channel control check, or interface control check occurs.

For all three checks, it provides the operator with information about channel errors that enables him to keep statistics about the channel or help bring about recovery from system termination conditions.

In addition, for channel control checks and interface control checks, it provides the device-dependent error recovery procedures (ERPs) of the I/O supervisor with information needed to attempt a retry of a channel operation that has failed.

Major Changes from MVT: In VS2, CCH is a standard facility; in MVT, it was optional.

Dynamic Device Reconfiguration (DDR)

Dynamic device reconfiguration (DDR) is a standard facility of VS2 that allows the system and the user to circumvent an I/O failure, if possible, by moving a demountable volume from one device to another. The device swap is accomplished without abnormal termination of the affected job and without another initial program load (IPL).

A request to move a volume may be initiated by either the system or the operator. The system requests DDR after a permanent (uncorrectable) I/O error has occured. The operator may request DDR at any time by issuing the SWAP command. He may substitute one device for another, or simply interrupt processing on a device to carry out cleaning procedure.

Major Changes from MVT: In VS2, DDR is a standard facility; in MVT, it was optional. In VS2, DDR for the system residence device is not supported. Also, DDR does not support the page data sets.

Standard Support Programs

In addition to the job management, task management, input/output supervision, data management, and recovery management components of the system control program, VS2 contains a number of other standard support programs that are necessary to run the system. Figure 14 provides a summary of the VS2 changes to these programs.

Facility	Change
OS/VS2 assembler	New assembler with improved diagnostics and extended language capability.
Linkage editor F and loader	New control statements and parameters to order and align CSECTs and common areas, and to designate APF codes.
Utilities	IEHATLAS - restricted by APF; IEHPROGM - restricted by APF; IEHDASDR - restricted by APF, and automatically invoked by AMDSADMP; IEBCOPY - supports new LOAD/UNLOAD functions for library distribution.
Dynamic support system (DSS)	New system debugging tool for authorized maintenance personnel.
Missing interruption checker	Now standard (was previously optional).
Online test executive program (OLTEP)	Executes in nonpageable storage, except for logout analysis; now standard (previously optional).
Problem determination	None.
Reliability data extractor (RDE)	None. (See the chapter "Options".)
Service aids	AMAPTFLE - supports independent component releases; AMASPZAP - restricted by APF.
Storage dumps	Provides dumps of real and/or virtual storage; new DSS dump facility.

Figure 14. VS2 Changes to Standard Support Programs

OS/VS2 Assembler

The OS/VS2 assembler is a standard facility that translates source statements into machine language, assigns virtual storage locations to instructions and other elements of the program, and performs auxiliary assembler functions designated by the programmer. (The auxiliary functions assist the programmer in such areas as checking and documenting programs, generating macro instructions, and controlling the assembler itself.)

The output of the assembler program is the object program, a machine-language translation of the source program. The assembler produces a printed listing of the source statements and object program statements, and additional information useful to the programmer in analyzing his program. The object program is in the format required by the linkage editor.

The OS/VS2 assembler is the only language translator distributed with the VS2 system control program. For a detailed description of the assembler, see *OS/VS and DOS/VS Assembler Language,* GC33-4010.

Major Changes from MVT: This facility is not available in MVT. It provides improved diagnostics and extended language capability over assembler F available in MVT.

Linkage Editor F and Loader

The linkage editor F and the loader are standard facilities of VS2 that are used to prepare the output of language translators for execution.

Linkage editor processing is a necessary step that follows source program assembly. Input to the linkage editor may consist of a combination of object modules, load modules, and control statements. The primary purpose of the linkage editor is to combine and edit these modules into a single load module that can be brought into virtual storage by program fetch and then executed.

The loader combines the basic editing and loading functions of the linkage editor and program fetch in one job step. It prepares the executable program in virtual storage and passes control to it prior to execution. The loader is designed for high-performance loading of modules that do not require the special processing facilities of the linkage editor and program fetch. It does not produce load modules for program libraries.

For a detailed description of the linkage editor and loader, see *OS/VS Linkage Editor and Loader,* GC26-3813.

Major Changes from MVT: In VS2, several new control statements and parameters are available to order CSECTs and common areas in a load module, to align CSECTs and common areas on a page boundary, and to designate the authorization code for APF. In VS2, linkage editor E is not distributed with the system control program; that is, only linkage editor F is distributed with VS2.

Utilities

The VS2 utilities are standard programs that organize and maintain data. They are divided into three categories: system utilities, data set utilities, and independent utilities. Figure 15 lists these utilities. For a detailed description of the utilities, see *OS/VS Utilities,* GC35-0005.

System Utilities	Data Set Utilities	Independent Utilities
IEHATLAS	IEBCOMPR	IBCDASDI
IEHDASDR	IEBCOPY	IBCDMPRS
IEHINITT	IEBDG	ICAPRTBL
IEHLIST	IEBEDIT	
IEHMOVE	IEBGENER	
IEHPROGM	IEBISAM	
IFHSTATR	IEBPTPCH	
	IEBTCRIN	
	IEBUPDTE	

Figure 15. VS2 Utilities

System Utilities

System utility programs manipulate collections of data and system control information. The programs are executed or invoked through the use of job control statements and utility control statements.

IEHATLAS

The IEHATLAS program is used when a defective track is indicated by a data check or missing address marker condition. It locates and assigns an alternate track to replace the defective track. Usable data records on the defective track are retrieved and transferred to the alternate track. The bad record from the defective track is replaced on the alternate by a correct copy. (The correct copy must be provided by the user.)

Major Changes from MVT: In VS2, use of this utility is restricted through the authorized program facility.

IEHDASDR

The IEHDASDR program prepares direct access volumes for VS2 use and ensures that any permanent hardware errors (i.e., defective tracks) encountered on direct access volumes will not seriously degrade the performance of those volumes. During generation of the stand-alone dump service aid program via the AMDSADMP macro instruction, IEHDASDR is invoked to dump the program on a residence volume.

In addition, the IEHDASDR program can write the entire contents or portions of a direct access volume onto a volume or direct access device type, onto a magnetic tape volume or volumes, or onto a system output device. Data that is written onto a magnetic tape volume is arranged so that it can subsequently be "restored" to its original organization by the IEHDASDR program. The direct access device types supported by the IEHDASDR program are IBM 2305 Fixed Head Storage Models 1 and 2, IBM 2314 Direct Access Storage Facility, IBM 2319 Disk Storage, and IBM 3330 Series Disk Storage.

IEHDASDR has been enhanced to allow the user to construct his own IPL program, and have it (and all IPL records necessary to initialize it) written on track 0 of a supported device.

The IEHDASDR program can be used to:

- Check tracks, assign alternate tracks for defective tracks, and perform initialization and formatting functions to make a direct access volume suitable for VS2 use. Surface analysis does not apply for the IBM 3330 Series Disk Storage.
- Initialize and format direct access devices.
- Assign alternate tracks for specified defective or questionable tracks on disk volumes.
- Create a backup or transportable copy of a direct access volume, or list the contents on a system output device.
- Copy data from a magnetic tape volume onto a direct access volume.

Major Changes from MVT: In VS2, use of this utility is restricted through the authorized program facility. Also, in addition to the enhancements noted above, the AMDSADMP service aid program now creates the job stream for the IEHDASDR program.

IEHINITT

The IEHINITT program places VS2 volume labels written in EBCDIC, in BCD, or in ASCII (American Standard Code for Information Interchange) onto any number of magnetic tapes mounted on one or more tape drives. Each volume label created by the program contains:

- A standard volume label with a user specified serial number and user identification.
- A dummy header label (an 80-byte record containing HDR1 and an ASCII character).
- A tapemark.

Major Changes from MVT: None.

IEHLIST

The IEHLIST program can be used to list:

- Entries in a catalog.
- Entries in the directory of one or more partitioned data sets.
- Entries in a volume table of contents, in edited or unedited form.

Major Changes from MVT: None.

IEHMOVE

The IEHMOVE program moves or copies logical collections of VS2 data.

The program can be used to move or copy:

- A data set residing on one or as many as five volumes.
- A group of cataloged data sets.
- A catalog, or portions of a catalog.
- A volume of data sets.

The scope of a basic move or copy operation can be enlarged by:

- Merging members from two or more partitioned data sets.
- Including or excluding selected members.
- Renaming moved or copied members.
- Replacing selected members.

Major Changes from MVT: None.

IEHPROGM

The IEHPROGM program provides facilities for modifying system control data and for maintaining data sets at an organizational level.

The program can be used to:

- Scratch a data set or a member.
- Rename a data set or a member.
- Catalog or uncatalog a data set.
- Build or delete an index or an index alias.
- Connect or release two volumes.
- Build and maintain a generation data group index.
- Maintain data set passwords.

Major Changes from MVT: In VS2, use of this utility is restricted through the authorized program facility.

IFHSTATR

The IFHSTATR program selects, formats, and writes information from type 21 (error statistics by volume) records. The records exist on the IFASMFDP tape. They can also be retrieved directly from SYS1.MANX or SYS1.MANY data sets (on a direct access storage device); however, the IFHSTATR program does not clear the SYS1.MANX or SYS1.MANY data sets and therefore does not make them available for additional records.

Major Changes from MVT: None.

Data Set Utilities

Data set utility programs manipulate partitioned, sequential, or indexed sequential data sets. Data ranging from fields within a logical record to entire data sets can be manipulated. The programs are executed or invoked through the use of job control statements and utility control statements.

IEBCOMPR

The IEBCOMPR program compares two identically organized data sets at the logical record level. Data sets to be compared can be either sequential or partitioned.

The program can:

- Verify a back-up copy of a sequential or partitioned data set.
- Verify portions of records within a sequential or partitioned data set.

User exits are provided at appropriate places for optional user routines that process user labels, handle error conditions, and modify source records.

Major Changes from MVT: None.

IEBCOPY

The IEBCOPY program can copy or merge partitioned data sets. Specified members of partitioned data sets can be selected for, or excluded from a copy operation. The program can also compress a data set in place, optionally replace identically named members on data sets, or optionally rename selected members.

The IEBCOPY program automatically lists the number of unused directory blocks and unused tracks available for member records in the output partitioned data set. By means of the LIST=NO operand the program can be made to suppress the names of copied members listed by input partitioned data set.

Major Changes from MVT: In VS2, this utility supports LOAD/UNLOAD functions which can be used for VS2 library distribution.

IEBDG

The IEBDG (data generator) program provides a pattern of test data to be used as a programming debugging aid. A (test) data set, containing records of any format, can be created by utility control statements, with or without input data.

An optional exit is provided to a user routine that can monitor each output record before it is written. Sequential, indexed sequential, and partitioned data sets can be used for input or output.

Major Changes from MVT: None.

IEBEDIT

The IEBEDIT program can create an output data set containing selected jobs or job steps. At a later time, the data set can be used as an input stream for job processing.

Input to the IEBEDIT program is obtained from a sequential data set. The input data set can reside on any input device supported by VS2 (e.g., magnetic tape, direct access, or card reader). The program can select JOB statements, JOBLIB statements, and job steps from the input data set and can include them in the output data set.

Major Changes from MVT: None.

IEBGENER

The IEBGENER program can copy a sequential data set or a partitioned member, or it can create a partitioned data set from a sequential data set or a partitioned member. The program expands existing partitioned data sets by creating partitioned members and merging them into the data set to be expanded. The program can also reblock or change the logical record length of a data set, or create user labels on sequential output data sets.

The IEBGENER program provides optional editing facilities with all applications. In addition, it provides user exits at appropriate places for routines that process labels, manipulate input data, create keys, and handle uncorrectable input/output errors.

Major Changes from MVT: None.

IEBISAM

The IEBISAM program can copy an indexed sequential data set directly from one direct access volume to another. Alternatively, the IEBISAM program can reorganize an indexed sequential data set into a sequential data set and place that data set on a direct access or on a magnetic tape volume. The data set is in a form that can be subsequently loaded; that is, it can be converted back into an indexed sequential data set.

Optionally, the IEBISAM program can be used to print the records of an indexed sequential data set.

Major Changes from MVT: None.

IEBPTPCH

The IEBPTPCH program prints or punches all, or selected portions, of a sequential or partitioned data set. Records can be printed or punched to meet either standard specifications or user specifications.

The IEBPTPCH program provides optional editing facilities. In addition, user exits are provided at appropriate places for routines that process labels and/or manipulate input or output records.

Major Changes from MVT: None.

IEBTCRIN

The IEBTCRIN program reads input from the IBM 2495 Tape Cartridge Reader (TCR), edits the data as specified by the user, and produces a sequentially organized output data set. The input consists of cartridges written by either the IBM Magnetic Tape SELECTRIC Typewriter (MTST) or the IBM 50 Magnetic Data Inscriber. An input data set (one or more cartridges) must consist of either all MTST cartridges or all Magnetic Data Inscriber cartridges.

The program can construct records from the stream of data bytes read sequentially from the Tape Cartridge Reader. The user has the option of gaining temporary control (via a user-supplied exit routine) to process each logical record.

The output produced by the program is a sequential data set that can be written on any QSAM-supported output device (for example, a system output device, a magnetic tape volume or a direct access volume). A second sequential data set may be produced for error records.

Major Changes from MVT: None.

IEBUPDTE

The IEBUPDTE program incorporates both IBM- and user-generated source language modifications into sequential or partitioned data sets. The input and output data sets may contain blocked or unblocked logical records with record lengths of up to 80 bytes. Exits are provided at appropriate places for user routines that process header and trailer labels.

The program can:

- Add, copy, and replace members or data sets.
- Add, delete, replace, and renumber the records within an existing member or data set.
- Assign sequence numbers to the records of a member or data set.
- Convert sequential input into partitioned output or vice versa.

In general, the program may be used to:

- Create and update symbolic libraries.
- Incorporate changes to partitioned members or sequential data sets.
- Change the organization of a data set from sequential to partitioned or vice versa.

Major Changes from MVT: None.

Independent Utilities

Independent utility programs are used to prepare direct access devices for system use and to ensure that any permanent hardware errors incurred on a direct access device do not seriously degrade the performance of that device. They operate outside, and in support, of System/370. They do not support the IBM 3066 System Console, which is only used with the System/370 Models 165II and 168. The user controls the operation of independent utility programs through utility control statements. Since the programs are independent of the operating system, job control statements are not required.

IBCDASDI

The IBCDASDI (DASDI) program performs two separate functions: it initializes direct access volumes for use with the operating system, and assigns alternate tracks on non-drum, direct access storage volumes. A single job can initialize one volume or assign alternates for specified tracks on one volume. DASDI jobs can be performed continuously by stacking complete sets of control statements.

Major Changes from MVT: None.

IBCDMPRS

The IBCDMPRS (DUMP/RESTORE) program dumps and restores the data on direct access volumes. The contents of a direct access volume (all data except the home address and the count field of record zero (R0)) can be "dumped" onto IBM 2314 Direct Access Storage Facility, IBM 2319 Disk Storage, or IBM 3330 Series Disk Storage devices or onto magnetic tapes, and restored onto a direct access volume that resides on the same type of device as the source volume. Both the source volume and the volume onto which data is to be restored must have been initialized to System/370 specifications. This utility is useful for preparing transportable copies and backup copies of direct access volume contents.

Major Changes from MVT: None.

ICAPRTBL

The ICAPRTBL (IBM 3211 Buffer-Loader) program loads the universal character set (UCS) buffer and the forms control buffer (FCB) for a IBM 3211 Printer.

When the IBM 3211 Printer is assigned as the output portion of a composite console and an unsuccessful attempt has been made to IPL because the UCS and FCB buffer contains improper bit patterns, this utility can load the buffers so the system can be initialized by the IPL program.

Under normal circumstances, where an operable console printer-keyboard is available, the buffers will be loaded under control of the operating system.

Major Changes from MVT: None.

Reliability, Availability, Serviceability (RAS)

Reliability, availability, and serviceability (RAS) facilities consist of recovery and repair procedures that are designed to reduce the frequency and impact of system interruptions caused by hardware failures. The RAS facilities are designed to improve the reliability of the hardware, to increase the availability of the computing system, and to improve the serviceability of the system hardware components.

Dynamic Support System (DSS)

The dynamic support system (DSS) is a debugging tool that can be used by authorized maintenance personnel, such as an IBM program systems representative, to help identify and correct causes of software failures.

DSS has its own input/output capability, and has access to all of virtual storage as well as real storage. When running, it is stand-alone and has control of the system; however, DSS can return control to the VS2 control program for further operations without system restart processing. Since DSS takes control from the system on each activation, time dependencies cannot be maintained. Thus, DSS should not be used while a time-sensitive program (for example, a teleprocessing or time sharing task) is running. DSS could be used in development testing of these programs or in locating and repairing a critical problem which prevents the execution of these programs, but it should not be invoked during any time-dependent production run.

No permanent changes can be made by DSS. That is, any modification made to the system will not be carried over to the next IPL. Also, DSS cannot be used to modify itself, IPL, or NIP.

Communication with DSS is by commands entered through the integrated operator's console. The DSS language provides capabilities to:

- Display and alter real storage, virtual storage, and registers.
- Provide control for the program event recording (PER) hardware of System/370.
- Stop operation of the system at any instruction or PER interruption, perform maintenance functions (that is; identify and correct the causes of the failure), and resume operations.
- Save information within DSS for later use, or write out the information on tape or high-speed printers.
- Use tape or card readers for secondary command input devices.
- Write procedures that are used for reiterative command sequences.

For a detailed description of DSS, see *OS/VS Dynamic Support System,* GC28-0640.

Major Changes from MVT: This facility is not available in MVT.

Missing Interruption Checker

Missing interruption checker is a standard facility of VS2 that notifies the operator if a device-end or channel-end interruption is not received within a specified period of time. The absence of such interruptions may mean that a mount message has not been satisfied or that a device has malfunctioned.

Specific actions an operator may have to take depend upon the conditions he encounters. He may be required to ready a device on which a volume has been mounted, examine indicator lights on the device for abnormal signs, or terminate the job.

Major Changes from MVT: This facility is available in MVT only as a program temporary fix (PTF).

Online Test Executive Program (OLTEP)

The online test executive program (OLTEP) acts as an interface between the operating system and online test programs that are to be run. OLTEP schedules and controls the activities of the test programs. The programs can be used to:

- Test control units and input/output devices.
- Diagnose equipment malfunctions.
- Verify repairs.
- Test engineering changes.
- Perform periodic maintenance checks.

While OLTEP is being run, continuous communication is maintained with the operator. The programs can be run to obtain printed diagnostic information, or they can be run to exercise a device while it is being tested with hardware testing equipment.

For a detailed description of OLTEP, see *OS/VS OLTEP,* GC28-0636.

Major Changes from MVT: In VS2, OLTEP is a standard facility; in MVT, it was optional. In VS2, OLTEP executes in pageable storage only when it is invoked to do logout analysis (LOA); otherwise, OLTEP can be executed only in nonpageable storage.

Problem Determination

Problem determination can be thought of as an activity that is required to identify a failing hardware unit or program and determine who is responsible for maintenance. It helps to establish whether a failure is in the system control program, in a software component added to the system by the user (type I programs and program products), or in the user's application programs.

Problem determination is accomplished with procedures specified by IBM. In some cases, these procedures may be initiated by a message or code issued by the system control program; the message or code, in turn, may direct the user to take certain definitive actions such as running a service aid or utility program. In all cases, a definite action will be specified and should be taken before a service call is to be placed to IBM for either hardware or programming assistance.

Major Changes from MVT: None.

Service Aids

The service aids help in diagnosing and repairing system or application program failures. For a detailed description of the IFCDIP00 and IFCEREP0 service aids, see *OS/VS SYS1.LOGREC Error Recording,* GC28-0638. For a detailed description of the other service aids, see *OS/VS Service Aids,* GC28-0633.

AMAPTFLE

The AMAPTFLE service aid is a problem program that is used to apply program temporary fixes (PTFs). AMAPTFLE has two functions:

- An application function generates control statements and applies PTFs in one operation.
- A generate function produces the JCL and control statements needed to apply PTFs. The JCL must be executed in a later, separate step.

Major Changes from MVT: In VS2, AMAPTFLE will support independent component releases.

AMASPZAP

The AMASPZAP service aid is a problem program that allows the user to inspect and modify data at the time a problem is diagnosed. Various combinations of AMASPZAP control statements enable the user to:

- Inspect and modify instructions and data in any load module that exists as a member of a partitioned data set.
- Inspect and modify data in a specific data record that exists in a data set residing on a direct access device.
- Dump an entire data set, a specific member of a partitioned data set, or any portion of a data set residing on a direct access device.
- Update the system status index (SSI) in the directory entry for a load module.
- Update the AMASPZAP identification record (IDR) for a load module.

Major Changes from MVT: In VS2, the use of this service aid is restricted through the authorized program facility.

AMBLIST

The AMBLIST service aid is a problem program that assists in problem determination by providing:

- Formatted listings of linkage editor/loader input and output (that is, object and load modules).
- A cross-reference listing of symbolic references within a load module without re-processing that module through the linkage editor.
- A formatted listing of all information in a module's control section (CSECT) identification records (IDR).
- A formatted listing of all IDR indications of program modifications for a load module or library.
- A map of a system's nucleus.
- A map of the pageable link pack area (LPA). (Note that this map is not of the modified LPA or fixed LPA.)

Major Changes from MVT: None.

AMDPRDMP

The AMDPRDMP service aid is a problem program that formats and prints the contents of the AMDSADMP output data set, the SYS1.DUMP data set, the TSO DUMP data set, and the GTF output data set. Users can control AMDPRDMP output with control statements. Some of the areas and traces that can be printed are:

- Active modules in the link pack area.
- QCB trace.
- System control blocks for all active tasks.
- Allocated pageable storage.
- Virtual storage ranges.
- Real storage ranges.
- Page data set dumps.
- System nucleus.
- GTF trace data.

Major Changes from MVT: In VS2, areas of virtual storage are printed; in MVT, only areas of main storage were printed.

AMDSADMP

The AMDSADMP macro instruction is used to generate a stand-alone service aid program that provides the user with a flexible, installation-tailored dump facility. It can generate one of the following types of dump programs:

- *Low-speed* - The low-speed dump program dumps the contents of real storage to a printer or tape. The dump output is translated and deblocked. Output which is directed to tape may be printed by the AMDPRDMP service aid or the IEBPTPCH utility program. (IPL of the program must be from a direct access volume.)
- *High-speed* - The high-speed dump program dumps the contents of real storage to tape. Optionally, it can aiso produce dumps of page data sets. The dump output is in large (2K), untranslated hexadecimal blocks. The AMDPRDMP service aid must be used to format, translate, and print the output. (IPL of the program may be from either a tape or a direct access volume.)

Major Changes from MVT: In VS2, the high-speed dump program can produce dumps of the page data set(s).

Generalized Trace Facility (GTF)

The generalized trace facility (GTF) service aid is a problem determination aid that can be used to trace software (system or problem program) behavior. GTF uses the monitor call (MC) instruction to detect occurrences of system events and to create trace records. Problem programs may also use a GTF macro instruction (GTRACE) to record problem program data in the trace data set.

GTF can single out those programming activities to be traced within the system, including I/O interruptions, program interruptions, and supervisor call interruptions.

The output from GTF can be a trace data set that is used with the edit function of the AMDPRDMP service aid. The AMDPRDMP service aid provides the capability to format specific trace activities. It operates as a problem program that can be called via the job control language.

Major Changes from MVT: None.

IFCDIP00

The IFCDIP00 service aid is a problem program that is used to:

- Initialize the SYS1.LOGREC data set during system generation.
- Reinitialize the SYS1.LOGREC data set if an error has resulted in the destruction of the SYS1.LOGREC header record and makes the data set unusable.
- Reallocate the SYS1.LOGREC data set to increase or decrease the space allocation.

This service aid is run and controlled by job control statements; no user or utility control statements are needed.

Major Changes from MVT: None.

IFCEREP0

The IFCEREP0 service aid is a problem program that is used to:

- Select and format environment records from the SYS1.LOGREC data set and write them to an output device.
- Select environment records from the SYS1.LOGREC data set and accumulate them on a history data set.
- Write the accumulated records on the history data set to an output device.
- Summarize the information contained in the records on the SYS1.LOGREC or history data sets.
- Process (edit, write, accumulate, and summarize) records produced on different machine models.

This service aid is run and controlled by job control statements; no user or utility control statements are needed.

Major Changes from MVT: None.

IMCOSJQD

The IMCOSJQD service aid is a problem program that produces a formatted copy of the contents of the job queue data set. It does not alter the status of the records to be written out. The user does not need to know the explicit address of the job queue data set, only the address assigned to the direct access device on which the volume containing the job queue is mounted.

When the program locates the job queue, the records are read and, either serially or selectively, identified by type and address, formatted, and written out. The job queue may be written out by job, job step, or in its entirety.

This service aid is run and controlled by job control statements; no user or utility control statements are needed.

Major Changes from MVT: None.

Storage Dumps

Storage dumps give the user a meaningful image of his programs. The routines available to provide dumps include:

- ABDUMP, which provides a formatted dump of selected areas of a problem program's virtual storage. ABDUMP is invoked by the SNAP and ABEND macro instructions.
- DSS DUMP, which allows the user of the dynamic support system (DSS) to dump both real and virtual storage. DSS DUMP is invoked by DSS.
- CONSOLE DUMP, started by the console operator, invokes SVC DUMP, described below. See the discussion of the DUMP command in the chapter "Compatibility".
- TSO DUMP, invoked by the time sharing option (TSO), invokes SVC DUMP, described below.
- SVC DUMP, provides a dump of selected areas of virtual storage (the nucleus, system queue area, local system queue area, region, and link pack area modules used). SVC DUMP is invoked by a key zero routine. The dump provided by SVC DUMP can be printed by the AMDPRDMP service aid.
- SADMP DUMP, which provides an installation-tailored dump of virtual storage, and can be printed by the AMDPRDMP service aid. SADMP DUMP is invoked by the AMDSADMP service aid.

For a detailed description of storage dumps, see *OS/VS2 Debugging Guide,* GC28-036. 2

Major Changes from MVT: In VS2, dumps are provided of real storage and/or selected areas of virtual storage; in MVT, dumps are for main storage only. Also, the DSS facility is not available in MVT.

Options

The items discussed in the preceding chapter are standard support programs of the VS2 system control program. In addition to the standard support programs, the user can add optional facilities to his system. There are two types of options: options included in the system during system generation and options included in the system after system generation.

Options Included During System Generation

Options included in the system during system generation are contained in the distribution library and added to the system during the IBM installation process. Figure 16 provides a list of these options and their changes from MVT.

Facility	Change
Alternate path retry (APR)	None.
Automatic priority group (APG)	New dispatching facility.
Automatic volume recognition (AVR)	None.
Device independent display operator console support (DIDOCS)/status display support (SDS)	None.
Reliability data extractor (RDE)	None.
Shared direct access storage devices (shared DASD)	None.
Time sharing option (TSO)	Support for swapping performed by VS2 paging supervisor; new use of virtual storage to support more TSO regions; new parameters on START and MODIFY commands; increased number of regions (from 14 to 42) for foreground jobs; parallel swapping allowed to four devices.
Time slicing	None.
Track stacking	None.
Access methods	Support for TCAM, VSAM, BTAM, BISAM, QISAM, and GAM.

Figure 16. VS2 Changes to Options

Time Sharing Option (TSO)

The time sharing option (TSO) of VS2 adds general purpose time sharing to the VS2 control program. The installation can dedicate the system to time sharing operations, or it can run concurrent time sharing and batch operations.

TSO adds a number of capabilities to the facilities already available through the VS2 control program:

- It gives users access to the system through a *command language* which is entered at *remote terminals* -- typewriter-like keyboard-printer or keyboard-screen devices connected through telephone or other communication lines to the computer.
- It gives those who may not be programmers the use of data entry, editing, and retrieval facilities.
- It makes the facilities of the operating system available to programmers at remote terminals to develop, test, and execute programs conveniently, without the job turnaround delays typical of batch processing. Both terminal-oriented and batch programs can be developed at terminals.
- It allows the management of an installation to dynamically control the use of the system's resources from a terminal.

- It creates a time-sharing environment for terminal-oriented applications. Some applications, such as problem-solving languages, terminal-oriented compilers, and text-editing facilities, are available as IBM program products. Installations can add others suited to their particular needs.

TSO consists of control routines, service routines and command processors, and a number of IBM program products. All the facilities available with TSO in MVT are available with TSO in VS2. VS2 enhances TSO in the following ways:

- The use of virtual storage makes it possible to run more foreground regions.
- The use of virtual storage allows the installation to place frequently used TSO commands or service routines in the pageable link pack area or TSO link pack area without reducing the amount of real storage available. When TSO is not active, the TSO LPA does not require address space.
- The paging supervisor, which does all swapping, uses external page storage for swapping, thereby eliminating the swap data set.

For a detailed description of TSO, see *OS/VS2 TSO Guide,* GC28-0644.

Differences in TSO for VS2

TSO in VS2 differs from TSO in MVT in several respects:

- The number of regions that can be assigned to foreground jobs is increased from 14 for TSO in MVT to 42 for TSO in VS2.
- The paging supervisor executes all swaps.
- The LSQA is no longer taken from the region; each TSO region is allocated one segment of LSQA.
- Swapping is now a process by which the valid pages (addressable pages) in a user's region are usually block paged to external page storage and always block paged from external page storage.
- The amount of external page storage required in VS2 must be sufficient to accomodate ('back up') the paging of background pages and the paging and swapping of foreground pages.
- Swapping can be directed to specific page data set devices. The parallel swapping capability that existed for TSO in MVT is expanded to allow swapping to as many as four devices. These can be the same devices that receive pages from background jobs.
- Background regions must have 100% backup in external page storage. For example, a background job that requests a 128K region is allocated 128K (32 pages) of external page storage as backup.
- Most TSO jobs do not continually need an amount of external page storage equivalent to the region sizes requested. Consequently, foreground jobs are backed up by an installation specified amount of external page storage that is a percentage of the total amount of external page storage required.

TSO in VS2 uses eight new parameters, including parameters to handle the assignment of external page storage and establish thresholds to guarantee that both background and foreground jobs have minimum amounts of external page storage. Figure 17 defines the parameters and Figure 18 shows the relationship between the TSOAUX and TSOMAX parameters in a system.

Parameter	When Specified	Default	Effect
TSOAUX	System generation, or system initialization, or START TSO, or MODIFY TSO	0 %	Specifies the minimum percentage of virtual storage dynamic area reserved for foreground jobs and backed up by external page storage. By implication, this parameter also specifies the maximum percentage of virtual storage dynamic area available for background jobs and backed up by external page storage.
TSOMAX	START TSO or MODIFY TSO	100 %	Specifies the maximum percentage of virtual storage dynamic area available for foreground jobs and backed up by external page storage. By implication, this parameter also specifies the minimum percentage of virtual storage dynamic area reserved for background jobs and backed up by external page storage.
BACKUP	START TSO or MODIFY TSO	100 %	Specifies the percentage of virtual storage dynamic area assigned to foreground jobs that will be backed up by external page storage.
SWAP	START TSO	None	Specifies up to 4 volume serial numbers of volumes containing page data sets to which swapping is to be directed. Parallel swapping is accomplished when two or more devices are specified. If no devices are specified, the primary page data set with the most page slots available will be used, or, if a secondary device is defined, it will be used.
NOSWAP	START TSO	NOSWAP	Specifies that the regular page data set is to be used for swapping. This parameter nullifies the SWAP parameter.
AUXLIST	START TSO or MODIFY TSO	None	Specifies that information concerning the use and availability of auxiliary storage is to be listed.
LPAR	START TSO	None	Specifies TSO LPA modules required. The START TSO command will be rejected unless the named modules are found in SYS1.LINKLIB or the TSO link pack area.
LPAF	START TSO	None	Specifies that, in addition to the processing for the LPAR parameter, the named TSO modules will be fixed in the TSO link pack area as long as TSO is active.

Figure 17. New TSO Operator Parameters for VS2

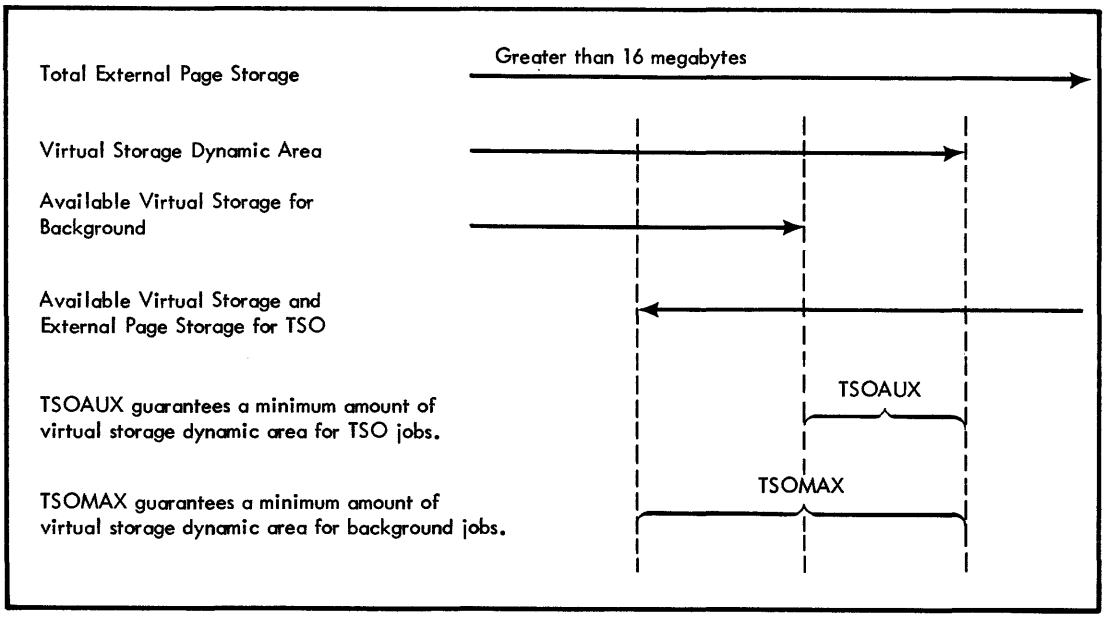


Figure 18. Comparison of TSOAUX and TSOMAX Parameters

Basic Preparation

The procedure to prepare TSO in VS2 is basically the same as it was in MVT. To prepare TSO, the installation must:

- Generate a system including TSO.
- Tailor a TSO-TCAM message control program.
- Define a member of SYS1.PARMLIB to contain TSO system parameters.
- Define LOGON procedures.
- Define the user attribute data set (UADS) to contain user identification records.

Major Changes from MVT: See the section "Differences in TSO for VS2" above. For a summary of all changes to the TSO operator commands, see "TSO Commands" in the chapter "Compatibility."

Automatic Volume Recognition (AVR)

Automatic volume recognition (AVR) is an option of VS2 that permits the operator to premount direct access or tape volumes on any available devices prior to the initiation of the job step that requires the volumes. The operator can therefore reduce the time lost in performing job setups.

If the operator does not premount tape volumes, AVR allows him to select devices he wishes to use during job step initiation. AVR also allows him to manually restart an I/O job without demounting volumes and then remounting the same volumes.

In VS2, the system attempts to balance channel loads by using the I/O load balancing algorithm to allocate data sets with nonspecific device requests. In some installations, however, a system slowdown may result if AVR causes one channel to be overloaded with most of the data handling.

Major Changes from MVT: None.

Device Independent Display Operator Console Support (DIDOCS)/Status Display Support (SDS)

Device independent display operator console support (DIDOCS)/status display support (SDS) is an option of VS2 that allows graphic display devices to be used as operator consoles. Its use can result in faster communication between the system and the operator than can be achieved with standard printer-keyboard or composite console devices. DIDOCS/SDS is not a standard facility of VS2; however, it will be included in the system when a graphic console is included. It provides the following advantages to the operator:

- He can respond to a message or enter a command while messages are being written to the screen.
- He is shown all action messages to be answered, and can delete any he no longer needs.
- He can use the light pen or cursor, when available, to delete messages and perform other display-oriented functions.
- He can obtain an in-line display or an out-of-line display within specified screen display areas.
- He can obtain a dynamic status display from a MONITOR ACTIVE command.
- He can initiate automatic command entry either by the use of the selector pen or from the program function keyboard (PFK) by an operator command.

Although multiple console support (MCS) is required for DIDOCS, MCS is standard in VS2, and need not be specified separately.

Major Changes from MVT: None.

Time Slicing

Time slicing is an option of VS2 that lets each task of a specified priority have control of the central processing unit (CPU) for a specified interval of time. Normally a task maintains control either until it is complete, until a higher-priority task becomes ready, or until it must wait for some event (such as an I/O operation). With time slicing, a group of tasks are allotted an interval of time which is divided among them.

When a member of the time-slice group has been active for the allotted length of time, it is interrupted and control is given to another member of the group which will, in turn, have control of the CPU for the same amount of time. In this way, all member tasks are given an equal slice of CPU time and no task within the group can monopolize the CPU. Only tasks in the group are time-sliced, and they are time-sliced only when the priority level of the group is the highest priority level that has a ready task.

When a time-sliced task loses control prior to the expiration of its interval, the remainder of the interval is not saved. If the task was waiting for an event, the next task is dispatched. If a task of higher priority became ready, the higher priority task is dispatched when control is returned to the group, not the task that lost control.

The priority level identified for the time slice group cannot also be identified for the automatic priority group (APG).

Major Changes from MVT: None.

Shared Direct Access Storage Devices (Shared DASD)

Shared direct access storage devices (shared DASD) is an option of VS2 that enables independent operating systems (VS1, VS2, MVT, and MFT) to share common data on a shared direct access storage device.

The sharing is made possible by a two-channel switch that allows the control unit for the device to be switched between two channels, each from a different system. (A 4-channel switch is supported by the IBM 3330 Series Disk Storage device.) The VS2 control program

protects data being used in one central processing unit (CPU) from modification by a program in the other CPU, and minimizes access-arm contention between the devices.

Shared DASD offers the following advantages:

- Reduction of time spent in data set creation and maintenance, and a reduction in space requirements (whenever a single data set can service multiple users).
- Flexibility in scheduling jobs on CPUs that have access to the common data base.
- Availability of a system through the flexibility of job scheduling in the event of a malfunction in any CPU.
- Increase in the amount of work processed, since two CPUs are using common data at the same time.

In a shared DASD environment, volume handling is conducted in parallel on both sharing systems. Therefore, the operator's responsibility is increased, and good communication between operators of the systems must be maintained.

For a listing of the system data sets and libraries that can be shared, see System Data Sets in the chapter "Compatibility".

Major Changes from MVT: None.

Alternate Path Retry (APR)

Alternate path retry (APR) is an option of VS2 that ensures that an alternate path to a device is tried (whenever possible) when a failing path is detected. Until a retry is successful, APR places any failing paths offline to prevent further retries from being initiated on these paths. When retry succeeds, APR restores all of the original paths, and resumes normal system operation.

APR also permits the operator to vary a path to a device online or offline. The operator, however, cannot vary the last remaining path to a device offline, nor can he vary teleprocessing paths or paths to shared direct access storage devices.

Major Changes from MVT: None.

Reliability Data Extractor (RDE)

The reliability data extractor (RDE) is an option of VS2 that generates system initialization (IPL) and system termination (EOD) records, and collects them on SYS1.LOGREC. The IPL record contains the reason for restarting the system and names the type of equipment or program (if any) that is responsible for the restart; the EOD record defines the time span of processing for the restart. The IPL and EOD records can be examined after use of the IFCEREP0 service aid program. The program can:

- Print the IPL and EOD records, and print a summary of all the IPL and EOD records that are on SYS1.LOGREC; this permits examination of the records to determine the reasons for repeated system initializations.
- Write the records from SYS1.LOGREC to a measurement data set (generally a tape). The information on SYS1.LOGREC includes, in addition to the IPL and EOD records, error recording records from the SDR, MCH, TPER, CCH, and OBR error recording routines.

For a detailed description of RDE, see *OS/VS RDE Guide,* GC28-0642.

Major Changes From MVT: None.

Track Stacking

Track stacking is an option of VS2 that is used to handle the data brought in from the SYS1.SYSJOBQE data set. It permits one or more logical tracks for a particular job to reside temporarily in storage as an ordered series.

If a larger part of the computer time at an installation is spent in job scheduling, performance can be improved by using track stacking. Track stacking allows the moving of an entire locical track from the input queues into storage for use by the initiators; this reduces the number of accesses and thus reduces arm interference. Track stacking may be specified at system generation time or at IPL time; if the track stacking facility is used, region sizes for the initiators are increased to include the additional buffer space.

Major Changes From MVT: None.

Automatic Priority Group (APG)

Automatic priority group (APG) is an option of VS2 that assigns a single priority level to a group of tasks in an attempt to provide optimum use of CPU and I/O resources by these tasks. The priority level may be identified at system generation or system initialization time.

Tasks are dispatched and shifted within the APG based on their characteristics. Tasks are considered CPU-oriented if they appear to be oriented more towards use of the CPU resource. Tasks are considered I/O-oriented if they seem more inclined towards use of I/O (paging I/O is excluded from this determination).

All tasks in the APG are dispatched with a time interval. A task is considered CPU-oriented in APG if it uses its entire interval of allotted time - that is, if it does not give up control voluntarily. A task is considered I/O-oriented in APG if it does not use its entire interval of allotted time.

Important features of the APG group are:

- Tasks within the I/O subgroup are ordered such that those which use smaller portions of their interval are ranked higher in the queue.
- CPU tasks receive control in a cyclic manner, thus ensuring that any available CPU time is distributed equitably among them.
- Potential I/O tasks are not locked at the bottom of the CPU subgroup indefinitely.
- APG is self-adjusting within user-specified limits to maintain a mix of CPU-bound jobs to I/O-bound jobs.

The priority level identified for the APG cannot also be identified as a time slice group.

Major Changes from MVT: This facility is not available in MVT.

Access Methods

Access methods are techniques for moving data between virtual storage and input/output devices. The access methods that are optional in VS2 are the telecommunications access method (TCAM), the virtual storage access method (VSAM), the basic telecommunications access method (BTAM), the basic indexed sequential access method (BISAM), the queued indexed sequential access method (QISAM), and the graphics access method (GAM).

Telecommunications Access Method (TCAM)

The telecommunications access method (TCAM) is an option of VS2 that is a generalized I/O control system that extends the techniques of data management to the teleprocessing environment. The data sets addressed by the user (via GET, PUT, READ, and WRITE macro instructions) are queues of messages coming from, or going to, remote terminals via communications lines. If the time sharing option (TSO) is included in the system, TCAM provides the terminal support.

In addition to controlling the transfer of messages to user-written application programs, TCAM provides a flexible, message control language. TCAM macro instructions can be used to construct an installation-oriented message control program for controlling message traffic among remote terminals, and between remote terminals and application programs.

A teleprocessing control system created through the use of the TCAM message control language can:

- Establish contact and control message traffic between computer and terminals.
- Delete and insert line control characters automatically, thus freeing the user from line control considerations.
- Assign, use, and free buffers dynamically.
- Edit incoming and outgoing messages.
- Forward messages to destination terminals and application programs.
- Take corrective action and provide special handling for messages containing errors.
- Maintain statistical information about message traffic.

TCAM offers an extensive set of facilities. They include:

- Online testing of teleprocessing terminals and control units.
- Program debugging aids.
- Network reconfiguration facilities.
- Checkpoint/restart.
- I/O error recording.
- Operator control to examine and alter network status.
- Alternate destination capability.

For a description of TCAM, see *OS TCAM Concepts and Facilities,* GC30-2022 and *OS/VS TCAM Programmer's Guide,* GC30-2034.

Major Changes from MVT: TCAM message control programs must be reassembled and linkage edited before execution in VS2. Also, VS2 TCAM supports fetch protection.

Virtual Storage Access Method (VSAM)

The virtual storage access method (VSAM) is a new access method that is used with direct access storage devices. VSAM creates and maintains two types of data sets.

One type of data set is sequenced by a key field within each record and is called a key-sequenced data set. Data records are located by using the key field and an index that records each key field and the address of the associated data, similar to ISAM.

The other type of data set is sequenced by the time of arrival of each record into the data set and is called an entry-sequenced data set. Data records are located by using the record's displacement from the beginning of the data set. The displacement is called the relative byte address, similar to the relative block address used with BDAM.

VSAM stores, retrieves, and updates user data records in these types of device independent data sets. The data records are stored in a new data format designed for device independence. This ensures long term data stability and suitability for use in data base applications. Data in both types of data sets can be accessed either sequentially or directly.

VSAM enhances may ISAM capabilities including performance, device independence, concurrent processing, data portability, and kinds of accessing supported. It provides multiple levels of password security protection. It also creates and maintains separate catalogs that contain information about each VSAM data set and are used to link a data set with its index.

VSAM provides a multifunction utility program that will define, delete, print, copy, and provide backing and portability of VSAM data sets. An ISAM interface routine to allow most existing ISAM programs access to VSAM data sets is also provided.

For a detailed description of VSAM, see *OS/VS Virtual Storage Access Method (VSAM) Planning Guide,* GC26-3799.

Major Changes from MVT: This facility is not available in MVT.

Basic Telecommunications Access Method (BTAM)

The basic telecommunications access method (BTAM) is an option of VS2 that provides the basic modules for constructing a teleprocessing program, including routines for controlling a variety of terminal units, communications lines, and transmission control units. BTAM can be used as a component in the development of more sophisticated teleprocessing systems, and can be modified to support special configurations not supported by other access methods.

BTAM provides the capabilities to:

- Poll terminals and receive messages.
- Address terminals and send messages.
- Dynamically chain input buffers.
- Dial and answer.
- Detect and correct errors.
- Write output buffer chains.
- Perform code translation.

BTAM controls terminal I/O operations initiated by READ and WRITE macro instructions in the user's problem program. Although BTAM controls data transmission, it does not provide elaborate message queuing or process messages.

For a detailed description of BTAM, see *OS/VS BTAM,* GC27-6980.

Major Changes from MVT: None.

Basic Indexed Sequential Access Method (BISAM)

The basic indexed sequential access method (BISAM) is an option of VS2 that is used in one form to directly retrieve or update particular blocks of a data set on a direct access device. An index, which is used to locate the data set, is stored with the data set. Other forms of this method can be used to store or retrieve sequential blocks of the same data set. (See description of basic and indexed sequential in the discussion of standard access methods in the chapter "System Control Program.")

Major Changes from MVT: None.

Queued Indexed Sequential Access Method (QISAM)

The queued indexed sequential access method (QISAM) is an option of VS2 that is an extended version of BISAM where a queue is formed of input data waiting processing or output data awaiting transfer to auxiliary storage or output devices. (See descriptions of queued and indexed sequential in the discussion of standard access methods in the chapter "System Control Program.")

Major Changes from MVT: None.

Graphics Access Method (GAM)

The graphics access method (GAM) is an option of VS2 that consists of I/O and control routines that transfer data to and from graphic devices. GAM, part of the graphic programming services (GPS), provides the following facilities:

- Buffer management routines that allocate, control, and protect buffer storage.
- Data management routines that store graphic orders and data in user-specified output areas.
- READ/WRITE level macro instructions that transfer data between virtual storage and the graphic device buffers.
- Basic and express attention handling routines that facilitate man-machine communication.
- Error handling routines that diagnose synchronous errors and perform the necessary error handling.

When GAM is included at system generation time, problem oriented routines (POR) and the graphic subroutine package (GSP) may also be included.

For a detailed description of GAM, see *OS/VS Graphic Access Method Logic,* SY27-7240.

Major Changes from MVT: None.

Options Included After System Generation

Type I programs are free and can be added to the system after system generation. They are not distributed with the control program, and must be ordered separately.

Program products, available from IBM for a license fee, provide the user with specialized problem programs; they are added to the system after system generation and installation. (For a catalog to IBM program products that provide language and sort/merge processing support, see *System/370 Program Products Language and Sort Processors,* GC28-8200.)

Information on the availability of all type I programs and program products, and documentation on their use, is available from the local IBM sales office.

Major Changes from MVT: Any restrictions on use of the type I programs and program products are available from the local IBM sales office.

Compatibility

VS2 is an extension of the MVT configuration; VS1 is an extension of the MFT configuration. This chapter describes the differences and incompatibilities between VS2 and MVT, with references, where applicable, to VS1 and MFT. It is divided into the following sections:

- Operator commands.
- Job control language.
- Problem programs.
- Coding guidelines.
- Emulators.
- Reassembly/recompilation.
- System data sets.
- System macro instructions.
- Major VS2 - MVT differences.
- Major VS2 - VS1 differences.

Operator Commands

Operator commands are statements to the control program, issued via a console device or the input stream, which cause the control program to provide requested information, alter normal operations, initiate new operations, or terminate existing operations.

This section does not describe all of the operator commands needed to run jobs in VS2. Rather, it describes the parameters that have changed from MVT to VS2.

MVT Commands

Except for the major changes noted below, the operator commands for VS2 are the same as the commands for MVT. Figure 19 provides a summary of the changes. For a detailed description of the commands, see *Operator's Library: OS/VS2 Reference,* GC38-0210.

Command	Parameter	Change
CANCEL		Operator can cancel job waiting for region or data set.
DISPLAY	A N Q	Output changed to reflect virtual storage. RJE and ASB reader queues not supported. RJE and ASB reader queues not supported.
DUMP	ALL	Parameter not supported.
MODE		One simplified command for all models.
MONITOR	A	Output changed to reflect virtual storage.
MOUNT	H	Parameter not supported.
SET	GMT	New parameter for time-of-day clock.
START	H LSQA	Parameter not supported. New parameter for local system queue areas.
VARY	F M OFFGFX ONGFX S	Parameter not supported. Parameter not supported. Parameter not supported. Parameter not supported. Parameter not supported.

Figure 19. VS2 Changes to MVT Operator Commands

CANCEL Command

The specification of the CANCEL command is the same for VS2 and MVT. However, the command can now be used to cancel a job waiting for a region or a data set.

DISPLAY Command

The specification of the DISPLAY command is the same for VS2 and MVT. However, the output generated by this command is different in the following instances:

- If the parameter A is specified in the command, the output will include the starting and ending virtual storage addresses of the region, the number of pages allocated for the local system queue area (LSQA) for the job step, and an indication of whether the job step is executing in pageable or nonpageable storage. Also, the output now permits up to 255 active tasks to be displayed.
- If the parameters N or Q are specified in the command, the output will no longer include the remote job entry or automatic SYSIN batching reader queues, since these facilities are no longer supported.

DUMP Command

The specification of the DUMP command is the same for VS2 and MVT. However, the parameter ALL is no longer supported. Since the output generated by the DUMP command in VS2 is virtual storage, if ALL was specified the output would consist of all 16,777,216 bytes of virtual storage.

MODE Command

The specification of the MODE command is different because in VS2 one simplified MODE command can be used for all machine models.

MONITOR Command

The specification of the MONITOR command is the same for VS2 and MVT. However, the output generated by the command is different in the following instance:

- If the parameter A is specified in the command, the output will include the starting and ending virtual storage addresses of the region, the number of pages allocated for the local system queue area (LSQA) for the job step, and an indication of whether the job step is executing in pageable or nonpageable storage.

MOUNT Command

The specification of the MOUNT command is the same for VS2 and MVT. However, the parameter H (hierarchy) should not be specified. If H is specified, the system will issue a message stating that H is an invalid keyword; the operator will then have to reissue the command, omitting the H parameter.

SET Command

The specification of the SET command at IPL time is the same for VS2 and MVT. However, a new parameter, GMT, can be specified. If GMT is specified, the DATE and CLOCK values in the command are Greenwich Mean Time.

START Command

The specification of the START command is the same for VS2 and MVT. However, the parameter H (hierarchy) should not be specified. If H is specified, the system will issue a message stating that H is an invalid keyword; the operator will then have to reissue the command, omitting the H parameter.

The START command also has a new parameter, LSQA, which specifies the number of segments of LSQA to be allocated to the task being started.

VARY Command

The specification of the VARY command is the same for VS2 and MVT. However, the parameters ONGFX and OFFGFX (and F, M, and S) should not be specified. If the parameters are specified, a rejection message will be issued and the operator will have to reissue the command, omitting the parameters. These parameters have been eliminated since the graphic job processor (GJP) and the satellite graphic job processor (SGJP) are no longer supported.

TSO Commands

This section does not describe all the changes to the TSO operator commands. However, it does list the significant changes to the MODIFY and START commands for VS2. It also lists the new abbreviations that can be specified for the operator parameters. For a detailed description of the TSO commands, see *OS/VS2 TSO Command Language Reference,* GC28-0646.

MODIFY Command

The following parameters allow installation control over auxiliary storage, and can be specified on the MODIFY command. The parameters are described in detail in the discussion of TSO in the chapter "Options".

AUXLIST
BACKUP

TSOAUX
TSOMAX

The following parameters have been changed on the MODIFY command:

REGNMAX=nnn, where nnn=0-42 -- can now be specified.

REGSIZE(region-number)=xxxK, where the maximum is 896K -- replaces old REGSIZE parameter.

START Command

The following new parameters allow installation control over auxiliary storage, and can be specified on the START command. The parameters are described in detail in the discussion of TSO in the chapter "Options."

AUXLIST
BACKUP

LPAF

LPAR

NOSWAP

SWAP

TSOAUX

TSOMAX

The following parameters have been deleted from the START command:

FORM

MAP

The following parameters have been changed on the START command:

DUMP=DUMP -- replaced by DUMP

DUMP=NODUMP -- replaced by NODUMP

LIST -- no longer positional.

LPA -- can now be specified.

REGNMAX=nnn, where nnn=0-42 -- can now be specified.

REGSIZE(region-number)=xxxK, where the maximum is 896K -- replaces old REGSIZE parameter.

Parameter Abbreviations

All operator parameters are listed in Figure 20. Following each parameter is its unique abbreviation for VS2. These abbreviations are new for VS2.

Parameter	Abbreviation	Parameter	Abbreviation
ACTIVITY	AC	NOPREEMPT	NOPRE
AUXLIST	AU	NOPRIORITY	NOPRI
BACKGROUND	BACKG	NOSWAP	NOSWAP
BACKUP	BACKU	NOSWAPLOAD	NOSWAPL
CYCLES	C	NOWAIT	NOW
DECAYACT	DECAYA	OCCUPANCY	O
DECAYWAIT	DECAYW	PREEMPT	PRE
DRIVER	DRIVER	PRIORITY	PRI
DSPCH	DS	REGNMAX	REGN
DUMP	DU	REGSIZE	REGS
HOLD	H	SERVICE	SE
LIST	LI	SMF	SMF
LPA	LP	SUBMIT	SUBM
LPAF	LPAF	SUBQUEUES	SUBQ
LPAR	LPAR	SWAP	SWAP
MAXOCCUPANCY	MAXO	SWAPLOAD	SWAPL
MAXSWAP	MAXS	TERMAX	TE
MINSLICE	MI	TSCREGSZ	TSC
NOACTIVITY	NOAC	TSOAUX	TSOA
NOAVGSERVICE	NOAV	TSOMAX	TSOM
NOBACKGROUND	NOB	USERS	U
NODUMP	NOD	WAIT	W
NOOCUPANCY	NOO		

Figure 20. Parameter Abbreviations for TSO Operator Commands

Job Control Language

Every job submitted for execution by the operating system must include job control language statements. These statements contain information required by the operating system to initiate and control the processing of jobs.

This section does not describe all of the job control language needed to run jobs in VS2. Rather, it describes the parameters and subparameters that have changed from MVT to VS2. However, most jobs that run in Release 21 of MVT can be executed in VS2 without changing the job control language. Also, if the default region size in VS2 is at least as large as the partition size in VS1 and MFT, VS1 and MFT jobs that run in Release 21 of MVT can be executed in VS2 without changing the job control language. Figure 21 provides a summary of the JCL changes.

For a detailed description of JCL, see *OS/VS JCL Reference,* GC28-0618.

Statement	Parameter	Change
JOB	ADDRSPC	Assigns pageable or nonpageable storage space, and defaults to pageable storage.
	REGION	Assigns real storage space in 4K multiples and virtual storage space in 64K multiples; reassigns MVT hierarchy 1 storage.
	ROLL	Ignores this parameter.
EXEC	ADDRSPC	Assigns pageable or nonpageable storage space, and defaults to pageable storage.
	REGION	Assigns real storage space in 4K multiples and virtual storage space in 64K multiples; reassigns MVT hierarchy 1 storage.
	ROLL	Ignores this parameter.
DD	DCB	Specifies BTAM online terminal test option in EROPT subparameter; ingores HIARCHY subparameter.
	OUTLIM	Ignores this parameter.
	SEP	Ignores unit separation for nonspecific devices.
	UNIT	Ignores unit separation for nonspecific devices as specified in SEP subparameter.

Figure 21. VS2 Changes to Job Control Language

JOB Statement

Except for the changes noted below, the JOB statement for VS2 is the same as the JOB statement for MVT.

ADDRSPC Parameter

A new parameter, ADDRSPC, is added to the JOB statement in VS2. It may be specified as either ADDRSPC=VIRT or ADDRSPC=REAL:

- If ADDRSPC=VIRT is specified, the job is assigned a region in pageable storage.
- If ADDRSPC=REAL is specified, the job is assigned a region in nonpageable storage.

If the ADDRSPC parameter is not coded, the default value is VIRT. Therefore, if the current job control language for a job is not changed, the job will be executed in pageable storage. If the job must be executed in nonpageable storage, ADDRSPC=REAL must be specified.

If the ADDRSPC parameter is coded on the JOB statement, its value overrides any value in the ADDRSPC parameters that may be specified in the EXEC statements for that job.

REGION Parameter

In MVT, the size requested in the REGION parameter of the JOB statement was rounded up to the next 2K multiple. In VS2:

- If ADDRSPC=REAL is specified, the requested region size is rounded up to the next 4K (page) multiple.
- If ADDRSPC=VIRT is specified or if the ADDRSPC parameter is not coded, the requested region size is rounded up to the next 64K (segment) multiple.

In VS2, hierarchy 1 storage does not exist. Therefore, if the current job control language for a job is not changed, the hierarchy 1 storage specified is added to the amount of hierarchy 0 storage specified. Hence, the job will get the amount of storage requested, but it will all be in contiguous virtual storage.

Since VS2 subpools are allocated in 4K blocks (vs 2K blocks in MVT), it may be necessary to increase the region size to accomodate this difference.

ROLL Parameter

In VS2, rollout/rollin is not supported since paging gives real storage to tasks as they need it. Therefore, if the current job control language for a job is not changed, the ROLL parameter of the JOB statement will be ignored. That is, the ROLL parameter will be treated as if ROLL=(NO,NO) had been specified.

If the current job control language for a job specifies ROLL=(,YES), it may be necessary to increase the value in the REGION parameter to account for the larger addressing space needed.

EXEC Statement

Except for the changes noted below, the EXEC statement for VS2 is the same as the EXEC statement for MVT.

ADDRSPC Parameter

A new parameter, ADDRSPC, is added to the EXEC statement in VS2. Its function is the same as that described under the ADDRSPC parameter of the JOB statement.

If the ADDRSPC parameter is coded on the EXEC statement, its value applies only to the job step specified on that EXEC statement. However, if the ADDRSPC parameter was coded on the JOB statement, its value on the JOB statement overrides any values specified in the EXEC statements for that job.

REGION Parameter

Changes to the REGION parameter of the EXEC statement for VS2 are the same as those described under the REGION parameter of the JOB statement.

ROLL Parameter

Changes to the ROLL parameter of the EXEC statement for VS2 are the same as those described under the ROLL parameter of the JOB statement.

DD Statement

Except for the changes noted below, the DD statement for VS2 is the same as the DD statement for MVT.

DCB Parameter

In VS2, a new value can be specified for the EROPT subparameter of the DCB parameter of the DD statement. If EROPT=T is coded, it indicates a request for the basic telecommunication access method (BTAM) online terminal test option.

VS2 does not support MVT main storage hierarchy support. Therefore, if the current job control language for a job is not changed, the HIARCHY subparameter of the DCB parameter will be ignored.

OUTLIM Parameter

In VS2, the function provided by the OUTLIM parameter of the DD statement is not supported. Therefore, if the current job control language for a job is not changed, the OUTLIM parameter will be ignored.

SEP Parameter

The SEP parameter of the DD statement is used to request channel separation. In VS2, if a nonspecific volume is requested for a data set and channel separation is also requested for that data set, the channel separation will be ignored; space allocation will be automatically controlled by the I/O load balancing facility. Therefore, in this situation, if the current job control language for a job is not changed, the SEP parameter will be ignored.

UNIT Parameter

Changes to the SEP subparameter of the UNIT parameter of the DD statement for VS2 are the same as those described under the SEP parameter of the DD statement.

Problem Programs

VS2 was designed to be an extension of MVT. Thus, all problem programs (MFT, MVT, and VS1) that execute under Release 21 of MVT, except as noted below, also execute, with no modifications, under VS2. The following problem programs do not have to be modified to execute in VS2:

- Problem programs that are not system or environment dependent.
- Problem programs that are coded according to the guidelines described in the OS System Reference Library publications.
- Problem programs that do not modify the system.
- Problem programs that do not access control block fields not available through normal system functions (that is, through EXTRACT or DEVTYPE macro instructions).

Basic and Extended Control Mode PSWs

Each interruption has associated with it a program status word (PSW). System/370 has two PSW formats, differentiated by bit 12, the former USASCII bit:

- When the bit is off (=0), the format is of basic control (BC) mode in which the features of a System/360 computing system and additional System/370 features, such as new machine instructions, are operational on a System/370 computing system. BC mode is used in VS2 at IPL time.
- When the bit is on (=1), the format is of extended control (EC) mode in which all the features of a System/370 computing system, including dynamic address translation, are operational.

Because of the two PSW formats, the following problem programs that execute under the BC mode PSW may have to be modified to execute under the EC mode PSW:

- Problem programs that reference the following PSW fields directly:

  system mask
  bit 12
  interruption code
  instruction length code (ILC)
  condition code (CC)
  program mask
- Programs that use the load PSW (LPSW) macro instruction to enable or disable the system. The LPSW macro instructions should be replaced with MODESET macro instructions; otherwise, system integrity may be impaired or addressability may be inaccurate.
- Programs that use absolute addresses to refer to locations in low storage above 127 decimal, since the system reserved area in low storage has been expanded.
- Programs that use the set storage key (SSK) or insert storage key (ISK) macro instructions, since the register containing the storage key now contains additional information.

Execute Channel Program (EXCP) Macro Instruction

The execute channel program (EXCP) macro instruction provides a device-dependent means of performing I/O operations. The user of EXCP must provide control information regarding the channel program to be executed; he has the option of specifying the use of I/O appendages during the progress of I/O operations associated with his data set.

Because of virtual storage, the following problem programs that contain EXCP macro instructions may have to be modified to execute in pageable storage:

- Problem programs that modify channel programs while they are executing. Since the I/O supervisor builds a translated copy of the channel program, any changes made to the original channel program would not be known to the I/O supervisor. If the problem programs are not modified, they must be executed in nonpageable storage.
- Problem program appendages. In order to avoid a disabled page fault, the appendages must ensure that referenced pages are fixed in storage before they are entered; this can be accomplished by using the new page fix appendage. If the appendages are not modified, they should be executed in nonpageable storage.

Coding Guidelines

Many coding techniques that are used in MVT can also be used in VS2. Because of virtual storage, however, VS2 necessitates new techniques that should be considered. Although the new techniques are not necessary in order for the programs to run, they may improve system performance.

Although paging occurs frequently in the system, a key objective in new coding techniques is to reduce paging whenever possible. The following guidelines may help achieve this goal:

- 4K-byte blocks (pages) should be coded. Frequently-used programs or loops within programs should be kept on one page whenever possible.
- Seldom used routines (such as initialization, error, and termination routines) should be kept separate from those which are frequently used.
- Frequently-used subroutines should be coded inline each time they are needed. In this case, any increase in total program size will be offset by the decrease in paging.
- Data areas should be arranged on a single page. Multiple-page lists, chains, and tables should be avoided.
- Input/output data areas and event control blocks (ECBs) should be kept on a single page. This prevents referring to several pages when searching for an ECB.
- Pages for buffers should be released as soon as they are all used.
- Dynamically changeable code should be kept on one page if possible. If a page has been changed, it must be paged out before the real storage it occupies can be reused; otherwise, the contents of the page can just be overlaid.

Emulators

Integrated emulator programs allow object programs written for another system to be executed in VS2 with little or no reprogramming. Utilizing the compatibility feature (consisting of hardware and microprogrammed routines that aid emulation), the emulator programs operate as problem program in VS2.

The emulators allows multiprogramming of emulator jobs with other jobs. However, since the emulators employ both programming and circuitry to imitate the processing of other systems, their availability is model-dependent. The emulators supported by the different models are listed in Figure 22.

Information on restrictions on use of the emulators and dates on availability are available from the local IBM branch office.

Emulator	Model 145	Models 155II and 158	Models 165II and 168
1401/1440/1460	x	x	
1410/7010	x	x	
DOS	x	x	
7070/7074		x	x
7080			x
709/7090/7094/7094-II			x

Figure 22. Emulators Supported in VS2

Reassembly/Recompilation

Essentially all programs executing under MVT can execute under VS2 without reassembly or recompilation. The primary exception is that users of the telecommunications access method (TCAM) must reassemble and linkage edit their message control programs before execution.

Also, PL/I F programs executing under the queued telecommunications access method (QTAM) prior to Release 20 of MVT must be reassembled and linkage edited before execution in VS2.

System Data Sets

Some system data sets are basic to VS2 and are required for every operating system. Other system data sets are optional and are required only when selected program options are included in the system.

Most system data sets in VS2 are the same as their counterparts in MVT. However, the SYS1.LPALIB data set, new in VS2, now contains all of the SVCs and most of the modules that were formerly contained in the SYS1.LINKLIB and SYS1.SVCLIB data sets; all the modules contained in this new data set appear in the fixed or pageable link pack areas in VS2.

VS2 also provides two other new data sets: SYS1.PAGE and SYS1.DSSVM. The SYS1.PAGE data set is used to satisfy demand paging requirements; the SYS1.DSSVM data set is used to contain the DSS command language modules.

Required Libraries and Data Sets

The following libraries and data sets are required for every operating system, and must be on the system residence volume.

- SYSCTLG (system catalog) -- The system catalog contains pointers to all the cataloged data sets in an operating system.
- SYS1.LOGREC -- This data set is used by recovery management to record statistical data about machine errors.
- SYS1.NUCLEUS (nucleus library) -- The nucleus library contains the resident portion (nucleus) of the control program and modules selected and link edited during system generation.
- SYS1.SVCLIB (SVC library) -- The SVC library contains recovery management support (RMS) modules required for system error recovery.
- PASSWORD -- This data set contains records that associate the names of protected data sets with the passwords assigned to the data sets.

The following libraries and data sets are required for every operating system, and must be on direct access volumes. They may be on the system residence volume.

- SYS1.DSSVM -- This data set contains the command language modules used by the dynamic support system (DSS).
- SYS1.IMAGELIB -- This library contains the 1403 and 3211 universal character set (UCS) and the forms control buffer (FCB) image modules.
- SYS1.LINKLIB (link library) -- The link library contains programs and routines that are not in the link pack area and are referred to by ATTACH, LINK, LOAD, or XCTL macro instructions.
- SYS1.LPALIB -- This library contains all modules appearing in the link pack area (LPA). Since all modules residing in this library are brought into the LPA at IPL time, this library may be placed on a demountable volume that may be removed after IPL.
- SYS1.MAN (SMF data set) -- The SMF data set contains the data collected by the system management facilities (SMF) routines. A primary data set (SYS1.MANX) and an alternate data set (SYS1.MANY) are required.
- SYS1.PAGE -- This data set is used to satisfy demand paging requirements. It is also used for block paging of TSO users.
- SYS1.PARMLIB -- This library contains the PRESRES list used by the master scheduler, the SMFDEFLT list used by the SMF routines, the BLDL list, and various parameters used by NIP.
- SYS1.PROCLIB (procedure library) -- The procedure library contains cataloged procedures used to perform specific system functions.
- SYS1.SAMPLIB -- This library contains the independent utility programs, the IPL text, the SMF exit routines, and the installation verification procedures needed to verify the operation of the generated system control program.
- SYS1.SYSJOBQE -- This data set is used as a work area by the job scheduler.

The following data sets are required if the time sharing option (TSO) is included in the system, and must be on direct access volumes.

- SYS1.BRODCAST -- This data set contains two types of TSO messages: notices and mail.
- SYS1.CMDLIB -- This library contains TSO command processors, service routines, and utilities.
- SYS1.HELP -- This library is required if the TSO HELP command is used. It contains information regarding the syntax, operands, and functions for each TSO command.
- SYS1.UADS -- This library contains a list of terminal users who are authorized to use TSO. It also contains information about each of them.

Optional Libraries and Data Sets

The following libraries and data sets are optional and, if selected, must be on direct access volumes. They may be on the system residence volume.

- SYS1.DUMP -- This data set is used to contain a dump recorded in the event of abnormal termination of a critical task. This data set may reside on a tape device.
- SYS1.MACLIB (macro library) -- The macro library contains macro definitions for the system macro instructions.
- SYS1.SYSVLOGX and SYS1.SYSVLOGY (system log data sets) -- The system log data sets are used to contain write-to-log (WTL) messages before they are printed on a system output device.
- SYS1.TELCMLIB (telecommunications library) -- The telecommunications library contains telecommunication subroutines in load module form.

The following data sets are optional if the telecommunications access method (TCAM) is included in the system, and, if selected, must be on direct access devices. They may be on the system residence volume.

- TCAM checkpoint data set -- This data set contains all information needed to reconstruct the message control program environment upon restart.
- TCAM message queues data set -- This data set contains the messages between the central computer and the remote stations.

Shared Data Sets

If the shared DASD option is included in the system, systems can share common data. Although any of the installation's application data sets can be shared, not all system data can be shared. Following is a list of those system data sets that cannot be shared.

- PASSWORD
- SYS1.LOGREC
- SYS1.LPALIB
- SYS1.MANX
- SYS1.MANY
- SYS1.NUCLEUS
- SYS1.PAGE
- SYS1.SVCLIB
- SYS1.SYSJOBQE
- SYS1.SYSVLOGX
- SYS1.SYSVLOGY
- SYSCTLG

System Macro Instructions

Several new macro instructions have been added to VS2 to support the new functions available. Other macro instructions have been changed from MVT. Although this section does not provide a detailed description of the new and changed parameters in the macro instructions, it does provide a brief overview of the functions and the uses of the macro instructions.

New VS2 Macro Instructions

The following macro instructions are new and are restricted to authorized programs, programs executing in the supervisor state, and programs operating under protection key zero. The PGFIX, PGFREE, PGLOAD, and MODESET macro instructions are described in the chapter "Supervisor Macro Instructions for System Programmers".

PGFIX -- This macro instruction prevents virtual storage from being paged. It can be used to support I/O operations and to avoid integrity problems that may result from disabled page faults.

PGFREE -- This macro instruction frees virtual storage to be paged. It is used to cancel the effect of the PGFIX macro instruction.

PGLOAD -- This macro instruction moves virtual storage pages into real storage page frames. It can be used to ensure that a page will be in real storage when needed.

MODESET -- This macro instruction is used to change the status of programs between supervisor state and problem program state, key zero and non-key zero, and enabled and disabled.

The following macro instruction are new and are not restricted in use to specific programs:

DEBCHK -- This macro instruction supports the data extent block (DEB) validity checking facility. It can be used to verify that a DEB is valid. The DEBCHK macro instruction is described in *OS/VS Data Management for System Programmers,* GC28-0631.

PGRLSE -- This macro instruction deallocates page frames and the slots allocated to the virtual storage pages. It is used to free up real storage and the external page storage associated with it. The PGRLSE macro instruction is described in *OS/VS Supervisor Services and Macro Instructions,* GC27-6979.

TESTAUTH -- This macro instruction supports the authorized program facility (APF). It is used to determine if a program is authorized to use a program or function restricted in use by APF. The TESTAUTH macro instruction is described in the chapter "Supervisor Macro Instructions for System Programmers".

Changed MVT Macro Instructions

The following macro instructions exist in MVT, but have been changed in VS2.

ENQ/DEQ -- These macro instructions now support an ECB option.

SPIE -- This macro instruction now supports program interruptions that result from page translation exceptions.

Major VS2 - MVT Differences

This section describes the major functional differences and incompatibilities that exist between VS2 and MVT. However, it does not include those areas (for example, job control language or operator commands) previously mentioned in this chapter.

Unsupported MVT Functions

Some of the major functions in MVT are not supported in VS2. Some of the functions have been replaced by improved functions in VS2; others have been eliminated and have no comparable replacements.

The following MVT functions are not supported in VS2, but are provided for by comparable VS2 functions.

- Conversational remote job entry (CRJE) -- This function is provided for by the time sharing option (TSO).
- IEBUPDAT utility program -- This function is provided for by the IEBUPDTE utility program.
- IEHIOSUP utility program -- This function is no longer needed because of the deletion of most modules from the SVC library.
- Queued telecommunications access method (QTAM) -- This function is provided for by the telecommunications access method (TCAM).
- Rollout/rollin -- This function is no longer needed because demand paging gives real storage to tasks as they need it.
- Scatter load -- This function is no longer needed because of the VS2 paging function.
- System environment recording routines (SER0 and SER1) -- This function is provided for by recovery management routines.
- Transient areas -- This function is no longer needed because all SVC routines reside in either the fixed or pageable link pack area.

The following major MVT functions are not supported in VS2, and have no comparable VS2 functions.

- Automatic SYSIN batching (ASB) reader.
- Direct system output (DSO) writer.
- Graphic job processor.
- Main storage hierarchy support.
- Multiprocessing.
- Remote job entry.
- Satellite graphic job processor (SGJP).
- TESTRAN.

VS2 - MVT Differences

Following are other differences that exist between VS2 and MVT.

- In VS2, lists of system parameters may be preformatted for use during system initialization. For more detailed information, see the following chapter "Defining the System".
- In VS2, part of MVT NIP has been replaced by a new system generation option, DEVSTAT.
- IN VS2, the authorized program facility (APF), DEB validity checking, and LSQA provide increased system integrity.
- In VS2, chained scheduling is supported only in nonpageable storage. If chained scheduling is requested in pageable storage, the request is ignored and normal scheduling is substituted.
- The VS2 SMF data set is not supported on tape. Also, SMF records in VS2 contain additional information.
- In VS2, priority queuing must be specified for all devices that contain the page data set.
- VS2 subpools will be allocated in 4K (page) blocks vs 2K blocks in MVT. VS2 also uses a "best fit" algorithm (i.e., storage closest in size to the request) vs a first fit algorithm (i.e., storage that is large enough to accomodate the request) in MVT.
- VS2 has no maximum channel program length.
- In VS2, storage protection implies both store and fetch protection.
- In VS2, all modules residing in the SYS1.LPALIB data set are brought into the link pack area at IPL time. If a BLDL macro instruction is issued for any of the modules, a return code indicating 'not found' is returned since the SYS1.LPALIB data set is no longer considered part of the system after IPL.
- Any modification (via AMASPZAP or the linkage editor) to modules that were made resident in virtual storage at IPL will require a subsequent re-IPL to cause the modification to become part of the system. This would include any module selected at system initialization via FIX and MLPA options as well as all modules from SYS1.LPALIB.

  In addition, if a module contained in SYS1.LPALIB is modified as above, at re-IPL either the pageable LPA must be rebuilt or the modified LPA must be used to cause the modification to become part of the system.
- In VS2, an OPEN macro instruction must be issued for every data extent block (DEB) created in order to support DEB validity checking.
- VS2 does not support the 6-hour and 24-hour pseudo-clocks, and interruptions from location 80; these functions are replaced by the time-of-day clock and the clock comparator.
- In VS2, the LPA directory cannot be modified after system initialization.
- VS2 allows the operator to cancel a job waiting for storage or a data set.

- In VS2, SVC DUMP has been expanded to alow dumping of selected area of virtual storage and selective setting of the system to non-dispatchable.
- In VS2, dynamic device reconfiguration for the system residence device and the page data set is not supported.
- Only Linkage editor F is distributed with VS2. In the job control language, only the aliases IEWL or LINKEDIT can be used; other names can be used only if the linkage editor is re-linkage edited with the other names declared as aliases.
- User-written EDIT exit routines for GTF running under MVT must be modified for operation in VS2 because of differences in the format of trace data for system events.
- VS2 assembler replaces both assembler E and assembler F. If the user has any macro instructions that have the same names as any new commands or instructions of the VS2 assembler, the macro instructions will be interpreted erroneously as the assembler commands or instructions.
- To ensure that direct access volume serial number verification is effective for the volume containing the page data set, the user must ensure that the direct access volume verification modules are always part of the fixed LPA, and that a fixed LPA is specified at each IPL. The necessary modules to accomplish this are provided in the IBM-supplied list IEAFIX00.
- In MVT, the user could issue a READ CCW and rely on the SLI bit to terminate data transfer. In VS2, the user must ensure that the buffer area assigned within his region is large enough to contain the full count specified in the CCW; if not, the request will be terminated. (Note that the assigned buffer space must be allocated space within the region -- for example, via GETMAIN.)
- In MVT and VS2, a user data set can be shared by tasks in the same family tree structure. In VS2, however, the user must ensure that all I/O is completed before the issuing task terminates. For a data set that is not owned by the terminating task (that is, was not opened by the terminating task), the terminating task will be abnormally terminated. For tasks with non-teleprocessing I/O outstanding, ABEND code E06 describes a mechanism to complete the I/O. Tasks with outstanding teleprocessing I/O must update their programs to ensure that the I/O is complete.

Major VS2 - VS1 Differences

This section does not attempt to give a functional description of VS1. Rather, it serves to list the functional differences and incompatibilities that exist between VS1 and VS2. For a complete description of VS1, see *OS/VS1 Planning and Use Guide,* GC24-0509.

- VS1 Release 1 is an extension of MFT Release 20.1; VS2 Release 1 is an extension of MVT Release 21.
- VS1 allows the user to specify up to 16,777,216 bytes of address space; VS2 always provides 16,777,216 bytes of address space.
- VS1 divides storage into pages of 2K bytes each; VS2 divides storage into pages of 4K bytes each.
- VS1 allocates external page storage on as many as 8 devices; VS2 allocates external page storage on as many as 16 devices.
- VS1 supports job processing through the job entry subsystem (JES) facility; VS2 supports job processing via MVT job management.
- VS1 reader procedure specifies PGM=IEFVMA; VS2 reader procedure specifies PGM=IEFIRC.
- VS1 supports the direct system output (DSO) writer; VS2 does not.
- VS1 uses transient areas; VS2 uses the link pack area.

- VS1 supports the function provided by the OUTLIM parameter of the DD statement; VS2 does not.
- VS1 service aid HMDPRDMP formats and prints VS1 dumps only; VS2 service aid AMDPRDMP formats and prints VS2 dumps only.
- VS1 service aid HMDSADMP reads records of 2K bytes each; VS2 service aid AMDSADMP reads records of 4K bytes each.
- VS1 does not support the time sharing option (TSO); VS2 does.
- VS1 does not support dynamic dispatching; VS2 does.
- VS1 does not support I/O load balancing; VS2 does.

Unsupported Devices

Figure 23 describes the VS1 support of those System/370 MVT hardware devices not supported in VS2.

For a list of all the devices that are supported in VS2, see Input/Output Devices in the chapter "Introduction".

Device	Supported in VS1
IBM 1017 Paper Tape Reader	
IBM 1018 Paper Tape Punch	
IBM 1255 Magnetic Character Reader	
IBM 1259 Magnetic Character Reader	
IBM 1270 Optical Reader Sorter	
IBM 1442 Model N1, Card Read Punch	x
IBM 1442 Model N2, Card Punch	x
IBM 2245 Printer	
IBM 2301 Drum Storage	
IBM 2303 Drum Storage	
IBM 2311 Disk Storage Drive	
IBM 2321 Data Cell Drive	
IBM 2402 Magnetic Tape Unit	
IBM 2403 Magnetic Tape Unit and Control	
IBM 2404 Magnetic Tape Unit and Control	
IBM 2415 Magnetic Tape Unit and Control	x
IBM 2596 Card Read Punch	x
IBM 2841 Storage Control Unit	
IBM 3881 Optical Mark Reader	
IBM System/370 Model 135	x

Figure 23. VS1 Support of MVT Devices Not Supported in VS2

Defining the System

During system generation and system initialization, the user is able to define or change the operation of a standard or optional feature of the system control program. By defining the system to meet the needs of the installation, he can increase the performance and overall efficiency of the system. This chapter provides preliminary planning information on defining the system, and is divided into the following categories:

- System generation.
- System initialization.
- System restart.
- System libraries.
- PRESRES volume characteristics list.

System Generation

System generation is the process of selecting VS2 modules from IBM-distributed libraries and tailoring them to create an operating system for an installation. There are five major steps in generating a new operating system:

- Planning -- The distribution libraries containing the VS2 modules must be ordered from IBM. The VS2 options must be selected that, with the standard VS2 features, will constitute the desired system. If an existing VS2 operating system is not available to generate the new system, the VS2 starter system must also be ordered.
- Stage I -- The macro instructions needed to specify the options and standard features of the new system must be selected and coded. These macro instructions are assembled and expanded into job control language and utility control statements.
- Initialization -- The direct access volumes that will contain the distribution libraries (and the starter system) must be initialized. The volumes on which the new system will be generated must also be initialized, and data sets on the system volumes must be allocated.
- Stage II -- The job control language and utility control statements from stage I assemble, link-edit, and specify modules to be moved or copied from the distribution libraries into the new operating system.
- Testing -- After the new system is generated, the IBM Program Systems Representative verifies via the installation verification procedures that the system is operational.

System generation is a process that results in an operating system adapted to both the machine configuration and the data processing needs of an installation. After installation needs have been determined, the "tailored" operating system to meet those needs is specified through system generation macro instructions.

The first time VS2 is installed, the IBM-supplied VS2 starter system is required. (The starter system consists of a VS2 operating system that is used to generate VS2.) For subsequent installations, either the starter system or an existing VS2 operating system can be used.

The IBM Program Systems Representative will perform the installation verification procedures (IVP) to ensure that the system is operational on the hardware configuration. This procedure will be performed as part of IBM's system control program installation procedure for the initial installation of VS2 and will also be performed for subsequent updates.

Two other types of system generation can also be performed: nucleus only and I/O device only. The nucleus generation is performed when only changes to the nucleus of the control program need to be made. The I/O device generation is performed when only I/O devices and channels are to be added, deleted, or modified. (Note: The processor-only generation available to users of MVT is not available in VS2.)

For a description of the system generation process, see *OS/VS System Generation Introduction,* GC26-3790.

In VS2, system generation is improved in the following ways:

- Many facilities that were optional in MVT are now standard. This results in fewer macro instructions and parameters to be specified.
- IBM generates (from the user's system generation statements) and installs one system control program per CPU. IBM installation also includes verification procedures to check that the installed system functions properly. (In MVT, these functions were performed by the user.)
- Additional system initialization lists are available to be preformatted and taken from the SYS1.PARMLIB data set for use at IPL time.

Macro Instructions

System generation macro instructions are used to describe the new system. The macro instructions describe the machine configuration, the control program, the data management routines, and the user-written routines. They are also used to specify the program options that are to be included in the system.

The macro instructions that can be specified in VS2 are:

- CENPROCS -- specifies the central processing unit and secondary model support.
- CHANNEL -- specifies the channel characteristics. A CHANNEL macro instruction is required for each channel in the installation's computing system.
- CKPTREST -- specifies standard system completion codes not eligible for automatic restart, and user completion codes that are eligible for automatic restart. If the CKPTREST macro instruction is not specified, the standard set of codes will be used.
- CTRLPROG -- specifies control program options. If the CTRLPROG macro instruction is not specified, the default values are assumed.
- DATAMGT -- specifies optional access methods (BTAM, BISAM, QSIAM, and TCAM) to be included in the system. (GAM is specified via the GRAPHICS macro instruction.)
- DATASET -- specifies system data sets and the volumes on which they reside. A DATASET macro instruction is required for each data set defined that is not located on the system residence device.
- EDIT -- specifies the physical characteristics and processing attributes of the data sets to be processed by the TSO EDIT command.
- EDITOR -- specifies linkage editor options. If the EDITOR macro instruction is not specified, the default values are assumed.
- GENERATE -- specifies the data sets, volumes, and I/O devices required for the system generation process, the system generation output options, and the type of generation being performed.
- GRAPHICS -- specifies inclusion of graphic programming services (problem oriented routines and graphic subroutine package) and the GAM access method.
- IODEVICE -- specifies characteristics of an I/O device and its operating system requirements. An IODEVICE macro instruction is required for each uniquely addressable I/O device.
- LINKLIB -- specifies user-written routines, in load-module form, to be added to the link library or the fixed link pack area of the new system.
- LOADER -- specifies options to be included in the loader processing program. If the LOADER macro instruction is not specified, the default values are assumed.

- LPALIB -- specifies user-written routines, in load-module form, to be added to the link pack area.
- MACLIB -- specifies functional macro instructions to be excluded from the macro library of the new system.
- PAGE -- specifies the page data set(s). The PAGE macro instruction may be used to define up to 16 page data sets by being specified once for each data set required. The specifications in the PAGE macro instruction may be overridden at NIP time.
- RESMODS -- specifies user-written routines, in object or load-module form, to be added to the system nucleus member being generated.
- SCHEDULR -- specifies job and master scheduler options.
- SECONSLE -- specifies secondary consoles for the multiple console support function.
- SVCTABLE -- specifies the number, type, APF authorization, and entry status of the user written SVC routines to be added to the new system.
- TSO -- specifies a system with the time sharing option.
- UCS -- specifies the IBM standard character set images for an IBM 1403 Printer with the universal character set feature or an IBM 3211 Printer. If the UCS macro instruction is not specified, it is expected that the character set images will be included by a process other than system generation.
- UNITNAME -- specifies a group of I/O devices. A UNITNAME macro instruction is required for each named group of I/O devices in the system, except for unique device types.

For a detailed description of the macro instructions, see *OS/VS2 System Generation Reference,* GC26-3792.

Options

System control program options are included in the system via the system generation macro instructions. Figure 24 lists the options and the macro instructions by which they are specified. For a description of each individual option, see the chapter "Options".

Option	Macro Instruction
Alternate path retry (APR)	IODEVICE
Automatic priority group (APG)	CTRLPROG
Automatic volume recognition (AVR)	SCHEDULR
Basic indexed sequential access method (BISAM)	DATAMGT
Basic telecommunications access method (BTAM)	DATAMGT
Device independent display operator console support (DIDOCS)/status display support (SDS)	SCHEDULR
Graphics access method (GAM)	GRAPHICS
Queued indexed sequential access method (QISAM)	DATAMGT
Reliability data extractor (RDE)	CTRLPROG
Shared direct access storage devices (shared DASD)	IODEVICE
Telecommunications access method (TCAM)	DATAMGT
Time sharing option (TSO)	TSO
Time slicing	CTRLPROG
Track stacking	SCHEDULR

Figure 24. VS2 Options and System Generation Macro Instructions

Planning Considerations

The primary step in generating a new system to satisfy the needs of an installation is to determine exactly what those needs are. Figure 25 lists planning questions that must be considered before system generation. The questions are divided into the following categories:

- Machine configuration.
- Control program.
- Data management routines.
- User-written routines.
- New system data set allocation and cataloging.
- Generation.

The first column in the figure lists the questions to be answered before system generation. The second and third columns indicate the system generation macro instructions and parameters in which the answers to the questions are specified. The last column points either to the publications that contain the answers to the questions or to the publications that contain the most information about the indicated subjects.

For a detailed description of the macro instructions and the parameters, see *OS/VS2 System Generation Reference,* GC26-3792. For information on those questions that do not have specific publication references, also see *OS/VS2 System Generation Reference.*

Question to be Answered Before System Generation	Sys Gen Macro	Sys Gen Parameter	Publication Containing Info Related to Question
Machine Configuration			
What CPU is to be used ?	CENPROCS	MODEL	
What additional CPUs will require EREP support ?	CENPROCS	SECMODS	OS/VS SYS1.LOGREC Error Recording
How many channels and of what type are to be used ?	CHANNEL	TYPE	
What addresses are to be specified for the channels ?	CHANNEL	ADDRESS	
What I/O devices are to be used to support the installation ?	IODEVICE	UNIT	
What addresses are to be specified for the I/O devices ?	IODEVICE	ADDRESS	
What nonstandard error routines are to be used for the I/O devices ?	IODEVICE	ERRTAB	
How much buffer space is to be allowed for 2250-1 programs that use EXPRESS attention handling routines ?	IODEVICE	EXPBFR	
What optional features are included on the devices ?	IODEVICE	FEATURE	
What are the graphic control units to which the 2260-2 devices are attached ?	IODEVICE	GCU	
What types of I/O device queueing are to be provided for the devices ?	IODEVICE	IOREQUE	
What are the model numbers of the devices to be used ?	IODEVICE	MODEL	
How many 256-byte buffers are to be allowed in a 2840 control unit assigned to a 2250-3 device ?	IODEVICE	NUMSECT	
How many work stations are to be connected to a 2715 in a 2790 data communication system ?	IODEVICE	OBRCNT	
Is alternate path retry (APR) to be included in the system ?	IODEVICE	OPTCHAN	
What numbers are to be assigned to 2840 devices to which 2250-3 devices are attached ?	IODEVICE	PCU	
Which set address (SAD) commands are to be issued for each 2702 device ?	IODEVICE	SETADDR	
What teleprocessing control units are to be used for telecommunications lines ?	IODEVICE	TCU	
What adapter is to be used to connect a telecommunications line to a transmission control unit ?	IODEVICE	ADAPTER	
What additional device characteristics are to be specified for non-IBM devices ?	IODEVICE	DEVTYPE	OS/VS2 System Data Areas

Figure 25. VS2 System Planning Considerations (Part 1 of 7)

Question to be Answered Before System Generation	Sys Gen Macro	Sys Gen Parameter	Publication Containing Info Related to Question
Machine Configuration (continued)			
What standard IBM character set images are to be used ?	UCS	IMAGE	OS/VS Data Management for System Programmers
What standard IBM character set images are to be used as defaults when the JCL for a job does not specify any sets ?	UCS	DEFAULT	OS/VS Data Management for System Programmers
What symbolic names are to be specified for groups of I/O devices ?	UNITNAME	NAME	
What addresses are to be specified for I/O devices in a group ?	UNITNAME	UNIT	
Control Program			
Which installation-created ABEND codes are to be eligible for automatic restart ?	CKPTREST	ELIGBLE	OS/VS Checkpoint/Restart
Which IBM ABEND codes are not to be eligible for automatic restart ?	CKPTREST	NOTELIG	OS/VS Checkpoint/Restart
Is the ASCII SVC to be included in the system ?	CTRLPROG	ASCII	
How many simultaneous I/O operations are to be processed in the system ? That is, how many request queue elements are to be included in the nucleus ?	CTRLPROG	MAXIO	OS/VS2 Storage Estimates
Is the BLDL table to be fixed or paged ?	CTRLPROG	BLDL	OS/VS2 Planning and Use Guide
Is the reliability data extractor (RDE) to be included in the system ?	CTRLPROG	RDE	OS/VS SYS1.LOGREC Error Recording OS/VS2 Planning and Use Guide
Is reduced error recovery for magnetic tapes to be included in the system ?	CTRLPROG	RER	OS/VS2 Planning and Use Guide
Is the automatic priority group (APG) option to be included in the system ?	CTRLPROG	APG	OS/VS2 Planning and Use Guide
What is to be the initial size of the DEB table used for DEB validity checking ?	CTRLPROG	DEBTSZE	OS/VS2 Planning and Use Guide
How much is the size of the DEB table to be expanded if the initial size is too small for a job step ?	CTRLPROG	DEBTINC	OS/VS2 Planning and Use Guide OS/VS2 Storage Estimates
What is to be the size and number of cells in the local system queue area ?	CTRLPROG	LSQACEL	OS/VS2 Planning and Use Guide OS/VS2 Storage Estimates
Is NIP to treat DASD and tape devices that are not ready as being offline ?	CTRLPROG	DEVSTAT	OS/VS2 Planning and Use Guide
How much real storage is needed to run jobs in nonpageable dynamic storage ?	CTRLPROG	REAL	OS/VS2 Planning and Use Guide OS/VS2 Storage Estimates

Figure 25. VS2 System Planning Considerations (Part 2 of 7)

Question to be Answered Before System Generation	Sys Gen Macro	Sys Gen Parameter	Publication Containing Info Related to Question
Control Program (continued)			
What is to be the size and number of cells in the system queue area ?	CTRLPROG	SQACEL	OS/VS2 Planning and Use Guide OS/VS2 Storage Estimates
Is the output to reflect local standard time instead of Greenwich Mean Time ?	CTRLPROG	TZ	OS/VS2 Planning and Use Guide
How many 64K segments are to be reserved for system queue area ?	CTRLPROG	QSPACE	OS/VS2 Storage Estimates
Is time slicing to be included in the system ?	CTRLPROG	TMSLICE	OS/VS2 Planning and Use Guide
Is the system trace table to be included in the system ?	CTRLPROG	TRACE	OS/VS2 Planning and Use Guide OS/VS2 Storage Estimates
How much virtual storage is to be available to the linkage editor and its text buffers ?	EDITOR	SIZE	OS/VS Linkage Editor and Loader OS/VS2 Storage Estimates
What library is to be used by the loader to resolve external references ?	LOADER	LIB	OS/VS Linkage Editor and Loader
What is to be the primary input data set for the loader ?	LOADER	LIN	OS/VS Linkage Editor and Loader
What loader options are to be included in the system ?	LOADER	PARM	OS/VS Linkage Editor and Loader
How much virtual storage is to be allowed for the loader, its tables, its buffers, and the problem program ?	LOADER	SIZE	OS/VS Linkage Editor and Loader OS/VS2 Storage Estimates
Which groups of macro definitions are to be excluded from SYS1.MACLIB ? That is, is macro support for BTAM, TCAM, GPS, OCR, and TSO to be included in the system ?	MACLIB	EXCLUDE	OS/VS2 Planning and Use Guide OS/VS2 Storage Estimates
Is the page data set(s) to be included in the primary or secondary device ?	PAGE	TYPE	OS/VS2 Planning and Use Guide
How much space is to be allocated to the page data set(s) ?	PAGE	SIZE	OS/VS2 Storage Estimates
Is the paging device to contain the pageable link pack area ?	PAGE	LPA	OS/VS2 Planning and Use Guide
What is to be the address of the paging device ?	PAGE	UNIT	OS/VS2 Planning and Use Guide
What volume is to contain the page data set ?	PAGE	VOLNO	OS/VS2 Planning and Use Guide

Figure 25. VS2 System Planning Considerations (Part 3 of 7)

Question to be Answered Before System Generation	Sys Gen Macro	Sys Gen Parameter	Publication Containing Info Related to Question
Control Program (continued)			
What is to be the address of the primary console ?	SCHEDULR	CONSOLE	OS/VS2 Planning and Use Guide
Are the volume error statistics to be written to the console or the SMF data sets ?	SCHEDULR	ESV	OS/VS2 Planning and Use Guide
Is error volume analysis to be included in the system ?	SCHEDULR	EVA	OS/VS2 Planning and Use Guide
What is to be the address of the hardcopy log, and how are routing codes to be assigned ?	SCHEDULR	HARDCPY	OS/VS2 Planning and Use Guide OS/VS Message Library: Routing and Descriptor Codes
How many real storage buffers are to hold logical tracks from SYS1.SYSJOBQE ?	SCHEDULR	INITQBF	OS/VS2 Storage Estimates
What is to be the number of records in a logical track in SYS1.SYSJOBQE ?	SCHEDULR	JOBQFMT	OS/VS2 Storage Estimates
How many records are to be reserved in SYS1.SYSJOBQE for each initiator ?	SCHEDULR	JOBQLMT	OS/VS2 Storage Estimates
How many records are to be reserved in SYS1.SYSJOBQE for termination of jobs that need more records for initiation ?	SCHEDULR	JOBQTMT	OS/VS2 Storage Estimates
How many records in SYS1.SYSJOBQE are to be reserved for write-to-programmer routines ?	SCHEDULR	JOBQWTP	OS/VS2 Storage Estimates
What routing codes are to be assigned to messages that do not have routing codes ?	SCHEDULR	OLDWTOR	OS/VS Message Library: Routing and Descriptor Codes
How many reply queue elements are to be used by WTOR routines ?	SCHEDULR	REPLY	OS/VS2 Storage Estimates
What additional routing codes are to be received by the master console ?	SCHEDULR	ROUTCDE	OS/VS Message Library: Routing and Descriptor Codes
Is a START INIT command to be executed automatically after each IPL ?	SCHEDULR	STARTI	Operator's Library: OS/VS2 Reference
Is a START RDR command to be executed automatically after each IPL ?	SCHEDULR	STARTR	Operator's Library: OS/VS2 Reference
Is a START WTR command to be executed automatically after each IPL ?	SCHEDULR	STARTW	Operator's Library: OS/VS2 Reference
What address is to be specified for the integrated operator console to be used with DSS ?	SCHEDULR	IOC	OS/VS2 Planning and Use Guide
Is automatic volume recognition (AVR) to be included in the system ?	SCHEDULR	VLMOUNT	OS/VS2 Planning and Use Guide
What density is to be selected for 7-track tapes to be used with AVR ?	SCHEDULR	TAVR	OS/VS2 Planning and Use Guide
What default SYSOUT class is to be assigned to write-to-log (WTL) messages ?	SCHEDULR	WTLCLSS	OS/VS2 Planning and Use Guide

Figure 25. VS2 System Planning Considerations (Part 4 of 7)

Question to be Answered Before System Generation	Sys Gen Macro	Sys Gen Parameter	Publication Containing Info Related to Question
Control Program (continued)			
How many buffers are to be reserved for WTO routines ?	SCHEDULR	WTOBFRS	OS/VS2 Storage Estimates
What dimensions are to be specified for display areas to be reserved for status displays by the primary console ?	SCHEDULR	AREA	Operator's Library: OS/VS2 Display Consoles
How many records are to be reserved in SYS1.BRODCAST for broadcast messages ?	SCHEDULR	BCLMT	OS/VS2 TSO Guide OS/VS2 Storage Estimates
Is the 2250 or 3277 console, if used, to have a PFK command entry and/or a light pen command entry ?	SCHEDULR	PFK	Operator's Library: OS/VS2 Display Consoles
How many temporary storage areas for WTL messages are to be reserved in SYS1.SYSVLOGX or SYS1.SYSVLOGY ?	SCHEDULR	WTLBFRS	OS/VS2 Storage Estimates
What are to be the addresses of the alternate consoles for the secondary console ?	SECONSLE	ALTCONS	OS/VS2 Planning and Use Guide
What dimensions are to be specified for display areas to be reserved for status displays for the secondary console ?	SECONSLE	AREA	Operator's Library: OS/VS2 Display Consoles
What is to be the address of the secondary console ?	SECONSLE	CONSOLE	OS/VS2 Planning and Use Guide
What are to be the routing codes associated with the secondary console ?	SECONSLE	ROUTCDE	OS/VS Message Library: Routing and Descriptor Codes
Is the 2250 or 3277 console, if used as a secondary console, to have a PFK command entry and/or a light pen command entry ?	SECONSLE	PFK	Operator's Library: OS/VS2 Display Consoles
Is the secondary console to be used as an output-only console for status displays, or as an output-only console for operator messages, or for full capacity ?	SECONSLE	USE	OS/VS2 Planning and Use Guide
Which groups of operator commands are to be entered from the secondary console ?	SECONSLE	VALDCMD	Operator's Library: OS/VS Console Configurations
What are the physical characteristics and processing attributes of the data sets to be handled by the TSO EDIT command ?	EDIT	DSTYPE, BLOCK, FORMAT, FIXED,VAR, CONVERT, USERSRC	OS/VS2 TSO Command Language Reference
What is the installation-supplied processor to be used by TSO EDIT to syntax-check lines in the data set ?	EDIT	CHECKER	OS/VS2 TSO Guide
What is the installation-supplied processor to be used by the RUN subcommand of TSO EDIT ?	EDIT	PRMPTR	OS/VS2 TSO Guide
What is the installation-supplied user exit routine to be used by TSO EDIT to interpret nonstandard data set type operand parameters ?	EDIT	USEREXT	OS/VS2 TSO Guide

Figure 25. VS2 System Planning Considerations (Part 5 of 7)

Question to be Answered Before System Generation	Sys Gen Macro	Sys Gen Parameter	Publication Containing Info Related to Question
Control Program (continued)			
How many lines at a terminal are to be entered before an attempt to LOGON is automatically canceled ?	TSO	LOGLINE	OS/VS2 TSO Guide
How many seconds are to elapse before an attempt to LOGON is automatically canceled ?	TSO	LOGTIME	OS/VS2 TSO Guide
What are to be the class defaults for the OUTPUT command ?	TSO	OUTCLS	OS/VS2 TSO Guide
How many logical tracks in SYS1.SYSJOBQE are to be reserved for background jobs initiated in a TSO foreground ?	TSO	SUBMITQ	OS/VS2 Storage Estimates
How much external page storage is to be made available for use by TSO ?	TSO	TSOAUX	OS/VS2 TSO Guide
Data Management Routines			
Which of the optional access methods (BTAM, TCAM, and ISAM) are to be added to the system ?	DATAMGT	ACSMETH	OS/VS2 Planning and Use Guide OS/VS2 Storage Estimates OS/VS BTAM OS TCAM Concepts and Facilities
Is the graphic subroutine package (GSP) to be added to the system ?	GRAPHICS	GSP	OS/VS Graphic Subroutine Package (GSP) for FORTRAN IV, COBOL, and PL/1
Are problem oriented routines (POR) to be added to the system ?	GRAPHICS	PORRTNS	OS/VS Graphic Programming Services (GPS) for the IBM 2250 Display Unit
User-Written Routines			
What routines are to be added to SYS1.LINKLIB ?	LINKLIB	MEMBERS	OS/VS2 Planning and Use Guide
What partitioned data set contains the routines to be added to SYS1.LINKLIB ?	LINKLIB	PDS	OS/VS2 Planning and Use Guide
Which of the routines being added to SYS1.LINKLIB are to become resident in the link pack area ?	LINKLIB	RESIDNT	OS/VS2 Planning and Use Guide
What routines are to be added to SYS1.LPALIB ?	LPALIB	MEMBERS	OS/VS2 Planning and Use Guide
What partitioned data set contains the routines to be added to SYS1.LPALIB ?	LPALIB	PDS	OS/VS2 Planning and Use Guide
Which of the routines being added to SYS1.LPALIB are to become resident in the link pack area ?	LPALIB	RESIDNT	OS/VS2 Planning and Use Guide
What routines are to be added to SYS1.NUCLEUS ?	RESMODS	MEMBERS	OS/VS2 Planning and Use Guide OS/VS2 Storage Estimates

Figure 25. VS2 System Planning Considerations (Part 6 of 7)

Question to be Answered Before System Generation	Sys Gen Macro	Sys Gen Parameter	Publication Containing Info Related to Question
User-Written Routines (continued)			
What partitioned data set contains the routines to be added to SYS1.NUCLEUS ?	RESMODS	PDS	OS/VS2 Planning and Use Guide
What is the number, type, and function code of each SVC to be added to SYS1.SVCLIB ?	SVCTABLE	operand	OS/VS2 Planning and Use Guide
New System Data Set Allocation and Cataloging			
What system data sets are needed ?	DATASET	system data set	OS/VS2 Planning and Use Guide
What is the device type of the volume containing each system data set ?	DATASET	VOL	OS/VS2 Storage Estimates
What unit of space and how much space should be allocated to each system data set ?	DATASET	SPACE	OS/VS2 Storage Estimates
Generation			
What type of system generation is being performed ?	GENERATE	GENTYPE	OS/VS2 System Generation Reference
What index qualifier should be specified for new system data sets ?	GENERATE	INDEX	OS/VS2 System Generation Reference
What data set is to contain the object modules assembled during stage II of system generation ?	GENERATE	OBJPDS	OS/VS2 System Generation Reference
What job class is to be used for output from stage II of system generation ?	GENERATE	JCLASS	OS/VS2 System Generation Reference
What output class is to be used for output from stage II of system generation ?	GENERATE	OCLASS	OS/VS2 System Generation Reference
What is the device type and serial number of the system residence volume ?	GENERATE	RESVOL	OS/VS2 System Generation Reference

Figure 25. VS2 System Planning Considerations (Part 7 of 7)

System Initialization

Before VS2 can process jobs, the control program and its associated control blocks and work areas must be loaded into virtual storage and prepared for operation. Loading these control program modules is the function of the initial program loader (IPL).

After the IPL program completes it loading functions, control passes to the nucleus initialization program (NIP), which performs functions necessary to initiate operation of the control program. (For example, NIP defines storage areas and initializes certain tables, work areas, and control blocks.) NIP also loads and initializes routines (such as fixed LPA routines) selected by the user.

The nucleus initialization program (NIP) provides initialization of both real and virtual storage in VS2. To give the installation control over virtual storage, NIP recognizes several new parameters. The installation is able to specify these new parameters, as well as the supported MVT parameters, via the master console or a new SYS1.PARMLIB member. The parameters specified are variable with each execution of the initial program loader (IPL) program.

SYS1.PARMLIB

The SYS1.PARMLIB data set in VS2 includes both IBM-supplied and user-supplied parameter lists. The formats for the lists have been revised from MVT to allow better space utilization within parameter records and to afford more flexibility in their definition. However, all MVT parameter lists that are still valid lists in VS2 need not have their formats changed.

IBM-Supplied Lists

The following IBM-supplied lists are currently included in SYS1.PARMLIB for MVT, but are not included in VS2:

- IEAIGE00 -- resident error recovery procedures list.
- IEAIGG00 -- resident access methods list.
- IEARSV00 -- resident SVC list.

The following lists are currently included in SYS1.PARMLIB for MVT, and are also included in VS2. A complete discussion of the lists is found in the chapter "Job Management and Supervisor Services for System Programmers".

- IEABLD00 -- resident BLDL list.
- LNKLST00 -- link library list.

The following lists are new for VS2:

- IEASYS00 -- It is created during stage II of system generation and includes all system parameters defined during system generation. If not modified in response to the "SPECIFY SYSTEM PARAMETERS" message, the IEASYS00 list of parameters is used for the IPL in process.
- IEAPAK00 -- It consists of lists of modules, where the modules in each list are to be packed together in one or more pages. Processing of one module in a list will usually result in another module in the same list receiving control, thus reducing paging activity. During initialization, NIP will load modules into the link pack area in the groupings specified by IEAPAK00. Those modules not included in IEAPAK00 will be loaded into the link pack area in the most efficient manner, based upon size. If desired, additional user-specified lists may be added to IEAPAK00 via the IEBUPDTE utility program.
- IEALOD00 -- It is used to place names of frequently-used modules from the pageable link pack area onto the active LPA queue. The list is supplied to improve performance by speeding up the search mechanism that finds the control blocks for the modules, thus reducing paging.

- IEAFIX00 -- It is used to indicate reenterable routines from the SYS1.LINKLIB, SYS1.LPALIB, and SYS1.SVCLIB data sets that are to be made resident as a nonpageable extension to the link pack area. The routines in this area (called the fixed link pack area) are resident only for the current IPL. If a routine is searched for, the fixed link pack area is searched first, the modified link pack area is searched second, and the pageable link pack area is searched last. If a system restart is initiated and a previously created link pack area is used, the routines in the fixed link pack area must be respecified if desired.

User-Supplied Lists

The SYS1.PARMLIB lists described below are not supplied by IBM, but are defined by the VS2 user via the IEBUPDTE utility program. They are all new for VS2.

- IEASYSxx -- The installation is able to specify system parameters through the IEASYSxx list in SYS1.PARMLIB; in fact, multiple sets of system parameters can be maintained in SYS1.PARMLIB. The set of system parameters which is to be effective for any given IPL process is determined by the operator's response to the "SPECIFY SYSTEM PARAMETERS" message. The operator's reply may include a list of parameters to be used or one of the SYS1.PARMLIB parameter lists. If the SYS1.PARMLIB list being used does not prohibit operator intervention, the operator can override all parameters. By using a SYS1.PARMLIB parameter list, the operator is freed from typing in all the necessary entries. The IEASYSxx list is specified by the SYSP system parameter.
- IEALPAxx -- The IEALPAxx list is used to indicate reenterable routines from the SYS1.LINKLIB, SYS1.LPALIB, and SYS1.SVCLIB data sets that are to be made resident as a pageable extension to the link pack area. The routine in this area (called the modified link pack area) are resident only for the current IPL. If a routine is searched for, the modified link pack area is searched first, followed by the pageable link pack area. If a system restart is initiated and a previously created link pack area is used, the routines in the modified link pack area must be respecified if desired. The IEALPAxx list is specified by the MLPA system parameter.
- IEAFIXxx -- The IEAFIXxx list is used to indicate reenterable routines from the SYS1.LINKLIB, SYS1.LPALIB, and SYS1.SVCLIB data sets that are to be made resident as a nonpageable extension to the link pack area. The routines in this area (called the fixed link pack area) are resident only for the current IPL. If a routine is searched for, the fixed link pack area is searched first, the modified link pack area is searched second, and the pageable link pack area is searched last. If a system restart is initiated and a previously created link pack area is used, the routines in the fixed link pack area must be respecified if desired. The IEAFIXxx list is specified by the FIX system parameter.
- IEABLDxx -- The IEABLDxx list specifies the names of modules from SYS1.LINKLIB which are to have directory entries built by NIP. This list eliminates the directory search required when a load module is requested from SYS1.LINKLIB. The IEABLDxx list is specified by either the BLDL or BLDLF system parameter.

VS2 System Parameters

Each parameter in the SYS1.PARMLIB lists is specified in the same format as that required by the "SPECIFY SYSTEM PARAMETERS" message. Figure 26 provides a summary of these parameters and, for convenience, groups them into nine general categories. Following the figure is a more detailed description of the parameters.

For a detailed description and specification of these parameters, see the explanation of the "SPECIFY SYSTEM PARAMETERS" message (IEA101A) in *OS/VS Message Library: VS2 System Messages*, GC38-1002.

Parameter	NIP Categories: Virtual Storage Initialization	Real Storage Initialization	Page Data Set Initialization	Paging Supervisor Initialization	Task Dispatching	System Queue Area Definition	Master Scheduler Region Definition	System Parameter Definition	Other Definitions	Value Specified
APG					x					Automatic priority group for dispatcher.
BLDL	x									Pageable directory for SYS1.LINKLIB.
BLDLF	x	x								Nonpageable directory for SYS1.LINKLIB.
CLPA	x									Link pack area to be recreated.
CPQE				x						Channel programs for paging supervisor.
DUMP									x	Tape volume for SYS1.DUMP.
FIX	x	x								Reenterable routines for nonpageable LPA.
HARDCPY									x	Hard copy log.
LSQACEL						x				Quickcells for local system queue area.
MLPA	x									Reenterable routines for pageable LPA.
MPA							x			Master scheduler region size.
OPI								x		Operator intervention restrictions in SYS1.PARMLIB.
PAGE			x							Location and allocation parameters for page data set.
PAL				x						Paging algorithm limits.
REAL	x	x								Nonpageable storage size.
SQA						x				Pageable system queue area size.
SQACEL						x				Quickcells for system queue area.
SYSP								x		SYS1.PARMLIB parameter list to be merged with IEASY00.
TMSL					x					Time slice groups.
TRACE									x	Entries in system trace table.
TSOAUX			x							Minimum auxiliary storage backup for TSO jobs.

Figure 26. VS2 System Initialization Parameter Summary

APG
specifies an automatic priority group for the dispatcher.

If the APG and TMSL parameters specify the same priority, the APG specification of that priority will be accepted.

BLDL
specifies a directory for the SYS1.LINKLIB data set in pageable storage.

The value specified is appended to IEABLD to form the name of a SYS1.PARMLIB member to contain the list of modules for which entries are to be built in the resident BLDL table.

The directory created in response to the BLDL parameter will reside in pageable storage.

If the BLDL and BLDLF parameters are both specified, the BLDL parameter will be ignored. A warning message will be issued to identify this condition.

If specified during system generation and not modified, the IEABLD00 list is used.

BLDLF
specifies a directory for the SYS1.LINKLIB data set in nonpageable storage.

The value specified is appended to IEABLD to form the name of a SYS1.PARMLIB member to contain the list of modules for which entries are to be built in the resident BLDL table.

The directory created in response to the BLDLF parameter will reside in nonpageable storage.

If the BLDLF and BLDL parameters are both specified, the BLDLF parameter will be accepted. A warning message will be issued to identify this condition.

If specified during system generation and not modified, the IEABLD00 list is used.

CLPA
specifies that the link pack area is to be recreated.

If a previously created link pack area is found in one of the page data sets, it is deleted and the space it occupied is made available for system paging use.

CPQE
specifies the number of channel programs to be used by the paging supervisor.

The value specified is the number of channel programs to be added to the minimum number of channel programs (10 for a single page data set, and 15 for two or more page data sets).

If the CPQE parameter is not specified and the time sharing option is included in the system, the minimum numbers described above will be increased by 80.

DUMP
specifies whether a tape volume is to be used for the SYS1.DUMP data set.

If the SYS1.DUMP data set is cataloged on a direct access device, the DUMP parameter will be ignored.

If the DUMP parameter is not specified and if the SYS1.DUMP data set is not cataloged, a message will be issued requesting that a tape volume be provided and specified for the SYS1.DUMP data set.

FIX
specifies reenterable routines from the SYS1.LINKLIB, SYS1.LPALIB, and SYS1.SVCLIB data sets or from a user-specified library to be made resident as a nonpageable extension to the link pack area.

The value specified is appended to IEAFIX to form the name of a SYS1.PARMLIB member to contain the list of modules to be loaded.

If specified during system generation and not modified, the IEAFIX00 list is used.

For more information on the IEAFIXxx list, see "User-Supplied Lists" in this chapter.

HARDCPY
specifies that the hard copy log is desired, whether the system log is to be used as hardcopy log, and whether messages are to be recorded on the hardcopy log.

If the console configuration contains an active graphic console or more than one active console, a hardcopy log is required. In this case, if routing codes 1, 2, 3, 4, 7, 8, or 10 are desired, they need not be specified since all of these codes are automatically assigned by the system.

LSQACEL
specifies quickcells (reserved space used to reduce time required to allocate space for a control block) in the local system queue area.

The size of a quickcell will be rounded up by the system to an even multiple of 8 bytes. The total size of the quickcell area is limited to 4K bytes.

MLPA
specifies reenterable routines from the SYS1.LINKLIB, SYS1.LPALIB, and SYS1.SVCLIB data sets to be made resident as a pageable extension to the link pack area.

The value specified is appended to IEALPA to the form the name of a SYS1.PARMLIB member to contain the list of modules to be loaded.

For more information on the IEALPAxx list, see "User-Supplied Lists" in this chapter.

MPA
specifies the master scheduler region in pageable storage.

The value specified is the number of 64K byte segments to be added to the minimum size (128K bytes) of the master scheduler region.

If the specified amount of virtual storage is not available, a warning message will be issued. The MPA parameter may be respecified at that time.

OPI
specifies operator intervention restrictions of the system parameters in the SYS1.PARMLIB list.

The OPI parameter can only be included in the SYS1.PARMLIB list. It may be specified for individual system parameters or for the entire set of system parameters.

PAGE
specifies the location and allocation parameters for the page data set(s).

The value specified may include the unit address or the volume serial number of the device containing the page data set, the number of storage blocks to be pageable to the data set, the number of tracks or cylinders to be assigned to the data set, the largest single area currently available on the device assigned to the data set, whether the page data set is to be reformatted, whether the page data set is to contain the link pack area and its directory, and whether the page definition is to be displayed on the console.

PAL
specifies page fix count limits, page replacement threshholds, and intervals and limits for the task disable algorithm.

REAL
specifies nonpageable storage.

The value specified is the number of 4K byte pages to be added to the minimum address space (64K bytes) reserved for nonpageable storage.

If the specified amount of real storage is not available, a warning message will be issued. The REAL parameter may be respecified at that time.

SQA
specifies the system queue area in pageable storage.

The value specified is the number of 64K byte segments to be added to the minimum size (64K bytes) of the system queue area.

If the specified amount of virtual storage is not available, a warning message will be issued. The SQA parameter may be respecified at that time.

The size of the system queue area cannot be increased during a restart that reuses the previously initialized link pack area. If the SQA parameter is specified on the restart, and the size is less than that specified on the previous system start, a warning message will be issued and the size will be increased to that of the previous system start. If the SQA parameter is specified on the restart, and the size is greater than that specified on the previous system start, a message will be issued requesting that the SQA parameter be canceled or a system start be initiated.

SQACEL
specifies quickcells (reserved space used to reduce time required to allocate space for a control block) in the system queue area.

The size of a quickcell will be rounded up by the system to an even multiple of 8 bytes. The total size of the quickcell area is limited to 4K bytes.

SYSP
specifies the SYS1.PARMLIB list of system parameters which is to be merged with the default list of system parameters, IEASYS00.

The value specified is appended to IEASYS to form the name of a SYS1.PARMLIB member to contain the list of system parameters to be used for the current IPL.

If not modified via this parameter, only the IEASYS00 list is used. The IEASYS00 member contains the values selected during system generation for various system parameters unless the contents of the list have been modified by the system programmer at the installation.

The members specified by the SYSP parameter are treated in an ascending priority sequence -- that is, parameters defined in a member specified near the beginning of the list will be replaced by the same parameters defined in a member specified near the end of the list. In the process of merging parameters with the IEASYS00 list, IEASYS00 assumes the lowest priority.

TMSL
specifies time slice groups.

The minimum acceptable time slice is 20 milliseconds and any specification less than this value will be increased to 20 milliseconds.

If the TMSL and APG parameters specify the same priority, the TMSL specification of that priority will be ignored.

TRACE
specifies entries in the system trace table.

The value specified here in the TRACE parameter overrides the value specified during system generation. If zero is specified, the system trace option is canceled for the current IPL. If zero was specified during system generation and overridden here, the trace table will not include entries for SIOs.

If the specified amount of real storage is not available, a warning message will be issued. The TRACE parameter may be respecified at that time.

TSOAUX
specifies minimum auxiliary storage backup for TSO jobs.

The value specified is the pages of auxiliary storage required to back up the pageable area to be reserved for paging TSO-related tasks.

The storage indicated in this parameter will not be available for allocation to non TSO-related tasks for the duration of the IPL; however, the TSOAUX value may be respecified when TSO is started.

If the number of pages remaining for non-TSO use is less than 500, the number of TSO-reserved pages will be decreased to provide the system with 500 pages. If the total number of pages is less than 500, the TSOAUX parameter will be ignored. In both cases, a warning message will be issued to identify the conditions.

Unsupported MVT Parameters

Some of the system parameters in MVT are not supported in VS2. Some of the parameters have been replaced by new parameters in VS2; others have been eliminated because their functions are not supported.

If an unsupported parameter is specified, a message will be issued identifying the condition; all parameters specified after the nonsupported parameter will have to be respecified for the system.

The following MVT system parameters are not supported in VS2, but are provided for by comparable VS2 functions.

- MIN -- This parameter is no longer needed in VS2 since the minimum space is now determined by the scheduler.
- MOD -- The CPU model identification is determined in VS2 by the store CPU identification instruction.
- MPS -- This parameter is replaced by the VS2 MPA parameter.
- RAM -- Routines from SYS1.SVCLIB and SYS1.LINKLIB can be added to the VS2 link pack area via the SYS1.LPALIB data set or via the MLPA or FIX parameters.
- RERP -- In VS2, error recovery procedure (ERP) routines are always resident in the link pack area.
- RESVC -- In VS2, SVC routines are always resident in the link pack area.
- SQS -- This parameter is replaced by the VS2 SQA parameter.

The following MVT system parameters are not supported in VS2, and have no comparable VS2 functions.

- ALTSYS -- Dynamic device reconfiguration for the system residence device is not supported.
- HRAM -- MVT hierarchy support is not needed in VS2.
- HSVC -- MVT hierarchy support is not needed in VS2.
- QBF -- The buffer space specified by this parameter can now be specified only at system generation time.

System Restart

It is sometimes necessary to shut down the system because of end-of-shift, end-of-day, normal maintenance, or system malfunction. At these times, in MVT, system restart allows the system to resume operations without redefining the job queues.

In VS2, as in MVT, system restart also allows reuse of the previously initialized job queue. However, in VS2, another type of system restart is available that allows reuse of the previously-initialized link pack area. Unless the CLPA parameter is specified requesting that the link pack area be recreated or unless the link pack area is reformatted via the PAGE parameter, this second type of restart is performed.

System Libraries

To increase system throughput, libraries should be balanced on devices, and devices should be balanced on channels. Ideally, each library should be on a different device, and each device should be on a different channel.

If all libraries are placed on the same device, throughput will be decreased because of excessive arm interference. However, when more than one library is placed on an IBM 2314 Direct Access Storage Facility, IBM 2319 Disk Storage, or IBM 3330 Series Disk Storage device, arm movement can be reduced by placing the volume table of contents (VTOC) approximately midway between the first and last cylinders being used. The libraries, starting with the most frequently referenced, can then be alternately placed on both sides of the VTOC with the least referenced libraries farthest from the VTOC.

For improved system efficiency in VS2, it is recommended that the following libraries be allocated on cylinder boundaries:

- SYS1.LINKLIB
- SYS1.MACLIB
- SYS1.PROCLIB
- SYS1.SYSJOBQE

The PRESRES Volume Characteristics List

This topic describes the creation and use of a direct access volume characteristics list that is placed in the system parameter library under the member name PRESRES (permanently resident and reserved).

Characteristics of the PRESRES List

The PRESRES volume characteristics list defines the mount characteristics (permanently resident, reserved) and allocation characteristics (storage, public, private) for any, or all, direct access volumes used by an installation. Use of the PRESRES list can only be suppressed by deleting the member from the parameter library (SYS1.PARMLIB).

The scheduler, after receiving control from the nucleus initialization program (NIP), compares the volume serial numbers in the PRESRES characteristics list with those of currently mounted direct access volumes. Each equal comparison results in the assignment to the mounted volume of the characteristics noted in the PRESRES entry. (Fields in the unit control block for the device on which the volume is mounted are set to reflect the desired characteristics.) If the volume is the IPL volume or the volume containing the data sets SYS1.LINKLIB, SYS1.PROCLIB, SYS1.SYSJOBQE, SYS1.PAGE, or a volume that cannot physically be demounted, the mount characteristic (permanently resident) has already been assigned and only the allocation characteristic is set.

A mounting list is issued for the volumes in the PRESRES characteristics list that are not currently mounted (except those for which mounting messages have been suppressed), and the operator is given the option of mounting none, some, or all of the volumes listed. The mount and allocation characteristics for the volumes mounted by the operator are set according to the PRESRES list entry for the volume; the operator mounts the unit on the volume he selects. The mounting list is subject to the following restrictions:

- A PRESRES entry identifying a volume that cannot physically be demounted will appear in the mounting list issued to the operator if the volume (device) is OFFLINE or is not present in the system.
- Only the first 102 volumes on the PRESRES list can be placed on the mount list.

After the scheduler has finished PRESRES processing, reading of the job input stream begins, and the PRESRES list is not referred to again until the next IPL.

To assign allocation characteristics other than "public" to volumes whose mount characteristic is "permanently resident", users can use a PRESRES characteristic list entry.

Selection of the volumes for which PRESRES entries are to be created should be done so that critical volumes are protected. Since the combination of mount and allocation characteristics assigned to a specific volume determines the types of data sets that can be placed on the volume and the volume's usage, the user can exercise effective control over the volume through a PRESRES list entry.

Writing the PRESRES Entry

Each PRESRES entry is an 80-byte record, consisting of a 6-byte volume serial number field, a 1-byte mount characteristic field, a 1-byte allocation characteristics field, a 4-byte device type field, a 1-byte mount-priority field, and an optional information field. Commas are used to delimit the fields, except that the optional information field is always preceded by a blank. All character representation is EBCDIC. The format of the entry is described below.

- Volume serial number -- up to six characters, left justified.
- Mount characteristic -- 0 to denote permanently resident, 1 to denote reserved. The default characteristic is "permanently resident" and is assigned if any character other than 0 or 1 is present in the field.
- Allocation characteristic -- 0 to denote storage, 1 to denote public, 2 to denote private. The default characteristics is "public" and is assigned if any character other than 0, 1, or 2 is present in the field.
- Device type -- a four-digit number designating the type of direct access device on which the volume resides, e.g. the IBM 2319 Disk Storage is indicated by the notation 2319. Note that this field only indicates the basic device type for the associated volume. The operator should be advised if the device requires special features to process the data on the designated volume.
- Mount priority -- N to denote that mount messages at IPL time for a volume should be suppressed. This field allows the user to list seldom used volumes in the PRESRES list without having a mount message issued at each IPL. When these volumes are required, they may be mounted and attributes will be set from the PRESRES list entry. If the mount message is not to be suppressed, the mount priority field and the preceding comma may be omitted.
- Optional information -- descriptive information about the volume. This information is not used by the system, but will be available to the user on a printout of the list. If necessary, comments may start in the second byte after the mount priority field or, if the mount priority field is omitted, in the second byte following the comma after the device type field.

Embedded blanks are not permitted in the volume serial, mount, allocation, or device type fields.

Adding the List

The IEBUPDTE utility program places the list (under the member name PRESRES) in the system parameter library, SYS1.PARMLIB. This utility is also used to maintain the list.

Job Management and Supervisor Services for System Programmers

This chapter describes how to modify, extend, or implement some of the job management and supervisor facilities of the control program. The information is designed primarily for system programmers responsible for maintaining, updating, and extending these facilities.

Descriptions of macro instructions needed to implement some of the above facilities are in the following chapter, "Supervisor Macro Instructions for System Programmers".

Note: For coverage of data management services for system programmers, see *OS/VS Data Management for System Programmers,* GC28-0631.

Job Classes

In VS2, the user can control not only the priority of jobs but also the actual mix of jobs in the system; this is done by assigning jobs to job classes. The job class is assigned on the JOB statement (CLASS=jobclass), and when an initiator that can handle that particular class is started, the job will be processed in priority order.

There are no absolute rules for assigning job classes, and some experimentation is necessary. Generally, jobs of similar characteristics should be assigned to the same class. For example, if several jobs are time-dependent and execute in nonpageable dynamic storage, it may not be desirable to tie up all of nonpageable dynamic storage by having these jobs running concurrently. These jobs may all be assigned to class B (or C or D -- class names have no inherent meaning); then, if only one initiator is started that can handle class B jobs, there will never be more than one of these jobs executing at once. In this manner, sections of storage are free to process other jobs.

Suppose the following assignments are made:

Class B = jobs that are time-dependent.

Class C = jobs of less than one minute running time.

Class D = jobs with high I/O requirements.

With these assignments the operator may issue these commands:

```
START INIT,,,BCD
START INIT,,,CDB
START INIT,,,DCB
```

If the three initiators are processing jobs with the same priority and all necessary resources (for example, I/O devices and data sets) are available, then three jobs, one from each of the three different classes, would run concurrently. If a job from one of the classes has higher priority than the others, it will be initiated first, assuming all necessary resources are available.

When the operator starts an initiator, he indicates what class or classes it can handle (START INIT,,,A). An initiator can handle up to fifteen job classes, and as many as sixty-three initiators can be running at once. (This means that a maximum of 63 user-programs in pageable dynamic storage can be operating concurrently in addition to 14 system tasks and jobs in nonpageable dynamic storage). The operator's job might be simplified by having mnemonically named catalog procedures for these initiators.

If the system uses automatic volume recognition (AVR), efficiency can be improved by assigning jobs that require non-resident volumes to the same class. With AVR, if volumes required by a job are not mounted at allocation time (and the DEFER subparameter is not specified), no other jobs can be initiated until the required volumes are mounted by the operator. Thus, if jobs that will require non-resident volumes are assigned to the same class, only one initiator will be issuing mount messages and it will be easier for the operator to anticipate which volumes to mount. Also, performance can be improved by assigning these

jobs the same priority; if they run at the same priority, they will be initiated on a first-in-first-out basis, and the operator can anticipate which volumes to mount next.

The job class is specified as a parameter on the JOB statement. Its format is:

//jobname JOB CLASS=jobclass

Jobclass may be any single letter from A-O. If no job class is specified, the default will be class A.

An initiator that can handle all job classes in the system should always be running. If a job is submitted and there is no initiator running for all the job classes, it will remain in the input queues indefinitely or until the operator checks the input queues and discovers the job (by means of the DISPLAY N command).

For more information, see *OS/VS JCL Reference*, GC28-0618.

Job and Dispatching Priorities

While job classes control which jobs are to be initiated, job priorities control the order of the jobs in each class. Priorities can range from 0-13 (where 13 is the highest). The priority is assigned as a parameter on the JOB statement as follows:

//jobname JOB PRTY=priority

The priority can be any decimal integer from 0-13. If no priority is specified, the default priority as specified in the reader catalog procedure is assumed. There are no absolute rules for assigning job priorities; they will depend on the job mixes and turnaround requirements of the jobs that are run.

The actual value used by the supervisor to determine which task receives control of the CPU next is dispatching priority. It may be calculated from the job priority stated on the JOB statement, or the dispatching priority may be stated explicitly on the EXEC statement by using the DPRTY parameter. Dispatching priorities should be assigned with the assumption that tasks of higher priority will be given control when competing with tasks of lower priority. Tasks with a large number of input/output operations should be assigned higher dispatching priorities than tasks with little input/output.

SYSOUT and Message Classes

Output from a problem program is assigned to an output class which will be processed by output writers. A maximum of 36 SYSOUT classes may be named with single letters (A-Z) or digits (0-9) for output class names. The names have no inherent meaning but are simply used to group output of similar characteristics. Writers are assigned to process only certain classes of output; these classes may be assigned in the output writer cataloged procedure, or the operator may assign them as parameters in the START command.

Careful planning of output classes can improve throughput at an installation. By assigning jobs to carefully planned output classes and making sure that operators know which writers to run at all times, all output devices are used constantly and there is no wasteful contention for these devices. If output is assigned to a class for which no writer is started, it will remain indefinitely in the output queue occupying direct access space that might be needed by another program.

In planning SYSOUT classes, the following should be considered:

- A special output class may be reserved for very high priority jobs. For instance, only jobs requiring turnaround time of less than an hour may be assigned a class of A.
- If the system has a universal character set printer that will be used as an output device, a separate output class may be assigned to each character set image stored in the system library. This will minimize the changing of printer chains and trains.

- If the system has a printer with a forms control buffer (FCB), a separate output class may be assigned to each FCB image. This will minimize the changing of printer forms.
- Although a writer may handle as many as eight different output classes, it can only write output on one device. For example, if the following command is issued,

  START WTR,00E,,ABCDEF

the writer started would handle six different classes, but all these classes would be written on the printer at the specified address 00E. If one of the classes is to be written on tape, it could not be handled by this writer.

The output class is specified as a parameter on the DD statement defining the output data set; its format is SYSOUT=x where x is any single letter (A-Z) or digit (0-9).

System messages that are generated during the execution of a program must also be routed to an output device; if it is desired that messages appear with their program output, messages should be assigned to the same message class as the output.

The message class is assigned as a parameter of the job statement; its format is MSGCLASS=x where x is any single letter (A-Z) or digit (0-9). If no message class is specified, the default class specified in the RDR procedure is used.

Standard IBM Cataloged Procedures

After the job classes and output classes are planned and it is decided what job mixes are needed to run concurrently, catalog procedures for readers, writers, and initiators can be planned. The operator's job can be simplified and time can be saved by having several cataloged procedures for each of these and having them named so that operators can tell what they do by their names. For instance, a reader that reads from tape with a blocking size of 3200 could be named RT32. An output writer that writes class A output to a printer could be WPA. An initiator cataloged procedure that processes class A FORTRAN jobs might be IFRTA. By thus naming procedures, when one of the system tasks stops or is waiting for work, the operator will know what task is involved by its name.

Generally, the cataloged procedures should be as specific as possible in providing parameters so the operator need only type the START command and the name of the procedure. If parameters are to be changed (for instance the blocksize in a reader procedure), symbolic parameters should be used. The operator can type in the changed parameter, and it will override the one in the procedure. The START command can also start problem programs, but SMF (system management facilities) will not be recorded, nor will checkpoint/restart be done for these jobs.

SYSIN and SYSOUT Data Blocking

Blocking input stream records reduces the time required to access and process them. If the records are blocked, auxiliary storage space is conserved since more records can occupy the same amount of space. There are two types of blocking for the input reader:

- input to the reader from the job stream
- output from the reader (input to the processing program)

If the input is to be blocked, the number of buffers and their sizes are specified on the IEFRDER statement in a reader cataloged procedure. This is dependent on the type of input device used. These can be overridden by specifying the new buffer sizes and number in the START command for the reader.

The output from the reader (the input stream data that is transferred from the input stream to a direct access device) can be blocked. The block size is indicated in the IEFDATA statement in the reader cataloged procedure.

Blocking system output will improve performance since it also reduces disk arm interference; of course, at the same time, it requires additional virtual storage allocation for the problem

program region, the reader, and the output writer. The additional space required in each case is equal to the logical record length times the blocking factor plus the input buffer space. This blocking is specified in the program which writes the output data set on a direct access device (for later writing by the writer), and should be considered when specifying region size in the writer procedure.

Cataloged Reader Procedures

IBM supplies four cataloged procedures for readers. These procedures can be used and modified, if desired, or the user's own procedures can be written. Every cataloged procedure for a reader requires four job control statements:

- An EXEC statement named IEFPROC to specify the reader.
- A DD statement named IEFRDER to provide the reader with a description of the input stream.
- A DD statement named IEFPDSI to describe the procedure library.
- A DD statement named IEFDATA to define the spooling, or concurrent peripheral operation (CPO), data set that is used for intermediate storage of input stream data.

RDR, RDR400, and RDR3200

The readers RDR, RDR400, and RDR3200 provided by IBM are essentially the same except for different blocking factors.

The standard reader procedure supplied by IBM is named RDR. It specifies a block size of 80 bytes for the concurrent peripheral operation (CPO) data set.

```
//IEFPROC   EXEC    PGM=IEFIRC,REGION=10K,                          X
//                  PARM='00103005001024905010SYSDAbbbE00001A'

//IEFRDER   DD      UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,         X
//                  DISP=OLD,                                       X
//                  DCB=(BLKSIZE=80,BUFL=80,                        X
//                  BUFNO=1,RECFM=F)

//IEFPDSI   DD      DSNAME=SYS1.PROCLIB,DISP=SHR

//IEFDATA   DD      UNIT=SYSDA,                                     X
//                  SPACE=(80,(500,500),RLSE,CONTIG),               X
//                  DCB=(BLKSIZE=80,LRECL=80,BUFL=80,               X
//                  BUFNO=2,RECFM=F,DSORG=RD)
```

A second reader procedure supplied by IBM is named RDR400. It specifies a block size of 400 bytes for the CPO data set.

```
//IEFPROC   EXEC    PGM=IEFIRC,REGION=12K,                          X
//                  PARM='00103005001024905010SYSDAbbbE00001A'

//IEFRDER   DD      UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,         X
//                  DISP=OLD,                                       X
//                  DCB=(BLKSIZE=80,LRECL=80,BUFL=80,               X
//                  BUFNO=1,RECFM=F)

//IEFPDSI   DD      DSNAME=SYS1.PROCLIB,DISP=SHR

//IEFDATA   DD      UNIT=SYSDA,                                     X
//                  SPACE=(80,(500,100),RLSE,CONTIG),               X
//                  DCB=(BLKSIZE=400,LRECL=80,BUFL=400,             X
//                  BUFNO=2,RECFM=FB,DSORG=PS)
```

A third reader procedure supplied by IBM is named RDR3200. It specifies a block size of 3200 bytes for the CPO data set.

```
//IEFPROC   EXEC    PGM=IEFIRC,REGION=14K,                          X
//                  PARM='0010300500102490501OSYSDAbbbE00001A'

//IEFRDER   DD      UNIT=2400,LABEL=(,NL),VOLUME=SER=SYSIN,         X
//                  DISP=OLD,                                       X
//                  DCB=(BLKSIZE=80,LRECL=80,BUFL=80,               X
//                  BUFNO=1,RECFM=F)

//IEFPDSI   DD      DSNAME=SYS1.PROCLIB,DISP=SHR

//IEFDATA   DD      UNIT=SYSDA,                                     X
//                  SPACE=(80,(500,12),RLSE,CONTIG),                X
//                  DCB=(BLKSIZE=3200,LRECL=80,BUFL=3200,           X
//                  BUFNO=1,RECFM=FB,DSORG=PS)
```

The IBM-supplied procedures can be adapted by an installation to meet its needs. The parameters to be specified for the EXEC and DD statements are described in the following sections.

EXEC Statement

The EXEC statement specifies the reader and its region size. It also passes a set of parameters to the reader. Its format is:

```
//IEFPROC   EXEC    PGM=IEFIRC, REGION=nnnnnK,                      X
                    PARM='bpptttooommmiiicccrlsssssssssaaaaefh'
```

The step name must be IEFPROC as shown. The parameter requirements are:

PGM=IEFIRC
specifies the reader. Its name is IEFIRC.

REGION=nnnnnK
specifies the region size for the reader. The value nnnnn represents a number from one to five digits that is multiplied by K (1024 bytes) to designate the region size. The region requirement depends on the size of the buffers . An insufficient size specification will result in an abnormal termination. If a blocked procedure library is used, the region size must be increased by the block size. This allows for the increase in buffer size. If double buffering is used, the region size must be increased by twice the block size .

PARM='bpptttooommmiiicccrlsssssssssaaaaefh'
is a set of parameters for the reader. This parameter field must consist of 35 characters. Their meanings are:

b
any character from 0 through 9 or A through F indicating whether an account number is required and whether a programmer name is required. The following chart shows the meaning of each possible character.

Characters	Accounting Information Required?	Programmer Name Required?
0,4,8, or C	no	no
1,5,9, or D	no	yes
2,6,A, or E	yes	no
3,7,B, or F	yes	yes

pp
two numeric characters from 00 to 14 indicating the default priority for jobs read from this input stream. When no priority is specified in the JOB statement, the default priority is assigned to the job. Priority 14 should be avoided because it is used by the system to expedite the processing of certain jobs.

ttt
three numeric characters indicating the default for the maximum time (in minutes) that each job step may run.

ooo
three numeric characters indicating the default for the number of primary tracks assigned for SYSOUT data sets. This primary allocation should meet most needs, so that secondary allocation will not usually be needed.

mmm
three numeric characters indicating the default for the number of secondary tracks assigned for SYSOUT data sets.

iii
three numeric characters less than 255 indicating the dispatching priority of this reader while it is processing JCL statements.

ccc
three numeric characters indicating the default for the region size (specified as a number of 1024 byte blocks) assigned to job steps read from this input stream.

r
a numeric character from 0 to 3 that specifies the disposition of commands read from the input stream. The character has the following meanings:

0 - The reader passes the command to the command scheduling routine to be executed.

1 - The reader displays the command (via a WTO macro instruction), and passes it to the command scheduling routine to be executed.

2 - The reader displays the command (via a WTO macro instruction), asks the operator whether the command should be executed (via a WTOR macro instruction), and passes the command to the command scheduling routine if the operator replies yes.

3 - The reader ignores the command and treats it as a "no operation".

The WTO and WTOR macro instructions issued by the reader are sent to the MCS master console.

l
a numeric character 0 or 1 specifying the bypass label processing option. 0 signifies that the bypass label processing parameter in the label field of a DD statement is to be ignored; the label parameter is processed as no label. 1 signifies that the bypass label processing is not to be ignored; the label parameter is processed as it appears.

ssssssss
eight alphameric characters specifying the default device for SYSOUT. This becomes the UNIT subparameter in the DD statement defining SYSOUT (if the UNIT field is omitted

from the DD statement). If the designation is fewer than eight characters, the ssssssss field must be padded to the right with blanks.

This default device can be specified by its address, group, or type. However, the UNIT=type form may cause all units of that type to be used for system output, since the device allocation program spreads the data sets among all candidate devices. To reserve some devices for private volumes, the UNIT group defined should be a subset of the available direct access devices. The name SYSOUT may be specified as the default unit name for the system output data sets if it was specified at system generation; when this default is used, a unit count of 1 is implied.

aaaa

four hexadecimal numbers from 0000 to E000 indicating which operator command groups are to be executed if read from this input stream. Four blanks default to 'E000'.

The following table shows the operator commands that are affected by the aaaa parameter. The commands are grouped by function. If the command is in a group authorized by the aaaa parameter, it is processed. If the command is not authorized by the aaaa parameter, it is ignored and an error message is sent to the master console.

Note: Informational commands (Group 0) are always valid when entered into the input stream.

Bit settings for the aaaa parameter are

Byte	Bits	Bit Settings	Meaning
0	0	1	Group 1 commands executed
	1	1	Group 2 command executed
	2	1	Group 3 commands executed
	3-7	00000	Reserved
1	0-7	00000000	Reserved

Example: If commands from command groups 2 and 3 to be executed when entered into the input stream, code the aaaa parameter: "6000".

Command Group	Function	Commands		
0	Informational	BRDCST	LOG	REPLY
		CONTROL	MONITOR	SEND
		DISPLAY	MSGRT	STOPMN
1	System Control	CANCEL	MODIFY	START
		HALT	RELEASE	STOP
		HOLD	RESET	WRITELOG
		MODE	SET	
2	I/O Control	MOUNT	UNLOAD	VARY
		SWAP		
3	Console Control	VARY		
1,2,3	Master Console	All commands are valid, plus VARY CONSOLES VARY HARDCPY VARY MSTCONS		

Note: VARY (Group 2) is accepted only for a non-console device online or offline. VARY (Group 3) provides only for console switching and console reconfiguration or secondary consoles.

ef

MSGLEVEL value in absence of a value in the JOB statement. If there is no MSGLEVEL= parameter in the JOB statement, job control statements and

allocation/termination messages are recorded in the system output data set according to the value of the ef parameter. The values and their effects are:

e

Kinds of job control statements recorded.

0 - JOB statement only.
1 - Input statements, cataloged procedure statements, and symbolic parameter substitution values.
2 - Input statements only, including instream procedures.

A blank defaults to a value of 0.

f

Kinds of allocation/termination messages recorded.

0 - None, except in the case of an abnormal termination. (In that event, all messages are recorded.)
1 - All.

A blank defaults to a value of 1.

h

MSGCLASS default value (A-Z, 0-9). If there is no MSGCLASS keyword parameter in the JOB statement, job control statements and allocation/termination messages are recorded according to the message class specified by this character. If the character is blank or absent, A is the default class.

DD Statement for the Input Stream

The procedure for the reader must include a DD statement that describes the input stream. The format for this statement is:

```
//IEFRDER   DD  UNIT=device,LABEL=( ,type),VOLUME=SER=SYSIN,      X
//              DCB=( list of attributes) [,DSNAME=name,          X
//              DISP=OLD]
```

The DD name must be IEFRDER as shown. The IEFRDER statement can be overridden with a START command. The parameter requirements are as follows:

UNIT=device

specifies the device from which the input stream is to be read. This can be any device supported by the queued sequential access method (QSAM). The device can be specified by its address, type, or group.

LABEL=(,type)

describes the data set label (needed only for tape data sets). If this parameter is omitted, a standard label is assumed.

VOLUME=SER=SYSIN

specifies the volume containing the input stream. This parameter is required for magnetic tape or direct access volumes. The serial SYSIN is recommended for identification of this volume, but other serials can be used.

DCB=(list of attributes)

specifies the characteristics of the input stream and the buffers. If the BLKSIZE, LRECL, and BUFL subparameters are not specified, an 80-byte value is assigned to each. Other subparameter fields may be specified as needed; if they are not specified, the QSAM default attributes are assigned, as follows:

BUFNO - two buffers
RECFM - U-format, with no control characters
TRTCH - odd parity, no data conversion, and no translation
DEN - lowest density

DSNAME=name
specifies the name of the input stream data set to be read. This keyword should be used only with direct access or tape input stream.

DISP=OLD
specifies that the input stream is an existing data set.

DD Statement for the Procedure Library

The procedure for the reader must include a DD statement that defines the procedure library. This statement must follow the IEFRDER statement which describes the input stream. The format for this statement is:

```
//IEFPDSI   DD  DSNAME=SYS1.PROCLIB,DISP=SHR
```

The DD name must be IEFPDSI as shown. The parameter requirements are as follows:

DSNAME=SYS1.PROCLIB
identifies the procedure library. To concatenate other data sets with the system library, the IEFPDSI DD statement may be followed by other unnamed DD statements, thus expanding the system procedure library.

DISP=SHR
specifies that the procedure library is an existing data set and can be shared with other tasks.

DD Statement for the CPO Data Set

The procedure for the reader/interpreter must include a DD statement that defines the spooling, or CPO (concurrent peripheral operation) data set. Two DCB parameters (BLKSIZE and BUFNO) may be overridden by parameters in the input stream on DD * and DD DATA statements. The CPO data set is used for intermediate storage of input stream data. The format for this statement is:

```
//IEFDATA   DD  UNIT=device,                                  X
//              SPACE=(units,(quantities)[,RLSE,CONTIG]),     X
//              VOLUME=SER=volser,DISP=(status,disp),         X
//              DCB=(list of attributes),DSORG=PS
```

This DD name must be IEFDATA as shown. The parameter requirements are as follows:

UNIT=device
specifies one or more direct access devices on which data sets from the input stream will be written. If more than one device is provided, the different data sets are not necessarily written in a continuous manner from device to device. Instead, the different data sets might be spread among the available devices according to a reader algorithm based on priorities and optimum access. If all the input stream data sets are to be written on the same device, the VOLUME parameter (described below) should be used in this DD statement to identify the specific volume. The DEFER option must not be used.

Caution: The UNIT group names should not be used unless the request is for no more than one device, or unless the group is defined to have devices of only one type.

SPACE=(units,(quantities)[,RLSE,CONTIG])
specifies space allocation for the direct access volume. The optional RLSE subparameter releases all unused space to the system when the data set is closed. The optional CONTIG subparameter ensures that space is allocated in contiguous tracks or cylinders.

VOLUME=SER=volser
identifies a specific direct access volume. This parameter is not required, but can be used to cause all input stream data sets to be written on the same volume. This parameter should be used if the DISP parameter is specified.

DISP=(status,disp)
specifies the status and disposition of the CPO data set. This parameter is not required, but can be used to bypass the first space allocation defined on the SPACE parameter. To do this, specify the parameter as DISP=OLD. The system then assumes that the data set exists, and does not allocate space for the reader/interpreter program. Subsequently, the reader/interpreter forces a DISP=(NEW,PASS) status for the CPO data set so that space is allocated on it for recording the input stream data sets.

DCB=(list of attributes)
specifies the characteristics of the CPO data set and the buffers to be used by the data set. The RECFM and the LRECL subparameters cannot be overridden and should not be specified. The values for these subparamaters are RECFM=FB and LRECL=80. The BLKSIZE and BUFL sybparameters must be specified in the IEFDATA DD statement. The BLKSIZE and BUFNO values may be overridden by specifying them on a DD * or DD DATA statement in the reader input stream. However, the BLKSIZE and BUFNO values on the IEFDATA statement are always used as upper limits. Thus, if the overriding statements exceed these limits, the IEFDATA values are used. The BUFNO and RECFM subparameters, if not specified, assume the QSAM default attributes as follows:

BUFNO -- two buffers.
RECFM -- U-format, with no control characters.

DSORG=PS
must be coded as shown.

IEFREINT

The procedure named IEFREINT is used to process job control statements for a job being restarted, and is a skeleton of the normal reader procedures. Its main functions are to define the restart reader program, named IEFVRRC, and to make the procedure library accessible to that program. The procedure is:

```
//IEFPROC  EXEC  PGM=IEFVRRC,  RESTART READER PROGRAM
//               REGION=50K,   RESTART READER REGION
//               PARM=RESTART
//IEFRDER  DD    DUMMY
//IEFPDSI  DD    DSNAME=SYS1.PROCLIB,DISP=OLD PROCEDURE LIBRARY
//IEFDATA  DD    DUMMY
```

The IBM-supplied procedure can be adapted by an installation to meet its needs. The parameters to be specified for the EXEC and DD statements are described in the following sections.

EXEC Statement

The EXEC statement specifies the reader and its region size. It also passes a parameter to the reader program. The format for the EXEC statement is:

```
//IEFPROC  EXEC  PGM=IEFVRRC,REGION=nnnnnK,PARM=RESTART
```

The step name must be IEFPROC, as shown. The parameter requirements are:

PGM=IEFVRRC

specifies the reader program. The name of the program must be IEFVRRC, as shown.

REGION=nnnnnK

specifies the region size for the reader. The value nnnnn represents a number from one to five digits that is multiplied by K (K=1024 bytes) to designate the region size. The region requirement depends on the size of the buffers . An insufficient size specification will result in an abnormal termination. If blocked procedure library has been specified, the region size must be increased by the block size. This is to allow for the increase in buffer size.

PARM=RESTART

must be coded as shown.

DD Statement for the Input Stream

The procedure for the restart reader must include a DD statement that describes the input stream. The format for this statement is:

```
//IEFRDER  DD  DUMMY
```

This statement must be named IEFRDER, as shown. The parameter requirements are:

DUMMY

must be coded as shown. System input is taken from the SYS1.SYSJOBQE data set which is already open.

DD Statement for the Procedure Library

The procedure for the restart reader must include a DD statement that defines the procedure library. This statement must follow the IEFRDER statement which describes the input stream. The format for this statement is:

```
//IEFPDSI  DD   DSNAME=SYS1.PROCLIB,DISP=SHR
```

This statement must be named IEFPDSI, as shown. The parameter requirements are:

DSNAME=SYS1.PROCLIB

identifies the procedure library. To concatenate other data sets with the system library, the IEFPDSI DD statement may be followed by other unnamed DD statements, thus expanding the system procedure library.

DISP=SHR

specifies that the procedure library is an existing data set. The procedure library is assigned the share status (SHR) when referred to by the reader.

DD Statement for the CPO Data Set

The procedure for the restart reader must include a DD statement that defines the CPO (concurrent peripheral operation) data set. Since the data is already in the checkpoint data set, DUMMY serves as the operand. The format for this statement is:

```
//IEFDATA  DD   DUMMY
```

This statement must be named IEFDATA, as shown. The parameter requirement is:

DUMMY
must be coded as shown.

Cataloged Initiator Procedures

Different initiator cataloged procedures may be written for the different types of jobs that initiators will handle. There may be a FORTRAN initiator (an initiator to handle the job classes to which FORTRAN jobs are assigned), a COBOL initiator, an initiator for I/O bound jobs or an initiator for CPU bound jobs.

A cataloged procedure for an initiator requires only one job control statement: an EXEC statement. Additional DD statements may be optionally added so that specific control volumes are mounted before an initiator is started.

An EXEC statement named IEFPROC specifies the initiator program and any job classes to be associated with the initiator (if the START command does not specify job classes). Optional DD statements specify control volumes to be allocated to the initiator task.

INIT

The standard initiator cataloged procedure supplied by IBM is named INIT. The procedure is:

```
//IEFPROC   EXEC        PGM=IEFIIC,PARM='A,LIMIT=13'
```

User-written initiator procedures must follow the format for the standard procedure. The parameters to be specified for the EXEC and DD statements are described in the following sections.

EXEC Statement

The EXEC statement specifies the initiator program and passes a set of parameters to it. The format for the EXEC statement is:

```
//IEFPROC   EXEC    PGM=IEFIIC,PARM='x[ (n)][,x1[(n1) ]... [,LIMIT=K]]'
```

The step name must be IEFPROC, as shown. The parameter requirements are as follows:

PGM=IEFIIC
specifies the initiator program. The name of the program must be IEFIIC, as shown.

PARM='x[(n)][,x_1[(n_1)]...[,LIMIT=K]]'

x - Job class. (Letter A - O. One to fifteen job classes may be named.)

n - (0 - 15) A *force value* priority at which all jobs from the preceding class will be run.

K - (0 - 15) The priority above which no jobs will be run by this initiator.

If the START command for an initiator includes any job class references, all job class definitions in the cataloged procedure are voided.

The LIMIT=K entry in the cataloged procedure means that no job may be run at a priority higher than the value indicated by K. The force value (n above) is used for a job *unless* it is greater than the limit value (K above). A force value (n) priority may not always be specified; if it is not, and the limit value, K, is not exceeded the priority is determined by the following order:

- The EXEC statement
- The JOB statement
- The cataloged reader procedure

If a job class is assigned a force priority, it overrides the priority indicated in any of the above three sources.

DD Statements for the Control Volumes

DD statements for control volumes are optional. The standard procedure INIT does not include a DD statement for a control volume.

A control volume that will be referred to during a catalog search can be mounted before the search begins. DD statements for control volumes may be included in initiator procedures cataloged in the procedure library (SYS1.PROCLIB). Such DD statement cause direct access volumes to be mounted and allocated for the initiator. This facility is particularly useful when control volumes will be needed for job batches.

Initiation with a DD statement for a control volume ensures that the control volume will be mounted prior to a catalog search for a specified data set. If such DD statements for control volumes are not included in initiator procedures, an attempt will be made to mount a required control volume if a catalog search can not be completed during allocation for a step. However, when control volumes are mounted in this manner, they are eligible for demounting immediately after the catalog search has been completed and will not necessarily remain mounted for the life of the job or job step requiring them.

By starting an initiator that includes a DD statement for a control volume, mounting is requested before the initiator is allowed to start initiating jobs. If the volume is already mounted, the initiator proceeds with initiation.

When a STOP command is issued for the started initiator and the volume is demountable and PRIVATE, it will be demounted if no other job steps or initiators are allocated to the volume. The volume would stay mounted until the last job step using it terminates or until the initiators using it are stopped, at which time the volume would be demounted.

As many volumes may be defined by DD statements in the initiator procedure as the user finds useful. The following is an example of a DD statement that could be included in an initiator procedure for a control volume:

```
//ddname    DD  VOLUME=(PRIVATE,SER=ser#),UNIT={address}    ,DISP=SHR
                                              {type   }
                                              {group  }
```

VOLUME=(PRIVATE,SER=ser#),
specifies the volume serial of the control volume. PRIVATE ensures that this volume will not be used to satisfy job step data set requests unless requested by the specific volume serial number. Also, unless already mounted and permanently resident or reserved, the volume will be demounted when the initiator is stopped, when last used by job steps being processed by other initiators, or when other initiators allocated to the volume are stopped.

UNIT= {address / type / group}

specifies the unit address, unit type, or group on which the control volume is to be mounted.

DISP=SHR

specifies that a temporary data set will not be allocated to the volume. A dsname will be generated for the temporary data set and when the initiator is stopped, a message will be written on the system output data set that the data set has been kept. This message can be ignored as no action needs to be taken.

DD Statements for the Dedicated Data Sets

Dedicated data sets save the time taken repeatedly to allocate (and deallocate) space used only temporarily during a job step. A dedicated data set is allocated space when the initiator is started and belongs to the initiator. Every job step running under that initiator can use the dedicated data set as a temporary data set. If dedicated data sets are used for temporary data sets, the checkpoint/restart facility is internally suppressed. To dedicate any data set quickly to successive jobs or job steps, a DD statement is added to the initiator procedure.

A data set is dedicated by adding a DD statement (for each data set to be dedicated) to the initiator procedure. The unit must be a direct access storage device; the space may be for a sequential or partitioned data set. Each DD statement must be of the following form:

```
//ddname        DD  UNIT=unitparms,VOL=volparms,
                SPACE=(kind,(amount,increment,dirblks)),
                DISP=(new,delete)
```

ddname

specifies user-supplied ddname to identify the DD statement. The ddname is used (in the form DSNAME= & ddname) in the DD statement of the problem program job step which is to make use of the dedicated data set.

UNIT=unitparms

specifies parameters that describe the unit to be used for the dedicated data set. The unit must be a direct access storage device. The AFF= and DEFER unit parameters may not be used. The unit parameters specified here override those of the job step DD statement for which the dedicated data set is used.

VOLUME=volparms

specifies volume parameters. A volume may be specified for each unit specified in the preceding unitparms entry. The volume parameters specified here override those of the job step DD statement for which the dedicated data set is used.

SPACE=(kind,(amount, increment,dirblks))

specifies type and size of space (in terms of CYL, TRK, avgbl, or ABSTR) to be allocated to the data set. If ,dirblks is omitted, the data set request implies sequential organization. If ,,dirblks is used, the data set request implies partitioned organization.

When a dedicated data set with partitioned organization reaches an EOV condition, the initiator must be restarted. The DD statement in the problem program job step that is to use a dedicated data set must describe a problem program data set of the same organization as the dedicated one.

DISP=new,delete
specifies these disposition parameters. They may either be coded explicitly or may take effect by default, if the DISP= entry is ommitted.

The effect of new is that the data set is allocated from any available space on the volume, each time a START initiator operator command is used or the system is restarted.

The effect of delete is that the data set is not kept when the initiator is stopped and the space is available for reallocation to other jobs.

If a dedicated data set is to be used temporarily in a job step, the temporary data set should be defined in a DD statement of the form:

```
//ddname        DD  DSNAME=&ddname,                              X
//              SPACE=(avgbl,(amount,increment,dirblks)),        X
//              UNIT=unitparms,DISP=(new,delete),DCB=dcbparms
```

DSNAME= & ddname
specifies the name of the DD statement for the dedicated data set, preceded by an & sign.

(avgbl,(amount,increment,dirblks))
specifies the space request, in terms of average block length only, needed for the temporary data set.

An attempt to allocate the dedicated data set will be replaced by the normal allocation procedure if one of the following conditions is encountered:

- If the total space (primary and increments) requested exceeds the total space (primary and increments) available to the dedicated data set.
- If the use of ,dirblks (presence or absence) differs from that in the DD statement of the dedicated data set, (or if ISAM is specified).
- If the use of ,dirblks requested exceeds the space for ,dirblks specified in the dedicated data set.
- If the space request is shown in other than average block length.
- Although the total space (primary and increments) requested is compared to the total space (primary and increments) available to the dedicated data set, the primary quantity in the DD statement of the initiator procedure will be allocated to the data set, and not the primary quantity requested here. If a secondary quantity is specified, it will override the secondary quantity specified in the initiator procedure's DD statement.

UNIT=unitparms
specifies unit parameters to be used for the temporary data set, if the dedicated data set is not used. The unit may be a magnetic tape unit, as well as a direct access storage device.

DISP=(new,delete)
specifies disposition parameters. They may either be coded explicitly or may take effect through default.

DCB=dcbparms
specifies DCB parameters required for the temporary data set. A previous user may have left the dedicated data set with undesired DCB parameters.

If a secondary increment is coded in the SPACE parameter, the DCB subparameter BLKSIZE should be coded since the system will use it to calculate the number of tracks required to fulfull the secondary quantity request.

INITD

Language processor programs, such as FORTRAN compilers, make much use of temporary data sets. To permit ready use of the dedicated data set feature with IBM-supplied processor procedures, IBM supplies the initiator procedure INITD. It becomes part of the system by inclusion in the SYS1.PROCLIB at system generation time.

INITD is an initiator procedure that dedicates five utility data sets commonly used with IBM-supplied processor procedures. To use the dedicated data set facility with these procedures, the INITD inititator should be started.

Before including the INITD procedure in the system, the space allocations, unit specifications, and ddnames used in the procedure should be reviewed against the system's requirements. If they are significantly different, they should be coded.

The INITD procedure is:

```
//IEFPROC EXEC  PGM=IEFIIC,PARM='A,LIMIT=13'
//SYSUT1    DD  DSNAME=&UT1,SPACE=(1700,(200,100),,CONTIG),UNIT=SYSDA
//SYSUT2    DD  DSNAME=&UT2,SPACE=(1700,(200,100),
                UNIT=(SYSDA,SEP=SYSUT1)
//SYSUT3    DD  DSNAME=&UT3,SPACE=(1700,(200,100)),
                UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2))
//SYSUT4    DD  DSNAME=&UT4,SPACE=(460,(700,100)),
                UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2,SYSUT3))
//LOADSET   DD  DSNAME=&LOADSET,UNIT=(SYSDA,SEP=SYSUT1),
                SPACE=(3600,(100,10))
```

Each statement of the INITD procedure is explained in detail in the following sections.

EXEC Statement

The EXEC statement for the INITD procedure is:

```
//IEFPROC   EXEC    PGM=IEFIIC,PARM='A,LIMIT=13'
```

IEFPROC
: specifies the step name. Must be coded as shown.

PGM=IEFIIC
: specifies the program to be executed in this job step. Must be coded as shown. Whether dedicated data sets are used depends on the DD statements that follow the EXEC statement, not on the name of the program.

PARM='A,LIMIT=13'
: specifies the parameter list for the initiator program. A is the class of jobs to be processed, LIMIT=13 is the dispatching priority limit for this initiator. Both of these values can be overridden by values used with the START command for the initiator.

DD Statements for the Dedicated Utility Data Sets

There are four DD statements in the INITD procedure that allocate space to four commonly used utility data sets. The statements are:

```
//SYSUT1        DD DSNAME=&UT1,SPACE=(1700,(200,100),,CONTIG),UNIT=SYSDA
//SYSUT2        DD DSNAME=&UT2,SPACE=(1700,(200,100)),
                 UNIT=(SYSDA,SEP=SYSUT1)
//SYSUT3        DD DSNAME=&UT3,SPACE=(1700,(200,100)),
                 UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2))
//SYSUT4        DD DSNAME=&UT4,SPACE=(460,(700,100)),
                 UNIT=(SYSDA,SEP=(SYSUT1,SYSUT2,SYSUT3))
```

DSNAME=
  specifies a temporary data set.

SPACE=
  specifies the first three data sets will be assigned space that can accommodate 200 blocks of 1700 bytes. When that space is exhausted, additional space will be allocated for 100 blocks at a time. Additionally, for the fourth data set, SYSUT4, all the primary space is to be allocated for 700 blocks of 460 bytes initially. When exhausted, space is to be allocated for 100 blocks at a time.

UNIT=
  specifies space to be allocated from direct access storage devices. If possible, each data set is to be on a separate device from every other data set to avoid contention for the device.

DD Statement for the LOADSET Data Set

In the INITD procedure, the dedicated data set for the object module -- the LOADSET data set -- is defined as follows:

```
//LOADSET   DD  DSNAME=&LOADSET,SPACE=(3600,(100,10)),
                UNIT=(SYSDA,SEP=SYSUT1)
```

LOADSET
  specifies the dsname of the dedicated data set.

DSNAME= & LOADSET
  specifies a temporary data set.

SPACE=(3600,(100,10))
  specifies space allocation commonly used in compilers.

UNIT=
  specifies space to be allocated on a direct access storage device but not the same one as the SYSUT1 data set.

Miscellaneous DD Statement Considerations

The three topcis described below indicate considerations when using dedicated data sets.

Use of Dedicated Data Sets by Processing Programs for Utility Data Sets: Presently, processor programs show the temporary nature of the utility data sets by omitting a DSNAME= entry. If these DD statements are revised with the addition of a DSNAME= & name entry, the system will attempt to use dedicated data sets of the INITD program for job steps processed under that initiator.

To illustrate the necessary change, the following is a DD statement from a catalog procedure for which a temporary data set will be allocated:

```
//SYSUT1          DD  UNIT=SYSDA,SPACE=(1024,(200,65))
```

The temporary character of this data set is shown by the absence of a DSNAME= entry. To force consideration of the dedicated data set, assuming that the step is running under the INITD procedure, a DSNAME= & name (or & & name) entry should be added referring to the dedicated data set to be considered for use:

```
//SYSUT1          DD  UNIT=SYSDA,SPACE=(1024,(200,65)),DSNAME=&SYSUT1
```

With the addition of the dedicated data set feature, the allocation program now first searches the DD statements in the initiator procedure for an already existing data set with a DD name like that following the & sign (the symbolic name). If the allocation program finds such a data set, it next determines whether the organization (sequential, partitioned) of the dedicated data set is the same as that of the temporary data set, and whether the total space requirements (primary and increments) of the temporary data set fall within the total space allocation of the dedicated data set. If there is no dedicated data set with the symbolic name, if the organizations are not the same, or if the temporary space does not fit within the dedicated space, the initiator will attempt normal allocation. It is for the latter event that unit parameters should be present.

System Library Data Sets as Dedicated Data Sets: System library data sets may be referred to rpeatedly in a batch of jobs. To save allocating the system data set in each job and step, the system data set can be dedicated in an initiator procedure. Caution must be exercised when dedicating system libraries or other non-temporary data sets. The DD statement in the initiator procedure must have the disposition specified as old or share and keep to prevent the deletion of the data set when the initiator is stopped. In the same manner, the disposition on the job step DD statement referring to the dedicated library must also be old or share and keep or pass to allow the dedication to take place without a space comparison. The example data set references are as follows.

The following is the DD statement in a procedure that results in the allocation of a COBOL library to the job step calling the procedure:

```
//SYSLIB          DD  DSNAME=SYS1.COBLIB,DISP=(SHR,KEEP)
```

The explicit data set reference (DSNAME=SYS1.COBLIB) requires a search of the catalog in each job step using the procedure. To save the repeated catalog search, move the DD statement should be moved to the initiator procedure and replaced in the COBECLG procedure with a DD statement in which the DSNAME= & name entry refers to the ddname of the dedicated data set. Allocation treats this as a dedication request, dedicated if so found. The new DD statement in the procedure, after adding the present one to the initiator, is:

```
//SYSLIB          DD  DSNAME=&SYSLIB,DISP=(SHR,KEEP)
```

The result is one catalog search per initiator instead of one catalog search every job step. However, this procedure requires the initiator with the dedicated data set. Using this modified procedure with an unmodified initiator will result in failure to allocate.

Disposition of Temporary Dedicated Data Sets: Allocation/termination routines do not delete temporary dedicated data sets at the end of each job step, but, instead, keep them until the initiator stops. This occurs even if there is a specification of DISP=(NEW,DELETE) or DISP=(MOD,DELETE) on the DD statement for the data set. Therefore, if an attempt is made to use such a data set a second time in the same job, it will contain data from the previous use. This can be a problem if cataloged procedures are being used and the same procedure is used twice within the same job. For example: assume that a procedure is used twice within the same job and it uses a dedicated data set with a disposition of (MOD,PASS) for the compile step and (OLD,DELETE) for the linkage edit step. When the procedure is entered for the second time, the object module produced by the second compile step will be placed in back of the object module produced by the first compile step. Since both object modules are assigned identical names by the compiler, only the first will be linkage edited.

This problem can be avoided by not using dedicated data sets for jobs that run the same cataloged procedure twice. Alternatively, using DISP=(NEW,DELETE), each cataloged procedure could be submitted as separate jobs instead of being submitting as separate job steps within the same job.

The following can be used to determine the disposition, by allocation/termination, of temporary data sets:

- DISP=NEW,OLD,SHR, or MOD is treated as OLD.
- DISP=,DELETE or , KEEP is treated as KEEP.
- DISP=,PASS is trated as PASS.

Cataloged Writer Procedures

A cataloged procedure for output writers requires two job control statements: an EXEC statement and a DD statement.

An EXEC statement named IEFPROC specifies the output writer program.

A DD statement named IEFRDER defines the output data set.

WTR

The standard output writer procedure supplied by IBM is named WTR. The WTR procedure is:

```
//IEFPROC   EXEC    PGM=IEFSD080,REGION=20K,                    X
//                  PARM='PA'
//IEFRDER   DD      UNIT=1403,VOLUME=(,,,35),                   X
//                  DSNAME=SYSOUT,DISP=(NEW,KEEP),              X
//                  DCB=(BLKSIZE=133,LRECL=133,BUFL=133,        X
//                  BUFNO=2,RECFM=FM)
```

When creating an output writer procedure, the procedure format and the statement requirements must be conformed to. The IBM-supplied procedure may be used as an example. The statements are explained individually in the following sections.

EXEC Statement

The EXEC statement specifies the output writer program and its region size. It also passes a set of parameters to the output writer program. The format for the EXEC statement is:

```
//IEFPROC   EXEC    PGM=IEFSD080,REGION=nnnnnK,                    X
//                  PARM='cxxxxxxxx,seprname'
```

The step name must be IEFPROC, as shown. The parameter requirements are as follows:

PGM=IEFSD080
: specifies the output writer program. The name of the program must be IEFSD080, as shown.

REGION=nnnnnK
: specifies the region size for the output writer. The value nnnnn represents a number from one to five digits that is multiplied by K (K=1024 bytes) to designate the region size. The region requirement depends on the size of the buffers and the data set writer used . An insufficient size specification will result in an abnormal termination.

PARM='cxxxxxxxx,seprname'
: is a set of parameters for the output writer program. The first part of this parameter field can contain from two to nine characters. The second part of this parameter field, if specified, is separated from the first part by a comma, and contains a program name from one to eight characters. Both parts of this parameter field are explained below.

c
: an alphabetic character, either P (for printer) or C (for punch), that specifies the type of control characters for the output of the writer.

xxxxxxxx
: from one to eight (no padding required) single-character class names for system output. These characters specify the type of output that the writer can process, and also establish the priority of the output classes, with the highest priority on the left. If class name parameters are included in the START command, they override this entire set of class names in the cataloged procedure.

seprname
: the name of the program (up to eight characters) that provides job separation in the output data set. The named program must reside in the link library (SYS1.LINKLIB) or the LPA library (SYS1.LPALIB). The name IEFSD094 specifies the output separator supplied by IBM, or the name of a user-written program can be specified. This subparameter may be omitted, in which case no output separator is used.

DD Statement for the Output Data Set

The procedure for the output writer must include a DD statement that defines the output data set. The format for this statement is:

```
//IEFRDER   DD      UNIT=device,LABEL=(,type)                      X
//                  VOLUME=(,,,volcount),                          X
//                  DSNAME=anyname,DISP=(NEW,KEEP)                 X
//                  DCB=(list of attributes),                      X
//                  UCS=(code[,FOLD][,VERIFY]),                    X
//                  FCB=(image-id  {,ALIGN }})
                                   {,VERIFY}
```

This DD name must be IEFRDER as shown. The parameter requirements are as follows:

UNIT=device
specifies the printer, magnetic tape, or card punch device on which the output data set will be written.

LABEL=(,type)
describes the data set label (needed only for tape data sets). If this parameter is omitted, a standard label is assumed.

VOLUME=(,,,volcount)
limits the number of tape volumes that can be used by this writer during its entire operation (from the time it is started to the time it is stopped). This parameter is not required for printer or card punch devices.

DSNAME=anyname
specifies a name for the output data set (tape only, for label purposes), so that it can be referred to by subsequent job steps. This name is also necessary for specification of the KEEP subparameter in the DISP field.

DISP=(NEW,KEEP)
specifies the KEEP subparameter to prevent deletion of the output data set (tape only) at the conclusion of the job step.

DCB=(list of attributes)
specifies the characteristics of the output data set and the buffers. The BLKSIZE and LRECL subparameter fields must be specified in all cases. The BUFL subparameter field, if not specified, is calculated on the basis of the BLKSIZE value. Other subparameter fields may be specified as needed; if they are not, they will assume the QSAM default attributes which follow:

BUFNO -- three buffers for the 2540 device, two buffers for all other devices.
RECFM -- U-format, with no control characters.
TRTCH -- odd parity, no data conversion, and no translation.
DEN -- lowest density.

UCS=(code[,FOLD][,VERIFY])
specifies the code for a universal character set (UCS) image that will be loaded into the UCS buffer. FOLD causes bits 0 and 1 to be ignored when comparing characters between the UCS buffer and the print line buffer. This option allows lowercase alphabetic characters to be printed in uppercase by an uppercase print chain or train. VERIFY causes the specified UCS image to be printed for verification by the operator. The UCS parameter is optional, and is valid only when the output device is a 1403 or 3211.

FCB=(image-id { ,ALIGN / ,VERIFY })
causes the forms control buffer (FCB) image with the specified image-id to be loaded into the FCB. One of two optional parameters, ALIGN or VERIFY, can be coded. Either parameter allows the operator to align forms. In addition, VERIFY causes the specified FCB image to be printed for visual verification. The FCB parameter is valid only when the output device is a 3211.

For the processing of output jobs that require special chains for printing, specific classes should be assigned for each different chain. The desired chain can be specified in the writer procedure, and when that writer is started the chain will be loaded automatically. (Printers used with special chains should be named with esoteric device names as defined at system generation time.)

The following sequence is an example of a writer cataloged procedure for the P11 chain.

```
//IEFPROC   EXEC    PGM=IEFSD080,REGION=20K,                          X
//                  PARM='PDEG,IEFSD094'
//IEFRDER   DD      UNIT=SYSPR,DSNAME=SYSOUT,FCB=(STD2,ALIGN),        X
//                  UCS=P11,                                          X
//                  DISP=(,KEEP),DCB=(BLKSIZE=133,BUFL=133,           X
//                  LRECL=133,BUFNO=2,RECFM=FM)
```

If the output device is a 3211, a UCS or FCB image can be loaded dynamically between the printing of data sets. Therefore, a mixture of data sets using different images in a single output class is allowed; however, this may require mounting trains and changing forms, and may not be desirable. When the output device is a 1403, the UCS image is specified at START WTR time and cannot be changed until the writer is stopped; all data sets within an output class must be printed using the same train. This parameter cannot be overridden for a specific data set when using the (asynchronous) sysout writer. The FCB image is ignored when the 1403 is specified.

Job Queue Format

The job queue format is specified when the system is generated and may be altered during subsequent system initialization procedures. Formatting consists of specifying the number of queue records in a job queue logical track, reserving queue records for initiators, the write-to-programmer routines, and reader/interpreters, and reserving queue records for job cancellation.

The basic element of the system job queue (the data set SYS1.SYSJOBQE) is a 176-byte record -- the queue record. The total number of queue records available is fixed by the space allocated to the SYS1.SYSJOBQE data set. Queue records contain the tables, control blocks, and system messages developed by the reader/interpreter, write-to-programmer, and initiator control program routines -- the information used to run a job.

Lack of queue records to work with is not critical for a reader/interpreter routine. Processing of the input job stream assigned to a reader/interpreter is suspended until queue records become available, at which time processing is resumed. *An initiator, however, must have sufficient queue records available to complete the initiation and running of a job, or the job is canceled*. Because one or more reader/interpreters and one or more initiators may be concurrently active, steps must be taken to ensure that queue records are available to each initiator started, so that it may complete its operations. In addition, queue records must be reserved for use by initiators in the event job cancellation does take place. The main function of job queue formatting is to reserve queue records for initiator use.

To format the job queue, each of the following must be designated:

- The number of queue records to be contained in a job queue *logical track* . A logical track consists of a header record (20 bytes) plus the designated number of queue records. Reader/interpreters and initiators are assigned queue records in terms of logical tracks.

- The number of queue records to be reserved for use by an initiator. Each initiator is allocated this number of records. If the allocation is insufficient for the job currently being processed by the initiator, the job is canceled.
- The number of queue records to be reserved for use in case of job cancellation. All initiators that cancel use these queue records. If the allocation is insufficient, the initiator is placed in a WAIT state and a messages issued.
- The number of queue records to be reserved for write-to-programmer routine use for each job that may be started by an initiator.

The balance of the queue (total queue records less the reservations in the last three items above) is available for use by the reader/interpreters.

Initial values for logical track size, queue record reservation for initators, queue record reservation for write-to-programmer, and queue record reservation for job cancellation, should be specified in the SCHEDULR macro instruction parameters JOBQFMT, JOBQLMT, JOBQWTP, and JOBQTMT respectively.

The service aid program IMCOSJQD provides a formatted dump of the entire job queue, or selected portions of it. The formatted dump includes the master queue control record (QCR) which contains the physical parameters of the job queue.

There are no comprehensive, foolproof formulas for calculating values of JOBQFMT, JOBQLMT, JOBQTMT, and JOBQWTP. The values to be estimated are dependent upon the requirements and structure of the jobs to be presented to the system, the number of job steps, the number of I/O devices required, the number and type of data sets, the number of volumes, and most unpredictable, the number of system messages issued during the initiation and running of a job. The rest of this topic provides some basic guidelines for use in determining these values.

Logical Track Size -- JOBQFMT

Logical track size -- the number of queue records in a logical track -- affects the efficient use of queue records. Reader/interpreters and initiators are allocated queue records in terms of logical tracks. Unused queue records in a logical track are *not available* for use by other reader/interpreters or initiators. Therefore, an over-generous logical track size specification results in wasted queue records and reduction of job queue capacity; i.e., the unused queue records, if available, could contain the required information for another job.

Logical track size affects performance to some extent. Specification of a logical track size of 10 queue records or less can result in excessive execution of the track assignment routines, etc., i.e., the "overhead" required to use very small logical track sizes impairs performance.

As a starting point, the default value for JOBQFMT (12 queue records) should be used.

Logical track size (or multiples of it) may correspond to the physical track capacity of the device on which the job queue is resident. For example, if the IBM 2305 Fixed Head Storage unit is to be used, 66 queue records may be contained in one physical track. In this case, a logical track size of 22 queue records should be specified, thereby allocating 3 logical tracks to one physical track (3 x 22 = 66 queue records). The 3 logical track header records (20 bytes each) use up the remaining record.

Logical tracks can contain the same number of queue records as are reserved for initiator use.

Initiator Queue Records -- JOBQLMT

The value specified for JOBQLMT must be large enough for the queue entries of any job that enters the system. The following list shows the factors that affect the value of JOBQLMT:

- Number of entire generation data groups in a job.
- Number of passed data sets in a job.
- Number of devices required for passed data sets.
- Number of volumes containing the data sets in a step.
- Number of system messages issued during initiation of a step.
- Use of automatic restart.

The sum of the queue records required for each of these items provides a JOBQLMT value.

When a START initiator command is issued, a check is made to see if enough free logical tracks are available to provide the required number of queue records for the initiator. If not, the command is rejected.

Each time an initiator is started, the number of records reserved for an intiator is added to the total number of records reserved for active initiators. For example, if the number of records reserved for each initiator is 60, the number of records reserved for termination is 40, and 4 initiators have been started, then the number of records reserved is 340. This total includes 60 records reserved for each initiator, 40 records reserved for termination, and 60 records reserved as a basic threshold.

Number of Generation Data Groups

Each entire generation data group (GDG) used during a job increases the number of queue records needed by an initiator. Two queue records should be reserved for every generation *in excess of the first* in a GDG. One queue record should be reserved for every four GDGs used in a job.

Thus, if a job uses two entire GDGs, one having 5 data sets (generations), and the other having 24 data sets, 55 queue records must be reserved -- (4+23)x2+1.

Number of Passed Data Sets

Two queue records are needed by an initiator for every three data sets passed during a job. If the number of data sets passed is not a multiple of three, queue records must be allocated as if the number of data sets passed was a multiple of three. Thus if one, two, or three data sets are passed, two queue records are allocated; if four, five, or six data sets are passed, 4 queue records are allocated, and so on.

Number of I/O Devices for Passed Data Sets

When a data set being passed requires more than ten I/O devices, one queue record is required by an initiator. This queue record accommodates 43 devices. If the number of required devices exceeds 53, a second queue record is needed. Separate calculations must be made for each data set.

Number of Volumes

An initiator requires queue records for each data set that occupies more than five volumes, and is located by a search of the catalog. (If a data set's location is specified in a DD statement, the reader routines acquire the necessary records.) One queue record is needed if the data set occupies between 6 and 20 volumes; two queue records if 21 to 35 volumes; three if 36 to 50 volumes; and so on. Separate calculations must be made for each data set.

Number of System Messages

An initiator requires queue records for system messages it issues. If it is assumed that each message is 80 characters in length, each queue record holds two messages. Messages from initiators are primarily device allocation, allocation recovery, data set disposition, SMF or accounting messages, and the keep messages for tapes used in each step.

To cover most device allocation messages, one queue record should be allowed for every three DD statements. To cover data set disposition messages, one queue record should be allowed for each DD statement. As part of the data set disposition messages, the SMF or accounting messages should be counted as two lines per queue record. Also two lines should be counted per queue record for tape messages.

Allocation recovery messages apply to devices that are offline. To cover most situations, queue records should be allocated as follows:

- Determine the largest number of devices of a given class that will be offline at any given time.
- Divide by seven.
- Add two.

Since this calculation is for a job step, the result should be multiplied by the number of steps in a large job.

System messages are the least predictable of all the variables used in calculating initiator queue record needs. The number of messages depends on the number of devices offline, the number not available, and the number required at any given time.

The initiator needs queue space for a TIOT (task input/output table) for each step. The space needed can be approximated as follows:

- Determine the number of DD statements in the largest step in a job.
- Multiply the number of DD statements by 20.
- Add 24.
- Divide by 172 and rounding the dividend up.
- Add 1.

This gives the largest amount of queue records required for a job.

Under certain conditions, the initiator may need additional space. Two specific conditions are:

- VOLT (volume table) -- The initiator builds a VOLT, if one does not exist, for all non-specific device requests. One queue record will hold 28 volume serial numbers.
- Mount CVOL (control volume) -- Five records will be created for each CVOL not mounted. The initiator builds a JCT (job control table), a SCT (step control table), a SIOT (step input/output table), a JFCB (job file control block) and a VOLT if a CVOL is not mounted. The initiator writes these queue records into the jobqueue.

Use of Automatic Restart

To use automatic restart in the system, the number or records specified for the JOBQLMT parameter must be substantially increased. Specifically:

- The *initiator* needs its normal set of queue records (described by the JOBQLMT parameter) to initiate the job for the first time; it needs an additional set of records to start a second job while the first job is going through the restart process.
- Since the restart process involves rereading, reinterpreting, and reinitiating the first job, an additional set of *reader/interpreter* records is needed, together with a third set of *initiator* records.

Finally, when checkpoint/restart is being performed, one or two sets of restart housekeeping records are needed. Altogether, the number of records to be specified for JOBQLMT when automatic restart is being used is:

JOBQLMT + (3 x L) + R + (a x 12)

L - Number of records normally specified for JOBQLMT (that is, when automatic restart is not being used).

R - Number of records normally needed by the reader/interpreter.

a=1 - If jobs may be automatically restarted only once.

a=2 - If jobs may be automatically restarted more than once.

12- Number of records needed for restart housekeeping.

If jobs with automatic restart may be held for operator restart, the initiator queue record requirement is further increased, because the system must keep both the queue records for the held jobs and their associated housekeeping records until the job is restarted. The formula then becomes:

JOBQLMT = (3 x L) + R + (a x 12) + H (L + (a x12))

H - Number of jobs that may be held.

Other terms

As explained previously.

Write-To-Programmer Queue Records -- JOBQWTP

Unless specified otherwise, the system allocates two job queue records to the write-to-programmer (WTP) function. Out of the 176 bytes in each of these records, 161 are available for WTP messages. A record can hold as many messages as will fit into the available space, each message occupying 1 byte per character plus 1 byte per message for an initiator assigned serial number.

To change the number of records available for this function, the number should be specified either with the JOBQWTP operand of the SCHEDULR macro instruction in the system generation statements or during initialization in reply to message IEA101A (but only if Q-F was used with the set command). However, since both system and application tasks contend for the space available to an initiator in the system job queue, and since WTP message may be created faster than the writer may be writing them out, caution should be exercised in raising the JOBQWTP value above 2.

Queue Records for Cancellation -- JOBQTMT

If an intiator's queue record requirements exceed the number of queue records reserved for it, the job associated with that initiator is canceled. Queue records must be reserved for this purpose. Enough queue records must be reserved to accommodate two (or more) initiators that may be cancelling concurrently. The JOBQTMT value (like the value JOBQLMT) is unpredictable because of factors such as the installation's configuration, the size of the job being canceled, and the number of jobs that can be multiprogrammed.

The following guidelines should be considered in calculating JOBQTMT:

- Number of devices used during a job.
- Number of jobs that might be concurrently canceled because of insufficient initiator queue records.
- For any system task to be started, combined JCL from its associated catalogued procedure and the START command must first be interpreted. This requires queue records, and the system allows assignment of records for this purpose whenever any logical track are available. During normal use of the queues, this space is always available. However, in order to insure availability of queue records for system tasks when the reserves approach the

critical state, the value of JOBQTMT should be increased over the above amount by the number of records necessary to get tasks started. (This is especially true for writer and initiator tasks, since they return queue records to the system.) This amount may be estimated in a manner similar to calculating JOBQLMT, taking into consideration that each valid START command generates one input and one output queue entry.

Number of Devices

The devices currently assigned to a job are released when the job is canceled. Since messages are issued when devices are released, a number of queue records should be reserved equal to the largest number of devices assigned at any one time to a job, multiplied by two. Thus if the largest job (in terms of devices) has three steps requiring 4, 11, and 8 devices respectively, 22 queue records should be reserved.

Number of Jobs

The number of queue records reserved for cancellation must be large enough to fill the requirements of all jobs being canceled at any one time because of insufficient initiator queue records. If the estimate of initiator queue records was accurate, it is unlikely that more than one job (if any) will be cancelling at any one time.

An initiator that runs out of queue records for cancellation is placed in the wait state and an operator message is issued. This can result in the interlocking of all reader/interpreters, initiators, and sysout writers functioning at the moment.

Output Separation

The system output writer can use the output separator facility to write separation records prior to writing the output of each job. These separation records make it easy to identify and separate the various job outputs that are written contiguously on the same printer or card punch device.

Characteristics of an Output Separator

The system output writer may be used by a problem program to channel its output eventually to a printer or punch. When this is done, however, the system output stream goes uninterruptedly from one job to another, making it difficult to separate the output of one job from that of another, unless output separation is provided for.

The output separator facility of the operating system provides a means of identifying and separating the output of various jobs processed by the same output unit. To do this, the separator writes separation records to the system output data set prior to the writing of each job's output.

The IBM output separator or the user's own output separator can be used.

The output separator function operates under control of the system output writer. The separator program must reside in the link library (SYS1.LINKLIB) or the LPA library (SYS1.LPALIB). Its name, IEFSD094, must be included as a parameter in the output writer procedure -- the second part of the PARM field in the EXEC statement -- to separate job output. (A cataloged procedure for the writer is fully described elsewhere in this chapter). The type of separation provided by the separator depends on whether the output is punch-destined or printer-destined.

Punch-Destined Output

The IBM-supplied separator provides three specially punched cards (deposited in stacker 1) prior to the punch card output of each job. Each of these separator cards is punched in the following format:

Columns	1 to 35	– blanks
Columns	36 to 43	– jobname
Columns	44 to 45	– blanks
Column	46	– output classname
Columns	47 to 80	– blanks

Printer-destined Output

The IBM-supplied separator provides three specially printed pages prior to printing the output of each job. Each of these three separator pages is printed in the following format:

- Beginning at the channel 1 location (normally near the top of the page), the jobname is printed in block character format over 12 consecutive lines. The first block character of the 8-character jobname begins in column 11. Each block character is separated by 2 blank columns.
- The next 2 lines are blank.
- The output classname is printed in block character format covering the next 12 lines. This is a 1-character name, and the block character begins in column 55.
- The remaining lines to the bottom of the page are blank.

In addition to the above, a full line of asterisks (*) is printed twice (overprinted) across the folds of the paper. These lines are printed on the fold preceding each of the three separator pages, and on the fold following the third page. This feature provides easy separation of job output in a stack of printed pages.

For printer-destined output with the IBM-supplied separator, a channel 9 punch should be included in addition to the channel 1 punch on the carriage control tape or in the forms control buffer (FCB). The channel 9 punch controls the location of the line of asterisks and should correspond to the bottom of the page. To print the line of asterisks on the fold of the pages, offset the printer registration should be offset.

Writing an Output Separator Program

The output separator program can be written by using the information provided by the output writer and by conforming to the requirements explained below. The separator program, when added to the link library (SYS1.LINKLIB) or the LPA library (SYS1.LPALIB), is invoked by specifying its name as a parameter in the EXEC statement of the output writer cataloged procedure.

Parameter List

The output writer provides the separator program with a 4-word parameter list of needed information. When the program receives control, register 1 contains the address of a 4-word parameter list, and the parameter list contains the following:

Bytes 0-3 –	In this word, byte 0 contains switches that indicate the type of output unit, and bytes 1-3 are reserved for future use.
Bytes 4-7 –	This word is the address of the output DCB (data control block).
Bytes 8-11 –	This word is the address of an 8-character field containing the jobname.
Bytes 12-15 –	This word is the address of a 1-character field containing the output classname.

In the parameter list, the three high-order bits of byte 0 are switches that the separator program uses to determine the type of output unit. The first bit to the left is set to 0. The second bit is set to 1 if the output unit is a punch device or a tape device with punch-destined output. The third bit is set to 1 if the output unit is a printer or punch device. The resulting bit combinations indicate the following:

011.	2520 or 2540 punch device
001.	1403, 1443, or 3211 printer device
010.	tape device with punch-destined output
000.	tape device with printer-destined output

The parameter list also points to the DCB for the output data set. This DCB is established for the queued sequential access method (QSAM), and is already open when the separator program receives control.

The address of the jobname and the address of the output classname are provided in the parameter list so that this information may be used in the separation records written by the separator program.

Programming Conventions

When using the system output writer, the separator program, if specified in the output writer cataloged procedure, is brought in by a LINK macro instruction issued from module IEFSD078 of the output writer. The separator program can be any size, but a program over 8K may affect the region requirement of the output writer.

Caution: Since the separator program operates with the supervisor protection key, but in the program mode, the separator program must insure data protection during its execution.

When writing a separator program, the following programming conventions must be observed:

- The program must conform to the standard linkage coventions. This includes saving and restoring the contents of registers 0 through 12, and 14. These registers can be preserved with the SAVE and RETURN macro instructions. When the program receives control, the address of a standard save area is in register 13.
- The program must use the PUT macro instruction in the locate mode to write separation records on the output data set. (This method is required by the QSAM DCB that is open for the output data set.)
- The program must establish its own synchronous error exit routine, and the address of this routine must be placed into the DCBSYNAD field of the output DCB. This gives control to the error exit routine in case an uncorrectable I/O error occurs while writing the program's output.
- The program should use the RETURN macro instruction to return control to the output writer. Before returning, the program must free any main storage it obtained during its operation; and the program must place a return code (binary) in register 15. The return codes signify:

  0 -- Successful operation.

  8 -- Unrecoverable output error (should be set if the error exit routine is entered).

Output from the Separator Program

The separator program can write any kind of separation identification. The jobname and the output classname for each job are available through the parameter list for inclusion in the output, if desired. An IBM-supplied routine can be used that constructs block characters (explained later). As many separator cards can be punched or as many separator pages can be printed as necessary.

The output from the separator program must conform to the attributes of the output data set. These attributes, which can be determined from the open output DCB pointed to by the parameter list, are:

- Record format (fixed, variable, or undefined length).
- Record length.
- Type of carriage control characters (machine, USASI, or none).

For printer-destined output, the separation records should be begun on the same page as the previous job output, or any subsequent page should be skipped to. However, the separator program should skip at least one line before writing any records, because in some cases the printer is still positioned on the line last printed.

After completing the output of the separation records, the separator program should write sufficient blank records to force out the last separation record. This also allows the error exit routine to obtain control if an uncorrectable output error occurs while writing the last record. The requirements are:

- One blank record for printer-destined output.
- Three blank records for punch-destined output.

Using the Block Character Routine

For printer-destined output, the separator program can use an IBM-supplied routine to construct separation records in a block character format. This routine is a reenterable module named IEFSD095, and resides in the module library (SYS1.AOSB0).

The block character routine constructs block letters (A to Z), block numbers (0 to 9), and a blank. The program furnishes the desired character string and the construction area. The block characters are constructed one line position at a time. Each complete character is contained in 12 lines and 12 columns; therefore, a block character area consists of 144 print positions. For each position, the routine provides either a space or the character itself.

The routine spaces 2 columns between each block character in the string. However, the routine does not enter blanks between or within the block characters. The program must prepare the construction area with blanks or other desired background before entering the block character routine.

To use the IBM-supplied block character routine, the separator program executes the CALL macro instruction with the entry point name of IEFSD095. Since the block characters are constructed one line position at a time, complete construction of a block character string requires 12 entries to the routine. Each time, provide the address of a 4-word parameter list should be provided in register 1. The parameter list must contain the following:

Bytes 0-3 –	This word is the address of a field containing the desired character string in EBCDIC format.
Bytes 4-7 –	This word is the address of a full word field containing the line count as a binary integer from 1 to 12. This represents the line position to be constructed on this call.
Bytes 8-11 –	This word is the address of a construction area in main storage where the routine will construct a line of the block character string. The required length in bytes of this construction area is 14n-2, where n represents the number of characters in the string.
Bytes 12-15 –	This word is the address of a fullword field containing, in binary, the number of characters in the string.

System Output Writer Routines

When a job is executing, system messages and data sets specifying the SYSOUT parameter (e.g., in the DD statement) are recorded on direct access devices. When the job completes, entries are made in system output class queues that represent the data sets and messages directed to the output classes. Later, system output writers remove these entries from the queues and process the data they represent. Processing consists of writing system messages to the output device and calling a data set writer routine for each data set encountered.

Th e data set writer routine used for a data set may be specified by name in a DD statement; otherwise, a standard IBM-supplied writer routine is used. The standard routine transcribes the data set to the specified output device, making only those data format and control character transformations required to conform to the attributes specified for the output data set.

The following material describes how to write a nonstandard data set writer routine.

Characteristics of an Output Writer

Before writing or modifying an output writer routine, the functions performed by the standard data set writer should be understood. In general, these functions include opening the data set (referred to as an input data set) that contains the processed information, obtaining the records of the data set, making any necessary transformations in record format or control character attributes, and placing these (possibly transformed) records in the output data set, which appears on a specified output device. The standard writer also must close the input data set and restore system conditions to the state they were in before the writer routine was invoked.

Writing an Output Writer Routine

To use the output writer routine, the name of the routine should be specified as a parameter in the SYSOUT operand of a DD statement. The routine must be in the link library (SYS1.LINKLIB) or the LPA library (SYS1.LPALIB). A writer routine is not limited in size except that size may influence the region requirements of the system output writer.

In VS2, the routine is attached (via the ATTACH macro instruction) when a data set requiring the routine is to be processed. The standard linkage conventions for attaching are used. Any storage required for work areas and tables should be obtained by the GETMAIN macro instruction and released by the FREEMAIN macro instruction. The output writer routines must be reenterable.

When the routine is finished, it must return control to the standard writer by using the RETURN macro instruction.

Parameter List

After job management routines perform initialization requirements and open the output data set into which the writer routine will put records, control is given to the routine via the ATTACH macro instruction. At this time, general registers 1 and 13 contain information that the program must use. Register 1 contains the storage address of a 12-byte list. The information in this parameter list follows:

	Output Device	Indicator.
Byte 0	Bit 0	(High-order bit): This bit should be off (set to 0).
	Bit 1	If this bit is on, the output unit is either a punch or a tape with a punch as the final destination.
	Bit 2	If this bit is on, the output unit is either a printer or a punch.
	Bits 3-7	No significant information.
Bytes 1-3	Not used, but must be present	
Bytes 4-7	This word contains the address of the data control block (DCB) for the opened output data set to be referred to by the writer.	
Bytes 8-11	This word contains the DCB address for the input data set from which your writer will obtain logical records. (At the time this 12-byte parameter list is given to the writer, the input data set is not open.)	

The switches indicated by the three high-order bit settings in byte 0 should be used to translate control character information from the input data set records to the form required by the output data set records. The high-order three bits of byte 0 signify the type of output device as follows:

011.....	2520 or 2540 punch unit
001.....	1403, 1443, or 3211 printer device
010.....	tape device with punch-destined output
000.....	tape device with printer-destined output

When the writer gets control, it must preserve the contents of register 0 through 12, and 14. Register 13 contains the address of a standard register save area that saves the contents of these registers. Save the contents of register 13 should be saved by using the SAVE macro instruction.

Programming Conventsions

An output writer routine must issue an OPEN macro instruction to open the desired input data set residing on a direct access device as a result of the previous execution of a processing program. (Note: The output data set used by a writer is opened by a job management routine before control is given to the writer. This output data set must be given records by a PUT macro instruction operating in the "locate" mode.

If the processing program that produces a given data set (to be used as an input data set by a writer) did not open the data set, the data set contains no records, and the DCBBLKSI and DCBBUFL fields of the input DCB contains zero. The DCBBLKSI field may also be zero even if the data set does contain records -- if the processing program did not put the block size value for the input data set in the DCB. If both these DCB fields are zero, a value (the standard writer uses the decimal value 18) is inserted in the DCBBLKSI field to permit the open routine to continue. The standard writer does this via a routine pointed to by an entry in the EXLIST parameter of the DCB. Since there is no data set, nothing is put on the output device. The data set writer must provide a SYNAD routine to process errors associated with the output as well as the input data set.

The standard data set writer also includes accounting support for the SMF output writer record (record type 6).

Before the OPEN macro instruction is issued, the DCBD macro instruction can be used to symbolically define the fields of the DCB, and the EXLIST and/or SYNAD routine addresses can be inserted. Other than SYNAD, no modifications can be made to the output DCB.

After the routine finishes writing the output data set, it must close the input data set and return using the RETURN macro instruction. A return code must be placed in register 15. This code should indicate that an unrecoverable output error either has occurred (code of 8) or has not occurred (code of 0).

3525 Note -- Interpret Punch: The programming support for the 3525 includes an INTERPRET PUNCH feature which is supported by BSAM and QSAM. The support for this feature includes the punching and printing of graphically printable punched characters on print lines one and three of the card. Line one includes the first 64 characters and line three includes the last 16 characters (right justified). Extraneous characters are printed for non-graphic eight-bit codes.

If the INTREPRET PUNCH function is designated via the new FUNC parameter in either a DCB or DD statement, an existing output data set will be interpreted as well as punched.

Note: The output must be 80 bytes, or 81 bytes if first character control is being used.

Processing Performed by the Output Writer

Figure 27 provides a general description of the procedures followed by the standard writer. When writing a writer routine, items can be deleted, modified, or added to some of these procedures, depending on the characterisitics of the data set(s). However, the procedures must be consistent with operating system conventions.

Saving Register Contents: Upon entering the writer program, the program must save the contents of the general registers, as previously discussed.

Obtaining Main Storage for Work Areas: In this work area, switches are established, record lengths and control characters are saved, and space is reserved for other uses. Storage is obtained by a GETMAIN macro instruction.

Processing Input Data Set(s): To process a data set, the writer must get each record individually from the input data set, transform (if necessary) the record format and the control characters associated with the the record in accordance with the output data set requirements, and put the record in the output data set. Data set processing by the standard writer can be considered in three aspects.

1. The first consideration is what must be done before actually obtaining records from an input data set. If the output device is a printer, provision must be made to handle the two forms of record control character that may accompany a record in an output data set. The printer is designed so that if the output data set records contain machine control characters, a record (line) is printed before the effect of its control character is considered. However, if USASI control characters are used in the output data set records, the control character effect is considered before the printer prints a record.

   Thus, if all the input data sets do not have the same type of control characters, it may be desirable to avoid overprinting of the last line of one data set with the first line of the following data set. If the records of the input data set have machine control characters (mcc) and the output data set records are to have USASI control characters (acc), the standard writer produces a control character that indicates one line should be skipped before printing the first line of output data.

   If the input data set records have acc and the output data set records are to be written with mcc, the standard writer prints a line of blanks before printing the first actual output data set record. Following this line of blanks, a one-line space is generated before the first output record is printed. The preceding "printer initialization" procedure (or a similar one based on the characteristics of the data sets) is recommended.

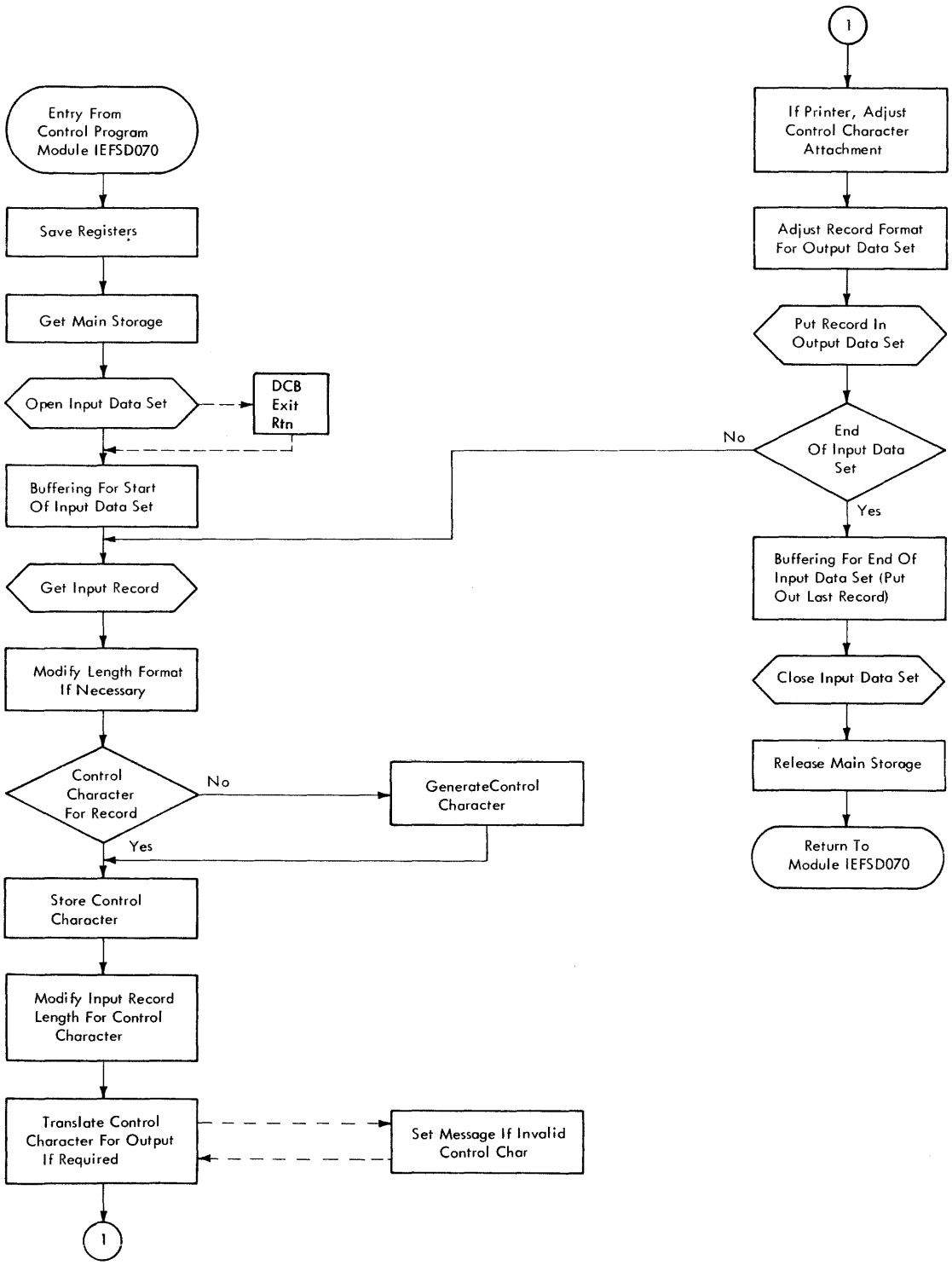


Figure 27. General Logic of Standard Output Writer

2. After an input data set is properly opened and any necessary printer initialization completed, the writer obtains records from the input data set. The locate mode of the GET macro instruction is used. As each record is obtained, its format and control character must be adjusted, if necessary, to agree with that required for output.

   Note: The MACRF field of the input data set DCB should be checked to see if GET in locate mode can be used. If not, the MACRF field must be overridden.

   Since the output data set is previously opened by another routine (job management), a writer routine must adhere to the established conventions. The data set is opened to receive records from the PUT macro instruction operating in the locate mode. For fixed-length record output, the length of the records in the output data set is obtained from the DCBLRECL field of the DCB. If an input record length is greater than the length specified for the records of the output data set, the standard writer truncates the necessary right-hand bytes of the input record. If the input record length is smaller than the output record length, the standard writer left-justifies the input record and adds blanks on the right end to give the correct length.

   When the output record length is variable and the input record length is fixed, the standard writer constructs each output record by adding control character information (if necessary) and variable record control information to the output record. The record control information is four bytes long and the control character information is one byte long. Both additions are made to the left end of the record. If the output record is not at least 18 bytes long, it is further modified by padding bytes (blanks) added to the right end of the record. If the output record length does not agree with the length of the output buffer, the standard writer makes the proper adjustment.

3. The third aspect is an end-of-input data set routine. The standard writer handles output to either a card punch unit or a printer unit, as required. Output to an intermediate device such as a tape unit is considered in light of the ultimate destination (e.g., punch or printer). If proper consideration is not given, all records from a given data set may not be available on the output device until the output of records from the next data set is started or until the output data set is closed. When the output data set is closed, the standard writer automatically puts out the last record of its last input data set.

Punch Output: Normally, when the standard writer is using a card punch as the output device, the last three output records are not in the collection pockets of the punch when the input data set is closed. To put out these three records with the rest of the data set and with no intervening pauses, the writer provides for three blank records following the actual data set records.

Printer Output: When the standard writer uses a printer as an output device, the last record of the input data set is not normally put in the output data set when the input data is closed. To force out this last record, the writer generates a blank record that follows the last record of the actual data set.

The problem of overprinting the last line of one data set by the first line of the following data set must also be considered. Depending on the combination of input record control character and required output record control character, a line of blanks and a spacing control character may be used either individually or in combination to preclude overprinting. (*Note:* If overprinting is desired for some reason, control characters in the data set records themselves may be used to override the effect (but not the action) of the previously described solutions to overprinting.)

Closing Input Data Set(s): After the standard writer finishes putting out the records of an input data set, it closes the data set before returning control to the system output writer. All input data sets must be closed.

Releasing Main Storage: The storage and buffer areas obtained for the writer must be released to the system before the writer relinquishes control. The FREEMAIN macro instruction should be used for this.

Restoring Register Contents: The original contents of general registers 0 through 12, and 14 must be restored. The RETURN macro instruction is used for this. To inform the operating system of the results of the processing done by the writer, a return code is placed in general register 15 before control is returned. If the writer routine terminates because of an unrecoverable error on the output data set, the return code is 8; otherwise, the return code is 0. Unrecoverable input errors must be handled by the data set writer.

Message Routing Exit Routines

This topic provides detailed information on how to write user exit routines that modify the routing and descriptor codes of WTO or WTOR messages for the VS2 operating system. Information is provided on inserting this exit routine into the resident portion of the control program. In addition, a description of the characteristics and configuration of MCS is supplied.

Characteristics of MCS

The multiple console support (MCS) facility routes messages to different functional areas according to the type of information that the message contains. In MCS, a functional area is defined as one or more operator's consoles that are doing the same type of work. (Some examples of functional areas are: (1) the tape pool area, (2) the disk pool area, and (3) the unit record pool area.) Each WTO and WTOR macro instruction is assigned one or more routing codes which are used to determine the destination of the message. There are fifteen routing codes that can be used. When the message is ready to be routed, the routing codes assigned to the message are compared to the routing codes assigned to each console. If any of the routing codes match, the message is sent to that console.

If the standard routing codes provided on application and system messages do not cover special situations at an installation, the routing codes can be modified by coding a user exit routine. The exit routine receives control prior to the routing of messages so users can examine the message text and modify the message's routing and descriptor codes. The system will use the modified routing codes to route the message. Descriptor codes provide a mechanism for message presentation and deletion, and are explained later in this chapter.

Automatic console switching occurs when permanent hardware errors are detected. Command-initiated console switching is provided to permit restructuring of the system console configuration and the hard copy log by system operators. Consoles can be moved into or out of functional areas at any time during system operation.

A hard copy log records messages, operator and system commands, and operator and system responses to commands. The hard copy log can be a console device or it can be the system log (SYSLOG). The number and type of messages recorded on the log is optional. The installation may wish to record a selected group of messages, or it may wish to record all messages. If commands are recorded, the system automatically records command responses.

Whenever possible, the hardcopy function should be delegated to an output-only device (such as a printer) or to the system log.

Programming Conventions For WTO/WTOR Routines

The programming conventions for the WTO/WTOR exit routine are summarized below. Details about many of the conventions are in the reference notes that follow; the notes are referred to by the numbers in the last column of the table.

Conventions	Requirements	Reference Code
Part of resident control program	Yes	
Size of routine	Any size	
Reenterable routine	Optional, but must be serially reusable	1
May not allow interruptions	Yes	2
Name of routine	Must be IEECVXIT	
Disposition of general registers	Registers must be saved at entry and restored prior to returning	
Format of text and codes	Provided through the DSECT IEECUCM	3
May issue WAIT, XCTL, WTO or WTOR macro instructions	No	
Method of abnormal termination	None	4
Exit from routine	RETURN macro instruction	

Reference Code **Reference Notes**

1 If the exit routine is to be reenterable, macro instructions should not be used whose expansions store information into an inline parameter list.

2 The exit routine should be written so that program interruptions cannot occur. If a program interruption occurs during execution of the exit routiine, the routine loses control and the communications task is terminated.

3 DSECT IEECUCM provides the format of the message text, routing codes and descriptor codes. The pointer in register 1 points to the first word of the message text, UCMMSTXT. The format is:

UCMMSTXT	Message flagging and text (128 characters-padded with blanks)
UCMROUTC	Routing codes (4 bytes)
UCMDESCD	Descriptor codes (4 bytes)

DSECT IEECUCM is contained in SYS1.MODGEN

System messages have a message code that may be examined to aid in identifying system messages, but it must not be modified.

The UCMROUTC field contains the routing codes. A bit setting of "1" indicates that the WTO or WTOR was assigned that particular routing code. Bit assignments and their meanings are:

Bit	Assignment	Meaning
Byte 0		
Bit 0	Routing code 1	Master Console Action
Bit 1	Routing code 2	Master Console Information
Bit 2	Routing code 3	Tape Pool
Bit 3	Routing code 4	Direct Access Pool
Bit 4	Routing code 5	Tape Library
Bit 5	Routing code 6	Disk Library
Bit 6	Routing code 7	Unit Record Pool
Bit 7	Routing code 8	Teleprocessing Control

Byte 1		
Bit 0	Routing code 9	System Security
Bit 1	Routing code 10	System Error/Maintenance
Bit 2	Routing code 11	Programmer Information
Bit 3	Routing code 12	Emulators
Bit 4	Routing code 13	Available for Customer Usage
Bit 5	Routing code 14	Available for Customer Usage
Bit 6	Routing code 15	Available for Customer Usage
Bit 7	Routing code 16	Reserved
Byte 2		Reserved
Byte 3		Reserved

The UCMDESCD field contains the descriptor codes. A bit setting of "1" indicates that the WTO or WTOR was assigned that particular descriptor code. Bit assignments and their meanings are:

Bit	Assignment	Meaning
Byte 0		
Bit 0	Descriptor code 1	System Failure
Bit 1	Descriptor code 2	Immediate Action Required
Bit 2	Descriptor code 3	Eventual Action Required
Bit 3	Descriptor code 4	System Status
Bit 4	Descriptor code 5	Immediate Command Response
Bit 5	Descriptor code 6	Job Status
Bit 6	Descriptor code 7	Application Program/Processor
Bit 7	Descriptor code 8	Out-of-Line Message
Byte 1		
Bit 0	Descriptor code 9	DISPLAY or MONITOR command response
	Descriptor codes 10 through 16	Reserved
Byte 2		Reserved
Byte 3		Reserved

4 The exit routine is part of the communications task. Abnormal termination of the exit routine causes the communications task to terminate abnormally.

Messages Not Using Routing Codes

There are certain messages that the exit routine does not see. These are messages that have the MSGTYP operand in the WTO or WTOR macro instruction coded with the JOBNAMES, STATUS, ACTIVE or Y parameter, multiple-line WTOs (including status displays) and messages that are being returned to the requesting console, i.e., a response to a DISPLAY A command. Routing of these messages is on criteria other than the routing codes; therefore, the system bypasses the exit routine.

Writing a WTO/WTOR Exit Routine

To modify the standard routing codes and descriptor codes, a WTO/WTOR Exit Routine must be written. This routine will be part of the control program. If a message's routing code field is used by the operating system to route the message, the routine will receive control prior to the routing of the message. When the routine receives control, register 1 contains a pointer to the first word of the message text. The message text field is 128 bytes long, followed by a four-byte routing code field and a four-byte descriptor code field. The exit routine may examine but not modify the message text.

A message will be sent to only those locations specified in the modified routing codes. All messages with modified routing codes are sent to the hard copy log when the log is included in the operating system. When the log is not included, the exit routine must not suppress

messages that contain a routing code of 1, 2, 3, 4, 7, 8, or 10 since messages with these codes are necessary for system maintenance. Message suppression is turning off all routing codes of a message, causing the message to be discarded. WTO messages can be suppressed. If a WTOR message is suppressed, it will be sent to the master console by the operating system.

Adding a WTO/WTOR Exit Routine to the Control Program

The WTO/WTOR exit routine is standard. If the user does not specify one, the IBM-supplied module (IEECVCTE) is included.

Task supervision must be performed for the exit routine every time a message is routed by its routing codes. If the communications task abnormally terminates, the linkage to the user exit routine is suppressed.

Inserting the WTO/WTOR Exit Routine

To enter the exit routine into the control program before system generation, the Linkage Editor should be used to replace the dummy WTO/WTOR exit routine IEECVCTE in SYS1.AOSC5 with the WTO/WOTR exit routine.

To enter the exit routine into the control program after system generation, the Linkage Editor should be used to replace the dummy WTO/WTOR exit routine IEECVCTE in the SYS1.NUCLEUS with the user-written WTO/WTOR exit routine. This will require a nucleus replace and a re-IPL.

STAE and STAI Exit and Retry Routines

Each STAE exit routine is represented by one or more STAE control blocks (SCBs). Each STAE control block is queued in a last-in, first-out order to the TCB (TCBNSTAE field) of the task within which they were created. STAI control blocks also represent exit routines, but are created when the STAI operand is specified in an ATTACH macro instruction. STAI control blocks are always placed at the top of the queue (ahead of the STAE control blocks) in a last-in, first-out order and are propagated (a duplicate STAI control block is created and queued) to all lower-level subtasks of the subtask created with the STAI operand.

Thus, if task A attached subtask B specifying the STAI operand, and subtask B attached subtask C which, in turn, attached subtask D, a STAI control block would be created and queued to the TCB for subtask B, and could be propagated to the queues originating at the TCBs for subtask C and subtask D. If a STAI control block were created for subtask C (the ATTACH macro instruction issued by subtask B specified the STAI operand), this STAI control block would be placed at the top of subtask C's SCB queue ahead of the STAI control block created for subtask B. In this case, both STAI control blocks would be propagated to the TCB for subtask D. All STAI control blocks precede all STAE control blocks on the SCB queue.

If a task is scheduled for abnormal termination, the exit routine specified by the most recently issued STAE macro instruction (represented by the highest STAE control block on the queue) is given control and executes under a program request block created by the SYNCH service routine. The STAE exit routine must specify, by a return code in register 15, whether a retry routine is to be scheduled. If no retry routine is to be scheduled (return code=0) and this is a subtask with a STAI control block on the SCB queue, the exit routine specified in the STAI control block is given control. If there is no STAI control block on the queue, abnormal termination continues.

If the STAE exit routine indicates that a retry routine has been provided (return code=4), register 0 must contain the address of the retry routine and register 1 must contain the address of the same work area passed to the exit routine. (The first word of the work area may be modified by the exit routine to point to another parameter list in his region.) The STAE control block is freed and the request block terminated up to, but not including, the RB of the program that issued the STAE macro instruction. This is done by pointing to an SVC 3 instruction in the old PSW field of each RB to be purged. In addition, open DCBs which can be associated with the purge RBs are closed and queued I/O requests associated with these DCBs being closed are deleted from the I/O restore chain.

The RB purge is an attempt to cancel the effects of partially executed programs that are at a lower level in the program hierarchy than the program under which the retry will occur. However, certain effects on the system will not be canceled by this RB purge. Example of these effects are as follows:

- Subtask created by a program to be purged.
- Resources allocated by the ENQ macro instructions.
- DCBs that exist in dynamically acquired virtual storage.

When the STAE exit routine gains control, it can examine the code in register 0 to determine if there were active input/output operations at the time of the ABEND and if the input/output operation are restorable. If there are quiesced restorable input/output operations, they can be restored, in the STAE retry routine, by using word 26 in the work area. Word 26 contains the link field passed as a parameter to SVC Restore. SVC Restore is used to have the system restore all I/O request on the I/O restore chain.

Users can selectively restore specific I/O requests on the I/O restore chain by using word 2 in the work area. Word 2 contains the address the first I/O block on the I/O restore chain. This address can be used as a starting point for issuing EXCP for the I/O requests that you want to restore.

In supervisor mode, users may want the failing task to remain in its present status and not be reestablished. A retry routine may be scheduled withough a purge of the RB chain by returning to the ABEND/STAE interface routine with an 8 in register 15, and register 0 and 1 initialized as described above. If the STAE retry routine is scheduled, the system automatically cancels the active SCB and the preceding SCB, if there is one, will become the active SCB. Users wanting to maintain within the retry routine must reestablish an active SCB within the retry routine, or must issue multiple STAE requests prior to the time that the retry routine gains control. Also, if a STAI had been issued for this task, it must be reissued by the retry routine to be made effective again.

A STAI exit routine, if specified in a previous ATTACH macro instruction, will receive control if a STAE exit routine is not specified, if a STAE exit routine is specified but indicates that a retry routine is not provided, if a STAE exit routine terminates abnormally, or if a STAE or a STAI retry routine abnormally terminates. The STAI exit routine must specify by a return code in register 15 one of the following:

Return Code	Action to be Taken
0	No retry provided. The next STAI exit routine is to be given control or, if there is not another STAI exit routine, abnormal termination is to continue.
16	No further STAI processing is to occur Abnormal termination processing is to continue.
4 or 12	A retry routine is to be scheduled and the request block queue is to be purged.
8	A retry routine is to be scheduled but the request block queue is not to be purged (if the user is not in supervisor mode, the return code will be ignored and abnormal termination processing continues).

When the RB queue is not to be purged, a new PRB is created for the retry routine and placed on the RB queue immediately after the SVRB for the ABEND routine, so that when the ABEND routine returns via an SVC 3 instruction the retry routine will receive control.

If the RB queue is to be purged, the STAI retry routine is executed under the PRB for the last STAE or STAI exit routine or, if no PRB for an exit routine exists on the queue, under the most recently created PRB that is pointed to by the oldest (first created) non-PRB on the queue (the oldest non-PRB will be the last RB purged).

Like the STAE/STAI exit routine, the STAE/STAI retry routine must be in storage when the exit routine determines that retry is to attempted. If not already resident within your program, the retry routine may be brought into storage via the LOAD macro instruction by either the user's program or exit routine.

Upon entry to the STAE/STAI retry routine, register contents are as follows:

Register 0:	0
Register 1:	Address of the work area, as previously described, except that word 2 now contains the address of the first I/O block and word 26 now contains the address of the I/O restore chain.
Register 2-13:	Unpredictable.
Register 14:	Address of an SVC 3 instruction.
Register 15:	Address of the STAE/STAI retry routine.

The retry routine should use the FREEMAIN macro instruction to free the 104 bytes of storage occupied by the work area when the storage is no longer needed. This storage should be freed from subpool 0 which is the default subpool for the FREEMAIN macro instruction.

Again, if the ABEND/STAE interface routine was not able to obtain storage for the work area, register 0 contains a 12; register 1, the ABEND completion code upon entry to the STAE retry routine; and register 2, the address of the first I/O Block on the restore chain, or 0 if I/O is not restorable.

Note: If the program using the STAE macro instruction terminates via the EXIT macro instruction, the EXIT routine cancels all SCBs related to the terminating program. If the program terminates via the XCTL macro instruction, the EXIT routine cancels all SCBs related to the terminating program except those SCBs that were created with XCTL=YES option. If the program terminates by any other means, the terminating program must reinstate the previous SCB by canceling all SCBs related to the terminating program.

SYS1.PARMLIB Data Set Lists

The SYS1.PARMLIB system data set contains the members used by the nucleus initialization program (NIP) during the initial program loading (IPL) process. Each member may be used in conjunction with operator action to dynamically tailor the system to individual installation requirements.

During system generation, the SYS1.PARMLIB data set is created and initialized with the basic member supplied by IBM. Members may be added or modified during later steps of system generation; for more information, see *OS/VS2 System Generation Reference,* GC26-3792.

When the system is generated, the SYS1.PARMLIB data set may be modified by use of the IEBUPDTE utility program.

Initialization

The nucleus initialization program searches the system catalog to locate the SYS1.PARMLIB data set. If it is not found in the catalog, SYS1.PARMLIB is assumed to reside on the IPL volume. If no VTOC entry can be found, the master console will receive the message "IEA2111 OBTAIN FAILED FOR SYS1.PARMLIB DATA SET" and the system will enter wait state with X'37'.

If the SYS1.PARMLIB data set is found, NIP opens SYS1.PARMLIB and processing continues.

Characteristics and Formats of SYS1.PARMLIB Members

The names and contents of the SYS1.PARMLIB lists are:

List Name	List Contents
IEABLD00 - standard list IEABLDxx - alternate list(s)	Names of SYS1.LINKLIB library load modules whose directory entries are to be entered in the BLDL table.
IEAFIX00 - standard list IEAFIXxx - alternate list (s)	Names of SYS1.SVCLIB, SYS1.LPALIB, and SYS1.LINKLIB modules which are to be loaded in real storage and fixed.
IEALOD00 - standard list	Names of SYS1.LPALIB modules whose directory control block information is to be maintained in nonpageable storage.
IEALPAxx - alternate list(s)	Names of SYS1.SVCLIB, SYS1.LPALIB, and SYS1.LINKLIB modules which should be included in the pageable LPA.
IEAPAK00 - standard list	Groups of names of SYS1.LPALIB modules which are to be packed together in the LPA.
IEASYS00 - standard list IEASYSxx - alternate list(s)	List of parameters that may be specified at "SPECIFY SYSTEM PARAMETERS" time of OS/VS2 nucleus initialization process.
LNKLST00 - standard list	Names of additional data sets that may be concatenated with SYS1.LINKLIB data sets.

All members of SYS1.PARMLIB have certain basic characteristics in common. Unless otherwise stated below, the input format to IEBUPDTE is of the following format:

- Record size is 80-byte card record images.
- Any columns between 1 and 71 may contain SYS1.PARMLIB data.
- Continuation is indicated by a comma after the last entry on a record, followed by a blank. Optionally, a non-blank character in column 72 may be used to show continuation.
- Leading blanks on all cards are suppressed, so that data on the records may be free from for the convenience of the user.

Resident BLDL Lists (IEABLD00 and IEABLDxx)

The resident BLDL list, IEABLDxx, specifies the names of modules from SYS1.LINKLIB or any data set concatenated to SYS1.LINKLIB which are to have directory entries built by NIP. This eliminates the directory search required when a load module is requested from SYS1.LINKLIB.

The directory created will reside either in the fixed or pageable area of the system, depending upon whether the BLDL or BLDL reply is specified by the operator in response to the "SPECIFY SYSTEM PARAMETERS" message. The BLDL and BLDLF parameters are mutually exclusive. The fixed BLDL list increases the *real* storage requirements of the system and should be used only for frequently used modules.

The member name for the standard list is IEABLD00. The load module names must be listed in the same order as they appear in the directory; that is, they must appear in ascending collating sequence. (Note: Directory entries in the resident BLDL table are not updated as a result of updating the load module in the library. The old version of the load module is used until an IPL process updates the resident BLDL table.)

If specified during system generation and not modified through operator communication or the SYS1.PARMLIB list of system parameters (IEASYS00), the default IEABLD00 is used.

Format: The format of IEABLDxx lists are as follows:

- The names of modules to be fixed are separated by commas;
- Continuation is indicated by a comma after the last module name on a record.

Fixed LPA Lists (IEAFIX00 and IEAFIXxx)

The fixed LPA list, IEAFIXxx, indicates those modules from SYS1.SVCLIB, SYS1.LPALIB, and/or SYS1.LINKLIB which should be included in the nonpageable extension to the link pack area. The list should contain a minumum number of modules which are required or may result in a significantly improved system performance when the modules are fixed rather than paged.

The IBM-supplied list (IEAFIX00) contains two modules that should always be part of the fixed LPA. These modules are required to handle direct access volume serial number verification for the volume containing the page data set.

Format: The format of the IEAFIXxx lists are as follows:

- The library name (i.e., SYS1.SVCLIB) must be represented in the initial record of the PARMLIB list, followed by at least one blank character.
- The string of module names which are to be loaded from that library follow the blank character on the record.
- Additional records do not require the library name unless the source library for the modules is to be changed.
- Module names are separated by commas.
- The last module name for a library must be followed by a comma if the list contains another library name. The last module name for the last library should not be followed by a comma.

Modified LPA List (IEALPAxx)

The modified LPA list, IEALPAxx, indicates those modules from SYS1.SVCLIB, SYS1.LPALIB, and SYS1.LINKLIB which should be temporarily included in the pageable link pack area. IEALPA00 may include the names of modules that reside on the SYS1.LPALIB data set; if this is done, the modules built in the modified LPA will beused in preference to the existing LPA modules.

Format: The format of the IEALPAxx lists are as follows:

- The library name (i.e., SYS1.SVCLIB) must be represented in the initial record of the PARMLIB list, followed by at least one blank character.
- The string of module names which are to be loaded from that library follow the blank character on the record.
- Additional records do not require the library name unless the source library for the module is to be changed.
- Module names are separated by commas.
- Continuation is indicated by a comma after the last module name on a record.

System Parameter Lists (IEASYS00 and IEASYSxx)

The system parameter list, IEASYSxx, may include all system parameters which are valid input to the "SPECIFY SYSTEM PARAMETERS" message. The standard list, IEASYS00, is created during Stage 2 of the OS/VS2 system generation process. This parameter list enables the user to specify all parameters needed by a particular IPL process without specifying them from an operator's console.

One of the additional aspects of this feature is that operator intervention may be restricted by use of the 'OPI-YES/NO' parameter. Operator override of an individual parameterlist may be restricted via the command.

Format: The format of the IEASYSxx lists are as follows:

- Parameters must be separated by commas; blanks between parameters are not acceptable.
- The standard message reply header (i.e., R 00,) required by the "SPECIFY SYSTEM PARAMETERS" message must not be present in the list.
- The set of parameters must not be enclosed in quotes.
- Continuation is indicated by a comma after the last parameter or subparameter on a record.

LPA Packing List (IEAPAK00)

The LPA packing list; IEAPAK00, specifies the names of modules residing on the SYS1.LPALIB data set that may be grouped together by packing the LPA. The placement of these modules may sharply reduce page faults as well as eliminate wasted space within the LPA.

The total size of all modules in a group must not exceed 4K (page size). Also, the modules should show an affinity for other modules of a group via system calls.

The proper loading of modules into the LPA is done at NIP time, based upon the data contained in IEAPAK00.

There are no alternate lists for IEAPAK00. If IEAPAK00 is not found as a member of SYS1.PARMLIB, NIP will continue processing.

Format: The format of the IEAPAK00 list is as follows:

- Each entry of the list is a group of module names enclosed in parantheses of the format (A,B,...,C)...,(A,B,...,C).
- Groups and module names within groups are separated by commas.
- Continuation after a group is indicated by a comma following the closing parenthesis.
- Continuation within a group is indicated by a comma following the module name within a group. The group must be ended by a parenthesis on the continuation record.
- Aliases for a module cannot be an entry within a group. Only major names are processed by NIP.

LPA Directory Load List (IEALOD00)

The LPA directory load list, IEALOD00, indicates those modules residing on SYS1.LPALIB or in the pageable LPA which are to have the contents supervision directory control block information initialized by NIP in nonpageable storage. This list improves performance since contents supervision does not have to search to find the module in the LPA directory, thus reducing the number of page faults that may occur. Candidates for this list are modules that are frequently used.

Format: The format of the IEALOD00 list is as follows:

- The names of modules to be loaded from SYS1.LPALIB are separated by commas.
- Continuation is indicated by a comma after the last module name on a record.

Link Library List (LNKLST00)

The link library list, LNKLST00, enables concatenating up to 16 data sets, on multiple volumes, to form SYS1.LINKLIB. LNKLST00 is not generated at system generation but may be added to SYS1.PARMLIB via the IEUPDTE utility program. Each data set that is concatenated with SYS1.LINKLIB may have up to 16 extents. (Note: After additions or changes are made to data sets concatenated with SYS1.LINKLIB, the system should be re-IPLed to update the description of the SYS1.LINKLIB to the system.)

Format: The format of the link library list, LNKLST00, is as follows:

- The string of data set names that are to be concatenated with SYS1.LINKLIB follw the name SYS1.LINKLIB, separated by commas.
- Continauation is indicated by a comma after the last name on a record.
- The maximum number of data set names other than SYS1.LINKLIB is 15. This allows 16 data sets including SYS1.LINKLIB to be concatenated.

Adding the Lists to SYS1.PARMLIB

To place these lists in SYS1.PARMLIB, the IEBUPDTE utility program may be used as shown below:

```
//ADDLISTS          JOB     61938,R.L. WILSON
//STEP              EXEC    PGM=IEBUPDTE,PARM=MOD
//SYSPRINT          DD      SYSOUT=A
//SYSUT2            DD      DSNAME=SYS1.PARMLIB,DISP=OLD
//SYSIN             DD      DATA
./                  ADD NAME=IEABLD00,LIST=ALL

./                  NUMBER  NEW1=01,INCR=02
SYS1.LINKLIB        IEFSD061,IEFSD062,IEFSD064,IEFSD104,
                    IEFVM1,IEFWC000,IEFWD000,IEFW21SD,
                    IEFW41SD,IEFW42SD,IEFXJ000
./                  REPL    NAME=IEALOD00,LEVEL=01,SOURCE=1,
./                  LIST=ALL
                    NUMBER NEW1=10,INCR=100
                    IEAFAB400, IGG0325A,IGG0325H,IGC0003B,
                    IGG0325B,IGG0325D,IGG0325E,
                    IGG0325G,IFG0202J,IFG0202K,IFG0202L
./                  REPL NAME=IEAPAK00,LEVEL=01,SOURCE=1,LIST=ALL
./                  NUMBER NEW1=01,INCR=02
                    (IEFAB400,IGG0325A,IGG0325H,IGC0003B),
                    (IGG0325B,IGG0325D,IGG0325E,
                    IGG0235G),(IFG0202J,IFG0202K,IFG0202L)
./                  ADD NAME=IEASYS05,LIST=ALL
./                  NUMBER NEW1=01,INCR=02
                    MLPA=(00,01),TRACE=(50,OPI=NO),
                    BLDL=00,SQA=2
./                  ADD     NAME=IEASYS06,LIST=ALL
./                  NUMBER NEW=01,INCR=02
                    MLPA=(02,03),BLDLF=(00,01),
                    TRACE=100,SQA=1,CPQE=10,
                    FIX=(00,01),OPI=NO
./                  ENDUP
/*
```

Shared Direct Access Storage Devices (Shared DASD)

The Shared DASD facility allows computing systems to share direct access storage devices. Systems can share common data and consolidate data when necessary; no change to existing records, data sets, or volumes is necessary to use the facility. However, reorganization of volumes may be desirable to achieve better performance.

Devices that Can be Shared

The following control units and devices are supported by the Shared DASD option:

- IBM 2314 Direct Access Storage Facility equipped with the two-channel switch -- IBM 2314 disk Storage Module.
- IBM 2314 Direct Access Storage Facility combined with the IBM 2844 Auxilliary Storage Control -- IBM Disk Storage Module. Device reservation and release are supported by this combination with or without the presence of the two-channel switch. Two channels -- one from System A and one from System B -- may be connected to the combination. In addition, the two-channel switch may be installed in either or both of the control units, thus permitting as many as four systems to share the devices.
- IBM 2835 Storage Control Unit with two-channel switch -- IBM 2305 Fixed Head Storage Facility.
- IBM 3830 Storage Control Unit with two-channel switch -- IBM 3330 Series Disk Storage Drive.

Alternate channels to a device from any one system may only be specified for the IBM 2314 Direct Access Storage Facility, or the IBM 3330 Series Storage Unit.

Volume/Device Status

The Shared DASD facility requires that certain combinations of volume characteristics and device status be in effect for shared volumes of devices. One of the following combinations must be in effect for a volume or device:

System A	Systems B, C, D
Permanently resident	Permanently Resident
Reserved	Reserved
Removable	Offline
Offline	Removable or reserved

If a volume/device is marked removable on any one system, the device must be in offline status on all other systems. The mount characteristic of a volume and/or device status may be changed on one system as long as the resulting combination is valid for other systems sharing the device. No other combination of volume characteristics and device status is supported.

System Configuration

Operating system configurations do not have to be identical to share a data set. The only additional equipment needed for the Shared DASD option is either a two-channel switch or a 2844 Auxilliary Control unit. The user must also observe certain restrictions about the data sets that are shared. The following data sets cannot be shared:

PASSWORD	SYS1.PAGE
SYS1.LOGREC	SYS1.SVCLIB
SYS1.LPALIB	SYS1.SYSJOBQE
SYS1.MANX	SYS1.SYSVLOGX
SYS1.MANY	SYS1.SYSVLOGY
SYS1.NUCLEUS	SYSCTLG

Volume Handling

Volume handling on the Shared DASD option must be clearly defined since operator actions on the sharing system must be performed in parallel. The following rules should be in effect when using the Shared DASD option:

- Operators should initiate all shared volume mounting and demounting operations. The system will dynamically allocate devices unless they are in reserved or permanently resident status, and only the former can be changed by the operator.
- Mounting and demounting operations must be done in parallel on all sharing systems. A VARY OFFLINE must be effected on all systems before a device may be dismounted.
- Valid combinations of volume mount characteristics and device status for all sharing systems must be maintained. To IPL a system, a valid combination must be established before device allocation can proceed. This valid combination is established either by specifying mount characteristics of shared devices in PRESRES, or varying all sharable devices off line prior to issuing START commands and then following parallel mount procedures described in *Operator's Library: OS/VS2 Reference*, GC28-0210.

Note: The Set-Must-Complete (SMC) parameter available with the ENQ macro instruction may also be used with RESERVE.

Note: If a restart occurs when a RESERVE is in effect for devices, the system will not restore the RESERVE; the user's program must reissue the RESERVE.

Macro Instructions Used with Shared DASD

The RESERVE macro instruction is used to reserve a device for use by a particular system; it must be issued by each task needing device reservation. The EXTRACT macro instruction is used to obtain the address of the task input/output table (TIOT) from which the UCB address can be obtained. The topic "Finding the UCB Address" explains this procedure. The macro instructions are described in the chapter "Supervisor Macro Instructions for System Programmers."

Releasing Devices

The DEQ macro instruction is used in conjunction with RESERVE just as it is used with ENQ. It must describe the same resource and its scope must be stated as SYSTEMS; however, the UCB=pointer address parameter is not required. If the DEQ macro instruction is not issued by a task which has previously reserved a device, the system will free the device when the task is terminated.

Preventing Interlocks

Certain precautions must be taken to avoid system interlocks when the RESERVE macro instruction is used. The more often device reservations occur in each sharing system, the greater the chance of interlocks occurring. Allowing each task to reserve only one device minimizes the exposure to interlock. The system cannot detect interlocks caused by program use of the RESERVE macro instruction and enabled wait states will occur on the system or systems.

Volume Assignment

Since exclusive control is by device, not by data set, consider which data sets reside on the same volume. In this environment it is quite possible for two tasks in two different systems -- processing four different data sets on two shared volumes -- to become interlocked. For example, data sets A and B reside on device C, and data sets D and E reside on device F. Task X in system X reserves device C in order to use data set A; task Y in system Y tries to reserve device F in order to use data set D. Now task X in system X tries to reserve device F

in order to use data set E and task Y in system Y tries to reserve device C in order to use data set B. Neither can ever regain control, and neither will complete normally. When the system has job step time limits, the task, or tasks, in the interlock would be abnormally terminated when the time limit expires. Moreover, an interlock could mushroom, encompassing new tasks as these tasks try to reserve the devices involved in the existing interlock.

Program Libraries

When assigning program libraries to shared volumes, precaution must be taken to avoid interlock. For example, SVCLIB for system A resides on volume X, while SVCLIB for system B resides on volume Y. Task A in system A invokes a direct access device space management function for volume Y, resulting in that device being reserved. Task B in system B invokes a similar function for volume X, reserving that device. However, each load module transfers to another load module via XCTL. Since the SVCLIB for each system resides on a volume reserved by the other system, the XCTL marco instruction cannot complete the operation, therfore an interlock occurs in this particular case, since no access to SVCLIB is possible, both systems will eventually enter an enabled wait state.

Finding the UCB Address

This explains procedures for finding the UCB address for use by the RESERVE macro instruction; it also shows a sample assembler language subroutine which issues the RESERVE and DEQ macro instructions and can be called by higher level languages.

Providing the Unit Control Block Address to RESERVE: The EXTRACT macro instruction is used to obtain information from the task control block (TCB). The address of the TIOT can be obtained from the TCB in response to an EXTRACT. Prior to issuing an EXTRACT macro instruction, the user sets up an answer area in main storage which is to receive the requested information. One full word is required for each item to be provided by the control program. If the user wishes to obtain the TIOT address, he must specify FIELDS=TIOT in the EXTRACT macro instruction.

The address of the TIOT is then returned by the control program, right adjusted, in the full work answer area.

The TIOT is constructed by job management routines and resides in main storage during step execution. The TIOT consists of one or more DD entries, each of which represents a data set defined by a DD statement for the jobstep. Each entry includes the DD name. Associated with each DD entry is the UCB address of the associated device. In order to find the UCB address, the user must locate the DD entry in the TIOT corresponding to the DD name of the data set for whcih he intends to issue the RESERVE macro instruction.

The UCB address can be obtained via the DEB and the DCB. The data control block (DCB) is the block within which data pertinent to the current use of the data set is stored. The address of the data extent block (DEB) is contained at offset 44 decimal after the DCB has been opened. The DEB contains an extension of the information in the DCB. Each DEB is associated with a DCB, and the two point to each other.

The DEB contains information concerning the physical characteristics of the data set and other information that is used by the control program. A device dependent section for each extent is included as part of the DEB. Each such extent entry contains the UCB address of the device to which that portion of the data set has been allocated. In order to find the UCB address, the user must locate the extent entry in the DEB for which he intends to issue the RESERVE macro instruction. (In disk addresses of the form MBBCCHHR, the M indicates the extent number starting with 0).

Procedures for Finding the UCB Address of a Reserved Device: If the data set is a multivolume sequential data set, it must be assumed that all jobs will process that data set in a sequential manner starting with the first volume of the data set. In this case, by issuing a RESERVE for the first volume only, the user effectively reserves all the volumes of the data set.

For data sets using the queued access methods in the update mode or for unopened data sets:

1. Extract the TIOT from the TCB.
2. Search the TIOT for the DD name associated with the shared data set.
3. Add 16 to the address of the DD entry found in step 2. This results in a pointer to the UCB address obtained in step the TIOT.
4. Issue the RESERVE macro specifying the address obtained in step 3 as the operand of the UCB keyword.

For opened data sets:

1. Load the DEB address from the DCB field labeled DCBDEBAD.
2. Load the address of the the field labeled DEBDVMOD in the DEB obtained in step 1. The result is a pointer to the UCB address in the DEB.
3. Issue the RESERVE macro specifying the address obtained in step 2 as the operand of the UCB keyword.

For BDAM data sets the user may reserve the device at any point in the processing in the following manner:

1. Open the data set successfully.
2. Convert the block address used in the READ/WRITE macro to an actual device address of the form MBBCCHHR.
3. Load the DEB address from the DCB field labeled DCBDEBAD.
4. Load the address of the field labeled DEBDVMOD in the DEB.
5. Multiply the "M" of the direct access address by 16.
6. The sum of steps 4 and 5 is the address of the correct extent entry in the DEB for the next READ/WRITE operation. The sum is also a pointer to the UCB address for this extent.
7. Issue the RESERVE macro specifying the address obtained in step 6 as the operand of the UCB keyword.

If the data set is an ISAM data set, QISAM in the load mode should by used only at system update time. Further, if it is a multivolume ISAM data set, it must be assumed that all jobs will access the data set through the highest level index. The indexes should never reside in main storage when the data set is being shared. In this case, by issuing a RESERVE macro for the volume on which the highest level index resides, the user effectively reserves the volumes on which the prome data and independent overflow areas reside. The following procedures can by used to achieve this:

1. Open the data set successfully.
2. Locate the actual device address (MBBCCHHR) of the highest level index. This address can be obtained from the DCB.
3. Load the DEB address from the DCB field labeled DCBDEBAD.
4. Load the address of the field labeled DEBDVMOD in the DEB.
5. Multiply the "M" of the actual device address located in step 2 by 16.
6. The sum of steps 4 and 5 is the address of the correct extent entry in the DEB for the next READ/WRITE operation. The sum is also a pointer to the UCB address for this extent.
7. Issue the RESERVE macro specifying the address obtained in step 6 as the operand of the UCB keyword.

RES and DEQ Subroutines: The following assembler language subroutine can be used by assembler language programs to issue the RESERVE and DEQ macro instructions. Parameters that must by passed to the RESDEQ routine, if the RESERVE macro instruction is to be issued, are:

DDNAME - the eight character name of the DDCARD for the device to be reserved.

QNAME - an eight character name.

RNAME LENGTH - one byte (a binary integer) that contains the RNAME length value.

RNAME - a name from 1 to 255 charcters in length.

The DEQ macro instruction does not require the UCB=pointer address as a parameter. If the DEQ macro is to be issued, a fullword of binary zeros must be placed in the DDNAME field before control is passed.

```
RESDEQ   CSECT

         SAVE    (14,12),T       SAVE REGISTERS
         BALR    2,0             SET UP ADDRESSABILITY
         USING   *,2
         ST      13,SAVE+4
         LA      11,SAVE         ADDRESS OF MY SAVE AREA IS STORED
         ST      11,8(13)        IN THIRD WORD OF CALLER'S SAVE AREA
         LR      13,11           ADDRESS OF MY SAVE AREA
         LR      9,1             ADDRESS OF PARAMETER LIST
         L       3,0(9)          DDNAME PARAMETER OR WORD OF ZEROS
         CLC     0(4,3),=F'0'    WORD OF ZEROS IF DEQ IS REQUESTED
         BE      WANTDEQ
*PROCESS FOR DETERMINING THE UCB ADDRESS USING THE TIOT
         XR      11,11           REGISTER USED FOR DD ENTRY
         EXTRACT ADDRTIOT,FIELDS=TIOT
         L       7,ADDRTIOT      ADDRESS OF TASK I/O TABLE
         LA      7,24(7)         ADDRESS OF FIRST DD ENTRY
NEXTDD   CLC     0(8,3),4(7)     COMPARE DDNAMES
         BE      FINDUCB
         IC      11,0(7)         LENGTH OF DD ENTRY
         LA      7,0(7,11)       ADDRESS OF NEXT DD ENTRY
         CLC     0(4,7),=F'0'    CHECK FOR END OF TIOT
         BNE     NEXTDD
         ABEND   200,DUMP        DDNAME IS NOT IN TIOT, ERROR
FINDUCB  LA      8,16(7)         ADDRESS OF WORD IN TIOT THAT
*                                CONTAINS ADDRESS OF UCB
*PROCESS FOR DETERMINING THE QNAME REQUESTED
WANTDEQ  L       7,4(9)          ADDRESS OF QNAME LENGTH
         MVC     QNAME(8),0(7)   MOVE IN QNAME
*PROCESS FOR DETERMINING THE RNAME AND THE LENGTH OF RNAME
         L       7,8(9)          ADDRESS OF RNAME LENGTH
         MVC     RNLEN+3(1),0(7) MOVE BYTE CONTAINING LENGTH
         L       7,RNLEN
         STC     7,RNAME         STORE LENGTH OF RNAME IN THE
*                                 FIRST BYTE OF RNAME PARAMETER
*                                 FOR RES/DEQ MACROS
         L       6,12(9)         ADDRESS OF RNAME REQUESTED
         BCTR    7,0             SUBTRACT ONE FROM RNAME LENGTH
         EX      7,MOVERNAM      MOVE IN RNAME
         CLC     0(4,3),=F'0'
         BE      ISSUEDEQ
         RESERVE (QNAME,RNAME,E,0,SYSTEMS),UCB=(8)
         B       RETURN
ISSUEDEQ DEQ     (QNAME,RNAME,0,SYSTEMS)
RETURN   L       13,SAVE+4   RESTORE REGISTERS AND RETURN
         RETURN (14,12),T
         BCR     15,14
MOVERNAM MVC     RNAME+1(0),0(6)
ADDRTIOT DC      F'0'
SAVE     DS      18F
QNAME    DS      2F
RNAME    DS      CL256
RNLEN    DC      F'0'
         EN
D
```

The Must Complete Function

System routines (routines operating under a storage key of zero) often engage in updating and/or manipulation of system resources such as system data sets, control blocks, and queues. These resources contain information critical to continued operation of the system and they must complete their operations on the resource. Otherwise, the resource may be left

incomplete or may contain erroneous information -- either condition leads to unpredictable results.

The ENQ service routine provides the must complete function and ensures that a routine queued on a critical resource(s) can complete processing of the resource(s) without interruptions leading to termination. The must complete function places other tasks in a wait state until the requesting task -- the task issuing a ENQ macro instruction with the set-must-complete (SMC) operand -- has completed its operations on the resource. The requesting task releases the resource and terminates the must complete condition by issuing a DEQ macro instruction with the reset-must-complete (RMC) operand. The ENQ and DEQ macro instructions are described in the chapter "Supervisor Macro Instructions for System Programmers".

For the time it is in effect, the must complete function serializes operations to some extent in the computing system. Therefore, its use should be minimized -- use the function only in a routine that processes system data whose validity must be ensured.

As an example, in multitask environments, the integrity of the volume table of contents (VTOC) must be preserved during an updating process so that all future users may have access to the latest, correct, version of the VTOC. Thus, in this case, enqueue on the VTOC and use the must complete function (to suspend processing of other tasks) when updating a VTOC.

Just as the ENQ function serializes use of a resource requested by many different tasks, the must complete function serialize execution of tasks.

Characteristics of the Must Complete Function

The must complete function can be applied at two levels:

The System Level: Only the current task, and system tasks can execute. All other tasks in the system are placed in a wait state.

The Step Level: In a region only the current task is allowed to execute. All other tasks in the jobstep, including the initiator task, are placed in a wait state.

CAUTION: Use of the must complete function at the system level should not be attempted until all aternatives have been exhausted. Except for extremely unusual conditions the system level of must complete should never be used.

When the must complete function is requested the requesting task is marked as being in the must complete mode and all asynchronous exits from the requesting task are deferred. Other tasks in the system (except the allowed tasks at the system level) or associated with the requesting task in a job step (step level) are placed in a wait state. Thus, tasks external to the requesting task are prevented from initiating procedures that will cause termination of the requesting task. Other external events, such as a CANCEL command issued by an operator, or a job step timer expiration are also prevented from terminating the requesting task.

The must complete mode of operation is not entered until the resource(s) queued upon are available.

The failure of a jobstep at the step level results in the abnormal termination for the jobstep and any associated tasks or resources. The programmer recieves a message stating that the failure occured in the step must complete status.

The failure of a jobstep at the system level is handled differently. The user can attempt to retry the operation, using STAE/STAI exit retry routines, for example. However, should the retry fail, the system will abnormally terminate the jobstep, and perform the following operations:

- Purges page-in requests for the jobstep.
- Purges I/O requests.
- Purges asynchronous exit queues.

- Provides a diagnostic dump.
- Purges any associated resources.
- Makes any system must complete resources permanently unavailabel.
- Allows resources dependent on system must complete resources to continue processing, if possible.
- Issues a message to the operator.
- Sets all tasks below the master scheduler non-dispatchable.
- Sets all tasks associated with the failing task non-dispatchable.
- Passes control to the dispatcher.

The operator should notify the system programmer, who can run diagnostics, such as DSS, to attempt recovery, or who can IPL the system.

Programming Notes

1. All data used by a routine that is to operate in the must complete mode should be checked for validity to ensure against a program-check interruption.
2. A routine that is already in the must complete mode may call another routine which also operates in the must complete mode. An internal count is maintained of the number of SMC requests; an equivalent number of RMC requests is required to reset the must complete function.
3. Interlock conditions can arise with the use of the ENQ function. Additionally, an interlock may occur if a routine issues an ENQ macro instruction while in the must complete mode. The wanted resource may already be queued on by a task placed in the wait state due to the must complete request already made. Since the resource cannot be released, job step tasks are abnormally terminated, and system tasks are retried.
4. The macro instructions ATTACH, LINK, LOAD, and XCTL should not be used, *unless extreme care is taken* , by a routine operating in the must complete mode. An interlock condition will result if a serially-reusable routine requested by one of these macro instructions has been requested by one of the tasks made non-dispatchable by the use of the SMC operand or was requested by another task and has been only partially fetched.

   For example, suppose routine "b" in task B has requested and is using subroutine "c". Subsequently routine "a" in task A (of a higher priority than task B) receives control of the processing before routine "b" finishes with subroutine "c". If routine "a" issues an ENQ macro instruction with the SMC operand and puts task B (and, thus, routine "b") in a non-dispatchable condition, subroutine "c" remains assigned to routine "b". Now, if routine "a" issues a request (via a LINK, LOAD, etc. macro instruction) for subroutine "c", an interlock will occur between tasks A and B: task A cannot continue since subroutine "c" is still assigned to task B, and task B cannot continue (and thus release subroutine "c") because task A in the must complete mode has made task B nondispatchable.
5. The time a routine is in the must complete mode should be kept as short as possible -- enter at the last moment and leave as soon as possible. One suggested way is to:

   a. ENQ (on desired resource(s))

   b. ENQ (on same resource(s)),RET=HAVE,SMC=$\left\{\begin{matrix}\text{SYSTEM}\\\text{STEP}\end{matrix}\right\}$

   Item a gets the resource(s) without putting the routine into the must complete mode.

   Later, when appropriate, issue the ENQ with the must complete request (Item b). Issue a DEQ macro instruction to terminate the must complete mode as soon as processing is finished.
6. The macro instruction STATUS changes the dispatchability staus of tasks for users with a protection key of 0. STATUS can also change the must complete status of a task.

Tasks placed in the wait state by the corresponding ENQ macro instruction are made dispatchable and asynchronous exits from the requesting task are enabled.

Program Properties Table

The program properties table is a nonexecutable module (IEFSDPPT) that is used to assign special properties to programs that must execute in a privileged mode. When a program is executed, the table is scanned to determine if any of the special properties defined in the table apply to the program.

Each entry in the table consists of ten bytes:

- The first eight bytes contain the program name. The program name is the name specified in the PGM parameter of the first EXEC statement of the job.
- The last two bytes indicate the properties assigned to that program. Bit 0 of the properties field is used to indicate that the job cannot be canceled. Bit 1 is used to indicate that a unique protection key is to be assigned to the job. The remainder of the field is reserved for the inclusion of more properties.

The program properties table contains five dummy entries. The user can place program names in these entries via the AMASPZAP service aid program. If more than five programs are to be added to the table, the module must be updated, and then reassembled and relink edited.

Authorized Program Facility (APF)

The authorized program facility (APF) limits the use of sensitive system and (optionally) user services and resources to authorized system and user programs. The authorization consists of a code that is specified when the program is link edited. For a program to be authorized, it must be link edited with this code and reside in either the SYS1.LINKLIB or SYS1.SVCLIB data sets, or the link pack area. The SYS1.LINKLIB, SYS1.SVCLIB, and SYS1.LPALIB data sets are the only data sets in which an authorized program can reside.

Authorization is at the job step level -- that is, the authorization of the first program executed in the job step determines the authorization of the job step. Therefore, if the first program executed has been link edited as authorized and resides in the SYS1.LINKLIB or SYS1.SVCLIB data sets or the link pack area, the job step will be authorized. If the authorized job step invokes (via the LINK, LOAD, ATTACH, or XCTL macro instruction) any program (authorized or not) residing in the SYS1.LINKLIB or SYS1.SVCLIB data sets or the link pack area, authorization is retained. However, if the authorized job step invokes any program that does not reside in the SYS1.LINKLIB or SYS1.SVCLIB data sets or the link pack area, the authorization is lost for the remainder of the job step. Also, once a job step becomes unauthorized, it cannot be reinitialized as authorized.

The system services and resources that are restricted in use include SVC 28 (CVOL), SVC 59 (OLTEP), SVC 82 (DASDR), SVC 85 (DDR), SVC 107 (MODESET), and SVC 113 (the new PGFREE, PGFIX, and PGLOAD system paging macro instructions). The system programs that are authorized under APF to use the restricted SVCs are the IEHDASDR, IEHATLAS, and IEHPROGM utility programs, and the AMASPZAP service aid program. All programs executing in the supervisor state or under protection key zero are authorized by the system for the use of all SVCs.

The user programs that must be authorized include those programs that call the authorized system programs and those programs that use the restricted SVCs directly. In addition, programs must be authorized to open a VTOC for updating, to delete operator messages other than one which the programs originated, and to have their messages identified as system rather than user messages. (Any problem programs executing in Release 21 of MVT that use these restricted SVCs or authorized programs must be relink edited and moved to the proper

libraries to become authorized under APF. During system generation, the user can designate the user SVCs that are to be authorized.)

A new system macro instruction, TESTAUTH, has been defined to support APF. The macro instruction tests the authorization of the caller and informs the caller if it is authorized to perform a particular function. For a complete description of the TESTAUTH macro instruction, see the chapter "Supervisor Macro Instructions for System Programmers".

Linkage Editor Authorization

The linkage editor permits an installation to establish authorization of the programs either through a new parameter in the linkage edit step or through a new linkage editor control statement.

To assign an authorization code via the new parameter, AC (1) should be coded in the PARM field, as follows:

```
//LKED  EXEC    PGM=HEWL,PARM=AC(1),...
```

If no authorization code is assigned in the linkage editor job step, the default is zero. The authorization code for a given output module can be overridden with the SETCODE statement.

If the SETCODE statement is used to establish authorization, it must be placed before the NAME statement for the output load module. The format of the SETCODE statement is:

Operation	Operand
SETCODE	AC(1)

The SETCODE statement will override an authorization code assigned in the PARM field of the EXEC statement. If more than one SETCODE statement is assigned to a given output load module, the last statement found will be used.

In the following example, the SETCODE statement assigns an authorization code to the output load module MOD1.

```
//LKED      EXEC    PGM=HEWL
//SYSPRINT  DD      SYSOUT=A
//SYSUT1    DD      UNIT=SYSDA,SPACE=(TRK,(10,5))
//SYSLMOD   DD      DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSLIN    DD      DSNAME=&&LOADSET,DISP=(OLD,PASS)
//                  UNIT=SYSDA
//          DD      *
    SETCODE         AC(1)
    NAME            MOD1(R)
/*
```

Nonpageable Dynamic Area

The nonpageable dynamic area is used for programs that are not to be paged during execution. Use of this are should be reserved for programs that cannot be executed in pageable dynamic storage and for programs that are not readily adaptable to a pageable environment.

If the nonpageable dynamic area is to be used, the following items should be considered:

- A large nonpageable dynamic area has a degrading effect on system performance, even when no regions in this area are allocated. This situation occurs because the allocation routines will perform extended searches and page manipulation in order to avoid allocation of long-term fixed pages and SQA and LSQA pages in the nonpageable dynamic area.
- To minimize the allocation of long-term fixed pages and SQA and LSQA pages in the nonpageable dynamic area, the size of the nonpageable dynamic area should be kept as small as possible in relation to the actual size of real storage. In a heavily overcommited

system, allocation of long-term fixed pages and SQA and LSQA pages in the nonpageable dynamic area becomes more likely and may present severe restrictions in allocating regions in this area.

- If long-term fixed pages or SQA or LSQA pages are allocated in the nonpageable dynamic area, regions cannot be allocated in the area that intersects with these pages. Since the anticipated duration of these pages is the life of the IPL, it is possible that the existence of these pages will have a cumulative degrading effect on the system.
- If short-term fixed pages are specified when long-term fixed pages should be specified, undetectable allocation in nonpageable dynamic area will occur. Also, severe system degradation will occur. (The long or short fix option selected should be based on the length of time between when a page is fixed and freed. As a rule of thumb, the long option should be specified if the time interval of the fix can be measured as a number of seconds.)
- If the actual usage of the nonpageable dynamic area is close to the size of the nonpageable dynamic area, any allocation of long-term fixed pages or SQA or LSQA pages that does occur is more likely to impact the ability to perform allocation in this area.
- Linkage to a module that resides in the link pack area by other than a LOAD macro instruction will not cause the module to be fixed. Therefore, if a module in the nonpageable dynamic area issues a LINK macro instruction (without a preceding LOAD macro instruction) for a module in the link pack area, the page(s) containing the referenced module will not be fixed. In addition, modules from the link pack area that are called via a LOAD macro instruction from a task in the nonpageable area are fixed outside of the nonpageable area.
- For key zero requests, all channel programs which originate outside of the nonpageable dynamic area will be translated. However, if a channel program originates in the nonpageable dynamic area and issues a TIC instruction outside this area, the CCW string may not be translated. Likewise, if a CCW string within the nonpageable dynamic area refers outside of the nonpageable area, translation will not occur. (Note that all non-key zero requests will not be translated.)
- When allocation is attempted for a region in the nonpageable dynamic area, any contiguous space that does not contain any long-term fixed pages or SQA or LSQA pages may be used. Although this will increase the probability of successful allocation for this request, it will also increase the probability of fragmentation in the nonpageable dynamic area.
- In a heavily I/O-oriented system, allocation in the nonpageable dynamic area may be blocked due to a high amount of fixed pages for I/O. Conversely, the existence of an allocated region in the nonpageable dynamic area may prevent I/O fixing. This situation occurs because allocated pages for the nonpageable dynamic area are included in fix threshold calculations.

Supervisor Macro Instructions for System Programmers

This chapter contains the description and formats of supervisor macro instructions and parameters that are restricted to authorized programs, programs executing in the supervisor state, or programs operating under protection key zero. The macro instructions are designed primarily for system programmers responsible for maintaining, updating, and extending the facilities of the control program.

Figure 28 lists the macro instructions described and identifies which macro instructions are fully described and which macro instructions are partially described in this chapter. For those macro instructions partially described, only those parameters that are restricted in use are described in this chapter; coverage of the other parameters of the macro instructions is contained in *OS/VS Supervisor Services and Macro Instructions*, GC27-6979.

Fully Described	Partially Described
CIRB EXTRACT IMGLIB MODESET PGFIX PGFREE PGLOAD QEDIT RESERVE STAE SYNCH TESTAUTH	ATTACH DEQ ENQ WTO WTOR

Figure 28. Supervisor Macro Instructions for System Programmers

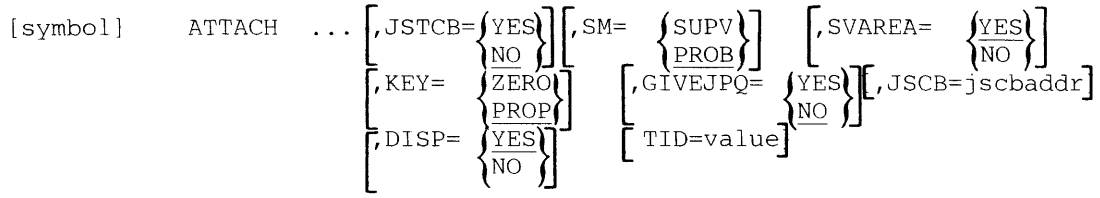
Note: For coverage of data management macro instructions for system programmers, see *OS/VS Data Management for System Programmers*, GC28-0631.

ATTACH

This explicit form of ATTACH permits greater flexibility in both the use and the result of use of the ATTACH macro instruction. This form of the macro instruction differs from the implicit form by the addition of eight keyword parameters to those described for the implicit form in the *Supervisor Services and Macro Instructions* publication. Only the added eight parameters are shown and explained in this description.

These eight parameters can be used only with tasks whose protection key is zero. If they are used with other tasks, the default values are used.

Note: The ATTACH macro instruction may not be issued by a task in 'must complete' mode.



JSTCB

Address to be placed in the TCBJSTCB field of the TCB of the newly created task. The address determines whether the attached task is a new job step or a task in the present job step.

YES-Address of the TCB of the newly created task; that is, this TCB points to itself, thus creating a new job step. A new job step is required if ownership of programs is being transferred from the attaching to the attached task, that is, if GIVEJPQ=YES is coded in the macro instruction.

NO- Address of the TCB of the task using the ATTACH; that is, the attached task is to be a task in the present job step.

SM

Operating state of the machine when executing the attached task.

SUPV -Supervisor mode.

PROB -Problem program mode.

SVAREA

Need for save area.

YES -A save area is needed for the attaching task. The ATTACH routine will obtain a 72 byte save area. If both attaching and attached task share subpool zero, the save area is obtained there; otherwise, it is obtained from a new 4K byte block.

NO -No save area is needed.

KEY

Protection/Key of the newly created (attached) task.

ZERO -Zero.

PROP -Copy the key from the TCBPKF field of the TCB for the task using the ATTACH.

GIVEJPQ

Ownership of programs used by the attaching task. If ownership is to pass to the attached task, the attached task must be a new job step; that is, JSTCB=YES must be used.

YES -Pass ownership to the newly created task. On completion of the new task, all programs, both those passed to the new task by the old and those acquire by it, are freed.

NO -Ownership of programs used by the attaching task remain with the task; programs acquired by the attached task remain with it. The attached task shares use of the programs of the attaching task during their common existence. At the conclusion of the attached task, the programs it acquired are freed; when the attaching task terminates, its programs are freed.

JSCB

Job step control block address. If specified, that job step control block is used for the new task. If not specified, the job step control blcok of the attaching task is also used for the new task.

DISP
Dispatchability of the subtask

YES -Dispatchable.

NO -Non-dispatchable.

TID
Task ID (0-255) to be placed in the TCBTID field of the attached task. The default is zero. The parameter may be specified as a decimal value or a value in one of general registers 2 through 12.

Note: If the task to be attached is to be a separate step (JCTCB=YES), ownership of programs may be passed (GIVEJPQ=YES) or retained (GIVEJPQ=NO). If the newly attached task is not to be a separate step (JSTCB=NO), ownership of programs cannot be passed but must be retained (GIVEJPQ=NO).

CIRB

The CIRB macro instruction is included in SYS1.MACLIB and must be included in the system at system generation time if the macro instruction is to be used. The issuing of this macro instruction causes a supervisor routine (called the exit effector routine) to create an interruption request block (IRB) if one does not already exist for the requesting task. In addition, other operands of this macro instruction may specify the building of a register save area and/or a work area to contain interruption queue elements, which are used by supervisor routines in the scheduling of the execution of user exit routines.

The CIRB macro instruction is written as follows:

```
[symbol]    CIRB    EP=addrx   [,KEY={PP|SUPR}] [,MODE={PP|SUPR}] [,STAB=code]
                               [,SVAREA={NO|YES}]      [,WKAREA={value|(reg)}]
                               [,RETIQE={NO|YES}]      [,TYPE={IRB|TIRB}]
                               [,ENABLE={NO|YES}]
```

EP
specifies the entry point of the user's asynchronous exit routine; this parameter is required with IRB creation; it may not be specified on a TIRB request.

KEY
specifies whether the asynchronous routine will operate with a key of zero (SUPR) or with a key obtained from the TCB of the task issuing the CIRB macro instruction (PP). For an IRB, the default is (PP); for a TIRB, the default is (SUPR). KEY=PP may not be specified with TYPE=TIRB.

MODE
specifies whether the asynchronous routine will be executed in problem program (PP) or supervisor (SUPR) mode. For an IRB, the default is (PP); for a TIRB, the default is (SUPR). MODE=PP may not be specified with TYPE=TIRB.

STAB
indicates bit settings to be made in the RBSTAB field of the newly created IRB or TIRB. Any of the following may be specified:

(RE) -turns on the RBUSIQE bit; if this bit is on, EXIT will free the SQE/IQE when the asynchronous routine returns; for a TIRB, the default is STAB=(RE).

(DYN) -turns on the RBFDYN bit; if this bit is on, EXIT will free the RB; STAB=(DYN) can not be specified for a TIRB.

(RE,DYN) -combines the above function; can not be specified for a TIRB.
(DYN,RE)

Note: For an IRB, the specification of (RE) has no effect, as the CIRB service routine unconditionally sets this bit (RBUSIQE) to zero when creating an IRB.

SVAREA
specifies whether a 72-byte register save area is to be obtained from the virtual storage assigned to the problem program. For both IRBs and TIRBs the default is SVAREA=NO; SVAREA=YES may not be specified with TYPE=TIRB. If a save area is requested, CIRB places the save area address in the IRB.

WKAREA
specifies the number of double words (decimal value) required for an area in which the invoker of CIRB can build IQEs; this value may also be placed in one of the general purpose registers 2-12, in which case the register may be identified by a numerical value (2) or symbolically (REGX). The default for both IRBs and TIRBs is zero. The WKAREA parameter may not be specified with TYPE=TIRB.

RETIQE
specifies whether the first bit of the RBIQETP subfield of RBSTAB should be turned on; this bit identifies the associated queue elements as IQEs or SQEs, as opposed to RQEs. If RETIQE=NO, the bit is set to one. The default for IRBs is RETIQE=YES; for a TIRB, the default is RETIQE=NO. RETIQE=YES may not be specified with TYPE=TIRB.

TYPE
specifies whether a TIRB or IRB is to be created. If this parameter specifies TYPE=TIRB, the 'WKAREA=' and 'EP=' parameters are invalid. Also, the following values must be specified if these parameters are explicitly coded; KEY=SUPR, MODE=SUPR, STAB=(RE), SVAREA=NO, RETIQE=NO, ENABLE=NO. If these parameters are omitted, the correct default values for TYPE=TIRB will be assumed.

For TYPE=IRB, the 'EP=' parameter must be supplied. Otherwise, there are no restrictions on parameter specification. The default is IRB.

ENABLE
specifies whether the RBOPSW field in the IRB or TIRB is to be enabled or disabled for I/O and external interruptions. If the IRB is to run in problem program mode, the default is ENABLE=YES; if it is to run in supervisor mode, the default is ENABLE=NO. ENABLE=YES may not be specified with TYPE=TIRB.

DEQ

The DEQ macro instruction is used to remove control of serially reusable resources from the active task. It should be used to release control of every resource assigned through the use of the ENQ macro instruction.

For users operating under a protection key of zero, the DEQ macro instruction may be used for several restricted functions, as indicated by the parameters below. For a description of the ordinary DEQ macro instruction parameters, see *OS/VS Supervisor Services and Macro Instructions,* GC27-6979. The other parameters are written as follows:

```
[symbol]    DEQ     ... [,RMC= {SYSTEM}] [,TCB= {tcb address}]
                           {STEP  }          {    (reg)   }
                        [,GENERIC= {YES}]
                                   {NO }
```

RMC
specifies that the requesting task release the resources and terminate the must complete function. The RMC (reset-must-complete) operand indicates the level (SYSTEM or STEP) to which the must complete function is to apply. The SYSTEM or STEP parameter must agree with the parameter specified in the SMC operand of the corresponding ENQ macro instruction.

TCB
specifies the address of a fullword on a fullword boundary that contains the address of a TCB , or it specifies a register (2-12) containing the address of a TCB (not necessarily the current TCB) on whose behalf the DEQ is to be done.

This parameter may not be specified with RMC=. The caller (not the directed task) will be abnormally terminated if RET=NONE is specified or RET= is omitted and an attempt is made to DEQ a resource not requested or not owned by the directed task.

GENERIC
specifies whether or not (YES or NO) all queue elements for the task under the specified major name will be dequeued, regardless of whether they have control of the resource. The specification of a minor name will be ignored.

This parameter may not be specified with RMC= or RET=NONE. The parameter must be specified with RET=HAVE.

ENQ

The ENQ macro instruction is used to request control of serially reusable resources for the active task.

For users operating under a protection key of zero, the ENQ macro instruction may be used for several restricted functions, as indicated by the parameters described below. For a description of the ordinary ENQ macro instruction parameters, see *OS/VS Supervisor Services and Macro Instructions,* GC27-6979. The other parameters are written as follows:

```
[symbol]    ENQ     ... [,SMC= {SYSTEM]}] [,TCB= {tcb address}]
                           {STEP   }          {    (reg)   }
                        [,ECB=ecb address]
```

SMC
specifies that the must complete function place other tasks in a wait state until the requesting task has completed its operations on the resource. The SMC (set-must-complete) operand indicates the level (SYSTEM or STEP) to which the must complete function is to apply.

TCB
specifies the address of a fullword on a fullword boundary that contains the address of a TCB, or it specifies a register (2-12) containing the address of a TCB (not necessarily the current TCB) on whose behalf the ENQ is to be done.

This parameter may not be specified with ECB=, SMC=, RET=HAVE, or RET=NONE. The parameter must be specified with RET=TEST, RET=USE, or RET=CHNG.

ECB
specifies a conditional request for all of the resources named in the macro instruction. The parameter specifies the address of an ECB.

This parameter may not be specified with RET=, SMC=, or TCB=.

Return codes are provided by the control program if ECB is designated. The return codes are:

Return Code	Meaning
0	Resource is available: caller has resource.
4	Resource is in use; the request has been placed on the queue.
8	The caller has already enqueued on the resource and is in control.
12	Resource is permanently unavailable.
16	A previous ENQ with ECB is outstanding.
20	The caller has already enqueued on the resource, but does not have control.

EXTRACT

The EXTRACT macro instruction causes the control program to provide information from specified fields of the task control block or a subsidiary control block for either the active task or one of its subtasks. The information is placed in an area provided by the problem program in the order shown in Figure 29.

The standard form of the EXTRACT macro instruction is written as follows:

```
[symbol]    EXTRACT    answer area address[,tcb location address]
                                          [,'S'                 ]
                       ,FIELDS=(codes)
```

answer area address
is the address in virtual storage of one or more consecutive fullwords, starting on a fullword boundary, to contain the requested information. The address may be written in an A-Type address constant, or one of registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses. The number of fullwords required is the same as the number of fields specified in the FIELDS operand, unless FIELDS=(ALL) is coded. If FIELDS=(ALL) is coded, seven fullwords are required.

tcb location address
specifies the address of a fullword on a fullword boundary containing the address of a task control block for a subtask of the active task. The address may be written in an A-Type address constant, or one of registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses. 'S' indicates that information is requested from the task control block for the active task. 'S' is assumed if the operand is omitted or if it is coded to specify a zero address.

FIELDS=
is one or more of the following sets of characters, written in any order and separated by commas, which are used to request the associated task control block information. The information from the requested field is returned in the relative order shown in Figure 29; if the information from a field is not requested, the associated fullword is omitted. If ALL is specified, the answer area will include all the fields in Figure 29 from GRS to TIOT, including the reserved word. Addresses are always returned in the low-order three bytes of the fullword, and the high-order byte is set to zero. Fields for which no address or value has been specified in the task control block are set to zero.

ALL
requests information from the GRS, FRS, RESERVED, AETX, PRI, CMC, and TIOT fields.

GRS
the address of the save area used by the control program to save the general registers (in the order of 0 through 15) when the task is not active.

FRS
the address of the save area used by the control program to save the floating point registers (in the order of 0, 2, 4, 6) when the task is not active.

AETX
the address of the end of task exit routine specified in the ETXR operand of the ATTACH macro instruction used to create the task.

PRI
the current limit (third byte) and dispatching (fourth byte) priorities of the task. The two high-order bytes are set to zero.

CMC
the task completion code. If the task is not complete, the field is set to zero.

TIOT
the address of the task input/output table.

COMM
the address of the command scheduler communications list. The list consists of a pointer to the communications event control block and a pointer to the command input buffer. The high-order bit of the last pointer is set to one to indicate the end of the list.

PSB
the address of the protected storage control block (PSCB), which is extracted from the job step control block. This field is meaningful only for jobs running in a time sharing environment.

TSO
the address of the time sharing flags field in the task control block. This field is meaningful only for jobs running in a time sharing environment.

TJID
the terminal job identifier (TJID) of the task specified in the tcb location address operand. This field is meaningful only for jobs running in a time sharing environment.

Note: The user must provide an answer area consisting of contiguous fullwords, one for each of the codes specified in the FIELDS operand, with the exception of the ALL code. If ALL is specified, the user must provide a 7-word answer area to accomodate the GRS, FRS, RESERVED, AETX, PRI, CMC, and TIOT fields. The ALL code does not include the COMM, PSB, TSO, and TJID codes.

For example, if FIELDS=(TIOT,GRS,PRI,TSO,PSB,TJID) is coded, a 6-fullword answer area address is required, and the extracted information will appear in the answer area in the same relative order as shown in Figure 29. (That is, GRS will be returned in the first word, PRI in the second word, TIOT in the third word, etc.)

If FIELD=(ALL,TSO,PSB,COMM,TJID) is coded, an ll-fullword answer area is required, and the extracted information will appear in the answer area in the relative order shown in Figure 29.

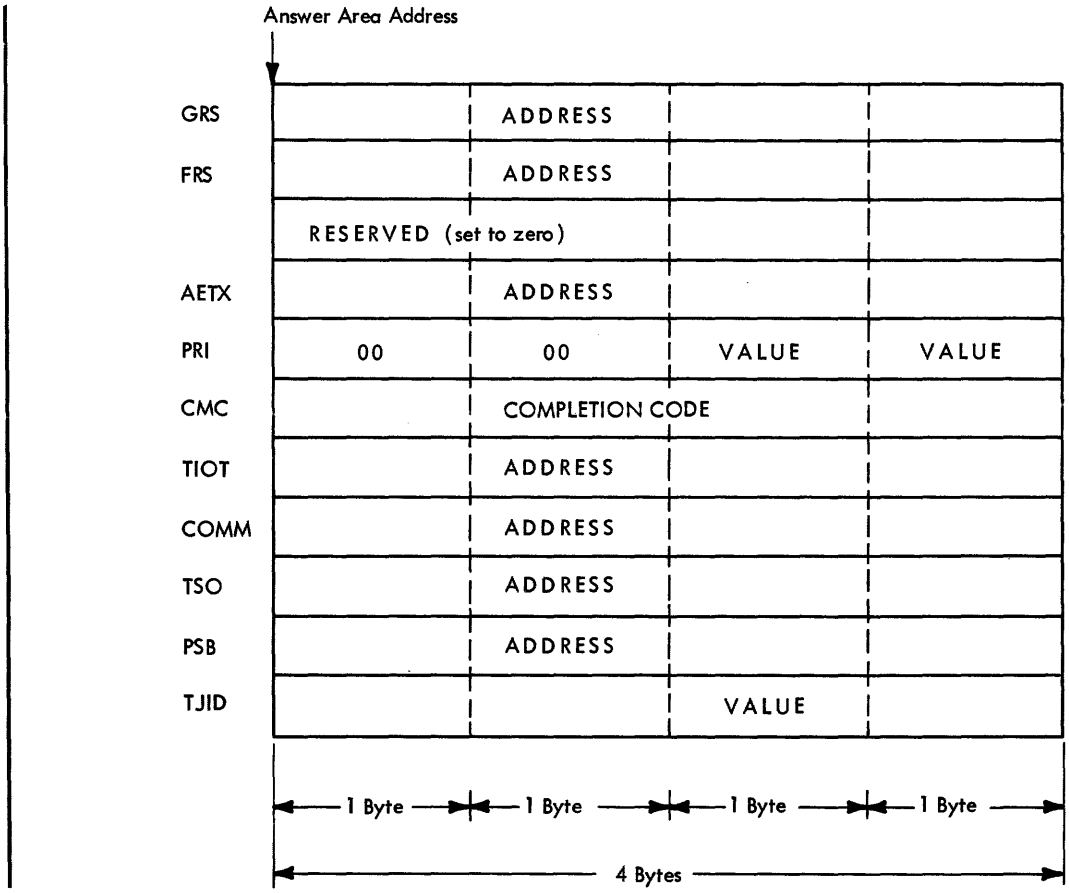


Figure 29. EXTRACT Answer Area Fields

EXTRACT -- List Form

The list form of the EXTRACT macro instruction is used to construct a control program parameter list.

The description of the standard form of the EXTRACT macro instruction provides the explanation of the function of each operand. The format description below indicates the optional and required operands in the list form only.

The list form of the EXTRACT macro instruction is written as follows:

```
[symbol]    EXTRACT [answer area address]  [, tcb location address]
                                           [, 'S'                ]
                    [,FIELDS=(codes)]   ,MF=L
```

address
is any address that may be written in an A-type address constant.

codes
are one or more of the sets of characters defined in the description of the standard from of the macro instruction. Each use of the FIELDS operand in the execute form overrides any previous codes.

MF=L
indicates the list form of the EXTRACT macro instruction.

EXTRACT -- Execute Form

A remote control program parameter list is referred to, and can be modified by, the execute form of the EXTRACT macro instruction.

The description of the standard form of the EXTRACT macro instruction provides the explanation of the function of each operand. The format description below indicates the optional and required operands in the execute form only.

The execute form of the EXTRACT macro instruction is written as follows:

```
[symbol]    EXTRACT [answer area address]  [,tcb location address]
                                           [,'S'                ]
                    [,FIELDS=(codes)]
                    ,MF=(E,   {control program list address})
                              {(1)                         }
```

address
is any address that is valid in an RX-type instruction, or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

codes
are one or more of the sets of characters defined in the description of the standard form of the macro instruction. If the FIELDS operand is used in the execute form, any codes specified in a previous FIELDS operand are cancelled and must be respecified if required for this execution of the macro instruction.

MF=(E,control program list address)
(1)
indicates the execute form of the macro instruction using a remote control program parameter list. The address of the control program parameter list can be coded as described under address, or can be loaded into register 1, in which case MF=(E,(1) should be coded.

IMGLIB

The IMGLIB macro instruction is used to open or close SYS1.IMAGELIB. When issued to open the image library, it is usually followed by a BLDL macro instruction and a LOAD macro instruction which, respectively, search the library for the image and load it into storage.

The IMGLIB macro instruction is written as follows:

```
[symbol]    IMGLIB {OPEN }  ,dcbaddr
                   {CLOSE}
```

OPEN
specifies that SYS1.IMAGELIB is to be opened and the address of the data control block (DCB) returned in register one.

CLOSE
specifies that SYS1.IMAGELIB is to be closed.

dcbaddr
is either the address of the SYS1.IMAGELIB DCB or is a register containing the SYS1.IMAGELIB DCB address.

MODESET

The MODESET macro instruction causes a supervisor routine (IEAVMODE) to alter the SVC old program status word (PSW) so that the desired PSW will be loaded when MODESET returns to the user via the type I exit routine.

The MODESET macro instruction is written as follows:

```
[symbol]    MODESET[KEY=   {NZERO}] [,MODE={PROB}]
                           { ZERO}]       {SUP }]
                [,ENABLE= {YES}] [,SYSMASK=
                          {NO }]
          ([,RELOC={YES}]  [,IO={YES}])]
                   {NO }       {NO }]
           [,MF=   (L                    )]  [REG= (1)]
                    (E,{address}         )
                       { (1)   }
```

KEY
specifies that the PSW key (bits 8-11) is to be either set to zero (ZERO) or set to the value in the caller's TCB (NZERO).

MODE
specifies that the mode indicator (bit 15) in the PSW is to be turned on (PROB) or turned off (SUP).

ENABLE
specifies that the I/O summary bit (bit 6) and external interrupt bit (bit 7) in the PSW system mask are to be turned on (YES) or turned off (NO).

SYSMASK
specifies that bits 5, 6, 7 are to be either turned on or turned off in the system.

RELOC
specifies that the address translation bit (bit 5) in the system mask is to be turned on (YES) or turned off (NO).

IO
specifies that the I/O summary bit (bit 6) in the system mask is to be turned on (YES) or turned off (NO).

EXT
specifies that the external interrupt bit (bit 7) in the system mask is to be turned on (YES) or turned off (NO).

MF
specifies the form of the macro instruction. If this parameter is omitted, the standard form of the macro instruction is assumed.

L indicates that the list form of the macro instruction is to be generated; the expansion will contain no executable code.

E, address indicates that the execute form of the macro instruction is to be generated; 'address' refer to the storage of the MODESET parameter list.

E, (1) indicates that the execute form of the macro instruction is to be generated; register 1 has been loaded with the address of the parameter list to be used.

REG
specifies that register 1 contains the parameter list to be passed to MODESET.

If MF=(E,address), MF=(E(1)), or REG=(1) is coded, then no other parameters may be specified with MODESET.

PGFIX

The PGFIX macro instruction is used to lock virtual page into page frames, preventing those pages from being paged out while the requesting task's region occupies real storage. The "locking" of these pages does not prevent them from being paged out when an entire region is swapped out of real storage for TSO users. Consequently, the user of the PGFIX macro instruction cannot assume a constant real address mapping for fixed pages that are susceptible to swapping.

The user of the PGFIX macro instruction should also be aware of the implication of PGFIX. VS2 uses a system oriented algorithm in controlling the number of fixed pages. The algorithm ensures that on a system basis, the number of fixed pages at any one time does not exceed a preassigned threshhold. As a result of using this approach, there is no control mechanism provided by VS2 that monitors fixing at the task level.

An installation should establish firm control over who uses the PGFIX macro instruction, and over what duration of fix. Poor control can result in any of the following:

- Low priority tasks being set nondispatchable because of thrashing. This occurs because the number of fixed pages has reduced the working set to a size that cannot handle the current paging rate.
- Degraded system throughput performance because of the increased paging rate.

- Nonpageable jobs not being started because long fixes have been assigned in the nonpageable dynamic area. Also, the more pages that are fixed, the smaller is the number of pages that are available and the greater is the contention for the smaller number of available page frames.
- Possible system wait states because no pages frames are available to satisfy SQA requests.
- Potential interlocks when the SUSPEND option is used.

The macro instruction is written as follows:

```
[symbol]    PGFIX   L,LA= {(reg)} [,ECB=     {(reg)}     ] ,LONG= {N}
                          {addr }  [        {addr }     ]        {Y}
                                   [,ECBIND=  {(reg)}     ]
                                   [          {addr }     ]
                  [,RELEASE= {Y}] [,SUSPEND= {N}]
                             {N}             {Y}
```

L
indicates that a parameter list is being supplied with this request.

LA
specifies the address of a parameter list that describes the extents of the virtual area(s) to be fixed.

The parameter list must be guaranteed nonpageable, and must remain unchanged until the operation is posted complete. If it is not protected and nonpageable, an exposure to system error is incurred. To be nonpageable, the parameter list must:

- Reside in subpool 233, 234, 235, 243, 244, 245, 253, 254, or 255 (i.e., LSQA or SQA).
- Reside in a nonpageable dynamic region.
- Reside in the nucleus or fixed link pack area.
- Be fixed by a previous PGFIX macro instruction.

Parameter List
If a return code of 4 or a completion code of 4 is presented to the issuer of the PGFIX macro instruction, the parameter list may have been modified to indicated that one or more virtual areas that were requested for fixing are undefined. (Virtual areas that have not been allocated to a region (in 4K blocks), are considered to be undefined and unaddressable.) The execution of the PGFIX macro instruction will flag a parameter list entry that describes an undefined virtual area. The flag is located at byte 4, bit 3 of the parameter list entry, and is referred to as the "error" flag in the parameter list description.

ECB
specifies the address of an event control block. Since paging I/O may be required to satisfy a FIX request, an ECB must be provided so that the requester can be informed when the FIX is done. The supervisor will post the ECB with a completion code when the request is completed.

Caution: Only one FIX at a time should be issued with a given ECB; two incomplete FIXes should not be waiting on the same ECB.

ECB Completion Code

If a hexadecimal return code of 8, 28 (with SUSPEND option specified), or 2C (with SUSPEND option specified), is received following macro execution, the ECB supplied as an operand of the PGFIX macro instruction will be posted asynchronously with a completion code. The completion code, stored in the ECB, should be examined after WAIT macro execution.

Completion Code	Meaning
00	Operation complete; no action.
04	Error detected; no pages were fixed.
48	Operation purged; the ECB must be reinitialized and the PGFIX macro instruction must be reissued. The completion code occurs when an asynchronous system event has caused the issuing task to be quiesced. The FIX operation must be rescheduled before it can complete.

ECBIND

specifies the address of an indirect area, where bytes 1-4 contain the address of an ECB and byte 0 contains the ECB completion code when the ECB is posted.

LONG

indicates to the supervisor the relative real time duration anticipated for the FIX. If Y is specified, the requester is indicating that the duration of the FIX will be relatively long. (As a rule of thumb, the duration of a FIX is considered long if the time interval can be measured on a ordinary timepiece). Special consideration may be given to a LONG request by the supervisor in allocating real storage to prevent fragmentation of the nonpageable dynamic area. Y is the default value.

If N is specified, the time duration of the FIX is assumed to be relatively short; for example, the time needed to complete an I/O operation to a direct access device is considered to be short.

RELEASE

If Y is specified, the contents of the virtual areas specified for fixing can be discarded before fixing occurs. For areas which consist of a 4K page or more, page-in operations for those pages can be bypassed if those pages do not occupy real storage; scratch page frames may be assigned to them. The use of this option does not guarantee that the FIX will complete immediately (a wait for page frame allocation may be required, for example).

If N is specified, the contents of the virtual areas being fixed must be retained. N is the default value.

SUSPEND

If Y is specified, the paging supervisor will internally queue the request if real storage is depleted.

If N is specified, the paging supervisor will reject the request if real storage is depleted. N is the default value.

Control is returned to the instruction following the PGFIX macro instruction. When control is returned, register 15 contains one of the following return codes:

Hex Code	Meaning
00	Operation complete.
04	Error detected; no pages were fixed.
08	Operation proceeding; a WAIT macro instruction should be issued for the ECB that was used with the PGFIX macro instruction.
24	FIX request too large; no pages were fixed.
If SUSPEND option is specified:	
28	FIX request queued due to a shortage of long-term fixable real storage (long wait). If no interlock exposure exists, and if a long-term wait (possibly for the "life" of a subsystem) is acceptable, a WAIT macro instruction should be issued. Otherwise, the PGFIX operation should be canceled using the PGFREE macro instruction with the ECB/ECBIND operand.

PGFIX -- Non-Standard Form

```
[symbol]    PGFIX   R,A=   {(reg)}  [,ECB=    {(reg)}      ]  [,LONG={N}]
                           {addr }  [        {addr }      ]     {Y}
                                    [,ECBIND= {(reg)}      ]
                                              {addr }      ]
```

This form of the PGFIX macro instruction differs from the standard form in the following ways:

- A parameter list is not used.
- SUSPEND and RELEASE options are not provided.
- Only one virtual area, one byte in length, can be fixed by execution of this macro form.

R

indicates that no parameter list is being supplied with this request.

A

virtual address of the single-byte area to be fixed.

Fixing actually occurs on a 4K storage block, even though the virtual area specified is a single byte in length. Thus, any virtual area less than 4K in length can be fixed with this macro form, as long as the area does not cross a 4K boundary. Note that an area less than 4K in length and allocated using the GETMAIN macro instruction with the BNDRY=PAGE parameter can be guaranteed as fixed using this form of the PGFIX macro instruction.

ECB

the address of an event control block. Since paging I/O may be required to satisfy a FIX request, an ECB must be provided to that the requester can be informed when the FIX is done. The supervisor will post the ECB with a completion code when the request is completed.

Caution: Only one FIX at a time should be issued with a given ECB; two incomplete FIXes should not be waiting on the same ECB.

ECB Completion Code

If a return code of 8 is received following macro execution, the ECB supplied as an operand of the PGFIX macro instruction will be posted asynchronously with a completion code. The completion code, stored in the ECB, should be examined after WAIT macro execution.

Completion Code	Meaning
00	Operation complete; no action.
04	Error detected; no fixing has occurred.
30	Operation purged; the ECB must be reinitialized and the PGFIX macro instruction must be reissued. This completion code occurs when an asynchronous system event has caused the issuing task to be quiesced. The FIX operation must be rescheduled before it can complete.

ECBIND
specifies the address of an indirect area, where bytes 1-4 contain the address of an ECB and byte 0 contains the ECB completion code when the ECB is posted.

LONG
indicates to the supervisor the relative real time duration anticipated for the FIX. If Y is specified, then the requester is indicating that the duration of the FIX will be relatively long; as a rule of thumb, the duration of a FIX is considered LONG if the time interval can be measured on an ordinary timepiece (a number of seconds, or more). Special consideration must be given to a long request by the supervisor in allocating real storage to prevent fragmentation of the nonpageable dynamic area. Y is the default value.

If N is specified, the time duration of the FIX is assumed to be relatively short; for example, the time needed to complete an I/O operation to a direct access device is considered to be short.

Control is returned to the instruction following the PGFIX macro instruction. When control is returned, register 15 contains one of the following return codes:

Hex Code	Meaning
00	Operation complete.
04	Error detected; no fixing has occurred.
08	Operation proceeding; issue a WAIT macro instruction for the ECB that was used with the PGFIX macro instruction.
28	FIX request rejected due to shortage of fixable real storage. Caller must reissue request at a later time. If no interlock exposure exists, reissue request with SUSPEND option. This option cannot be specified in non-standard form.
2C	FIX request queued due to shortage of fixable real storage (short-term wait). If no interlock exposure exists, issue WAIT macro instruction as above. If an interlock exposure exists, cancel the PGFIX operation using the PGFREE macro instruction with the ECB/ECBIND operand.
If SUSPEND option is not specified:	
2C	FIX request rejected due to shortage of fixable real storage. Caller must reissue request at a later time. If no interlock exposure exists, reissue request with SUSPEND option.

PGFREE

The PGFREE macro instruction is used to reverse the action of the PGFIX macro instruction, thus making specified virtual areas eligible for paging. A PGFREE macro instruction must be issued for each PGFIX macro instruction that was issued against a page. In addition to reversing the action of the PGFIX macro instruction, the PGFREE macro instruction may be used to cancel PGFIX requests before they complete.

The macro instruction is written as follows:

```
[symbol]    PGFREE  L,LA=  {(reg)}  [,ECB=  {(reg)}      ]
                           {addr }          {addr }
                   [,RELEASE={Y}]   [,ECBIND= {(reg)}    ]
                             {N}            {addr }
```

L
indicates that a parameter list is being supplied with this request.

LA
the address of a parameter list that describes the extents of the virtual area(s) to be freed (un-fixed).

The same restriction applies to PGFREE parameter list as that which applies to the PGFIX parameter list (i.e., the parameter list must be nonpageable).

ECB
the address of an event control block that was used in a prior execution of a PGFIX macro instruction. When this operand is specified, a PGFIX operation that has not completed and was issued with the same ECB address as the ECB operand of PGFIX will be cancelled. If no incomplete PGFIX operation exists with that ECB address, the ECB/ECBIND operand is ignored by PGFREE.

If an incomplete PGFIX operation exists and the ECB/ECBIND operand is not specified with PGFREE, the results of the PGFIX operation are unpredictable.

ECBIND
specifies the address of an indirect area, where bytes 1-4 contain the address of the ECB and byte 0 contains the ECB completion when the ECB is posted.

RELEASE
If Y is specified, the contents of the virtual area(s) specified for unfixing can be discarded after unfixing occurs. For areas which circumscribe a 4K page or more, future page-out operations for those pages can be eliminated; the real storage occupied by those areas can be made available at once.

If N is specified, the contents of the virtual area(s) being unfixed must be retained. N is the default value.

PGFREE -- Non-Standard Form

```
[symbol]    PGFREE  R,A=  {(reg)}  [,ECB=  {(reg)}      ]
                          {addr }          {addr }
                          [,ECBIND=  {(reg)}    ]
                                     {addr }
```

This form of the PGFREE macro differs from the standard form in the following ways:

- A parameter list is not used.
- RELEASE option is not provided.
- Only one virtual area, one byte in length, can be unfixed by the execution of this macro form.

R
indicates that no parameter list is being supplied with this request.

A
specifies the virtual address of the single-byte virtual area to be unfixed.

ECB and ECBIND
specifies the address of an event control block; see the description of the operand in the standard form of PGFREE.

General register 15 contains a return code after macro execution:

Return code = 0 -- operation successful

Return code = 4 -- operation could not be done.

Parameter List Structure
The basic parameter list is composed of one or more contiguous doubleword entries; each entry describes an area of virtual storage. In addition, the first entry defines indicators for the functions and/or options requested; these indicators are set by the PGFIX and PGFREE macros. Each parameter list entry has the following format:

```
0       1               4       5               7
FLAGS   START ADDRESS   FLAGS   END ADDRESS +1
```

- START ADDRESS is the virtual address of the origin of the virtual area to be fixed or freed.
- END ADDRESS+1 is the virtual address of the byte immediately following the end of the virtual area. Note that this address is equal to the start address plus the length, in bytes, of the area.
- FLAGS are defined as followed:

  Byte 0, bit 0 (1... ....) Continuation Flag
  The entry is defined as 4 bytes in length; bytes 1 through 3 are a pointer to the next parameter list entry. This feature allows several parameter lists to be chained as a single logical parameter list.

  Byte 0, bits 1-7 (.xxx xxxx) Reserved
  These bites are used in the first entry by the interface; they may not be used by the macro caller.

  Byte 4, bit 0 (1... ....) Last Entry Flag
  This flag must be used to terminate the parameter list; it is set in the last doubleword entry in the list.

  Byte 4, bit 1 (.1.. ....) Null Entry Flag
  When this flag is set, the entry in which it is set is ignored.

  Byte 4, bit 2 (..1. ....) Reserved

  Byte 4, bit 3 (...1 ....) Error Flag
  This flag is set by PGFIX in conjunction with completion code 4; it indicates that the flagged entry represents a virtual area of which some portion is undefined.

  Byte 4, bites 4-7 (.... xxxx) Reserved

PGLOAD

The PGLOAD macro instruction can explicitly request that virtual pages be placed in real storage. The function precludes any logical dependence upon the residency of those pages at the completion of the PGLOAD operation.

The misuse of this function can have adverse effects on system performance. Causing unnecessary pages to be brought into real storage will force more useful pages to be displaced, and consequently, cause unnecessary paging activity.

The PGLOAD function can be used in conjunction with the EC mode SPIE capability, so that a program may execute in parallel with the resolution of a page fault. In terms of performance, this usage is beneficial only when the program using the function is a bottleneck to the rest of the system (e.g., a heavily used subsystem).

The macro instructions is written as follows:

```
[symbol]    PGLOAD  { R,A=    {(reg)}  [,ECB=    {(reg)}]                          }
                    {addr }  [,ECBIND= {(reg)}]                                    }
                             {addr }                                               }
                    { L,LA=   {(reg)}  [,ECB=    {(reg)}]  [,RELEASE= {Y}]      }
                    {addr }  [,ECBIND= {(reg)}]              {N}      }
                             {addr }                                               }
```

R
specifies that a single virtual storage address is provided as an argument.

A
specifies the virtual storage address of an area that resides within the 4K page that is to be brought into real storage.

L
specifies that a parameter list, containing one or more virtual address ranges, is provided as an argument.

LA
specifies the address of the parameter list.

The basic parameter list is composed of one or more continguous doubleword entries; each entry describes an area of virtual storage. In addition, the first entry defines indicators for the functions and/or options requested; these indicators are set by the PGLOAD macro. Each parameter list entry has the following format:

```
0       1               4       5               7
FLAGS   START'ADDRESS   FLAGS   END ADDRESS+1
```

- START ADDRESS is the virtual address of the origin of the virtual area to be loaded into real storage.
- END ADDRESS+1 is the virtual address of the byte immediately following the end of the virtual area. Note that this address is equal to the start address plus the length, in bytes, of the area.
- FLAGS are defined as follows:

  Byte 0, bit 0 (1... ....) Continuation Flag
  The entry is defined as 4 bytes in length; bytes 1 through 3 are a pointer to the next parameter list entry. This feature allows several parameter lists to be chained as a single logical parameter list.

  Byte 0, bits 1-7 (.xxx xxxx) Reserved
  These bits are used in the first entry by the interface; they may not be used by the macro caller.

Byte 4, bit 0 (1... ....) Last Entry Flag
This flag must be used to terminate the parameter list; it is set in the last doubleword entry in the list.

Byte 4, bit 1 (.1.. ....) Null Entry Flag
When this flag is set, the entry in which it is set is ignored.

Byte 4, bit 2 (..1. ....) Reserved

Byte 4, bit 3 (...1 ....) Error Flag
This flag is set by the LOAD PAGE processor; it indicates that the flagged entry represents a virtual area, of which some portion is undefined. The setting of this flag will accompanied by a return code or completion code of 4.

Byte 4, bits 4-7 (.... xxxx) Reserved
The parameter list for PGFIX must be guaranteed nonpageable, and must remain unchanged until the operation is posted complete. To be nonpageable, the parameter list must:

- Reside in subpool 233, 234, 235, 243, 244, 245, 253, 254, or 255 (i.e., LSQA or SQA).
- Reside in a nonpageable dynamic region.
- Reside in the nucleus or fixed link pack area.
- Be fixed by a PGFIX macro instruction.

The parameter list is used like a control block by the supervisor and in particular by the interruption handlers. If it is not protected and nonpageable, an exposure to system error is incurred.

ECB
The address of an event control block. The complete bit in the ECB should be set to 0. The ECB will be posted by the supervisor after the PAGE LOAD operation has completed.

Caution: Even though a LOAD PAGE operation has been posted complete, the virtual area may not be in real replacement in an asynchronous fashion by the supervisor. The posting of the ECB indicates only that a page-in operation was successfully completed at some point prior to the posting of the ECB.

ECBIND
specifies the address of an indirect area, where bytes 1-4 contain the address of the ECB and byte 0 contains the ECB completion code when the ECB is posted.

RELEASE
If Y is specified, the contents of the virtual areas specified can be discarded before the operation occurs. For areas which contain a 4K page or more, page-in operations for those pages can be bypassed if those pages do not already occupy real storage; scratch page frames may be assigned to them. The use of this option does not guarantee that the PGLOAD will complete immediately (a wait for page frame allocation may be required, for example).

If N is specified, the contents of the virtual areas will be retained. N is the default value. General register 15 contains a return code after macro execution.

Hex Code	Meaning
00	Operation complete; no action.
04	Error detected.
08	Operation proceeding; issue a WAIT macro instruction for the ECB that was used with the PGLOAD macro instruction.

ECB Completion Code

If a return code of 8 is received following macro execution, the ECB supplied as an operand of the PGLOAD macro instruction will be posted asynchronously with a completion code. This completion code, stored in the ECB, should be examined after WAIT macro execution.

Hex Code	Meaning
00	Operation complete.
04	Error detected.

Parameter List

If a return code of 4 or a completion code of 4 is presented to the issuer of the PGLOAD macro instruction, the parameter list may have been modified to indicate that one or more virtual areas were requested for loading are undefined. (Virtual areas that have not been allocated to a region in 4K block, are considered to be undefined and unaddressable.) The execution of the PGLOAD macro instruction will flag any parameter list entry that describes and undefined virtual area. The flag is located at byte 4, bit 3 of the parameter list entry, and is referenced to as the "error" flag in the parameter list.

QEDIT

The QEDIT macro instruction generates the required entry parameters and the linkage to SVC 34 for the following uses:

- Dechaining and freeing of a command input buffer (CIB) from the CIB chain for a task.
- Setting a limit for the number of CIBs that may be simultaneously chained for a task.

The QEDIT macro instruction is written as follows:

```
[symbol]    QEDIT   ORIGIN=address  [,BLOCK=address]
                                    [,CIBCTR=number]
```

ORIGIN=address

specifies the address of the pointer to the first CIB chain for the task. This address is obtained using the EXTRACT macro instruction. If ORIGIN is the only parameter specified, the entire CIB chain will be freed. The address is any address valid in an RX instruction or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

BLOCK=address

specifies the address of the CIB that is to be freed from the CIB chain for a task. The address is any address valid in an RX instruction or one of general registers 2 through 12, previously loaded with the indicated address. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

CIBCTR=number

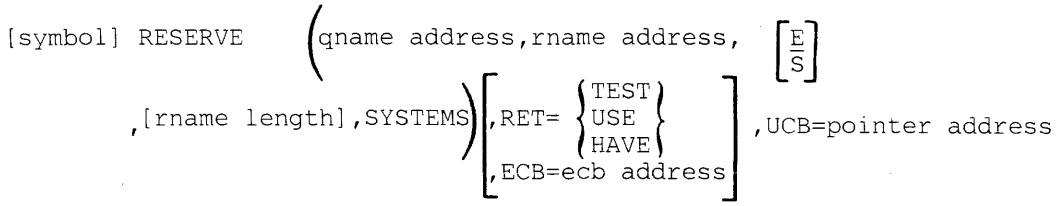
is an integer (from 0 to 255) to be used as a limit for the number of CIBs to be chained at any time for a task.

RESERVE

The RESERVE macro instruction is used to reserve a device for use by a particular system; it must be issued by each task needing device reservation. The RESERVE macro instruction protects the issuing task from interference by other tasks in the system. Each task issuing two RESERVE instructions for the same resource without an intervening DEQ will result in an abnormal termination unless the second RESERVE specifies the keyword parameter RET=. (If a restart occurs when a RESERVE is in effect for devices, the system will not restore the

RESERVE; the user's program must reissue the RESERVE.) Even if a DEQ is not issued for a particular device, termination routines will release devices reserved by a terminating task.

The RESERVE macro instruction is written as follows:



qname address
: the address in virtual storage of an eight-character name. Every task (within the system) issuing RESERVE against the same resource (data and device) must use the same qname-rname combination to represent the resource. The qname should not start with SYS.

rname address
: the address in virtual storage of a name used in conjunction with the qname to represent the resource. The rname can be qualified, and may be 1 to 255 bytes in length.

E or S
: specifies either exclusive control or the resource (E) or shared control with other tasks in the system(S). The default is E.

rname length
: the length, in bytes, of rname. If omitted, the assembled length of rname is used. If zero (0) is specified, the length of rname must be contained in the first byte of the field designated by the rname address.

SYSTEMS
: specifies that the resource represented by qname-rname is known across systems as well as within the system whose task is issuing RESERVE; i.e., the resource is shared between systems.

RET
: specifies a conditional request for all the resources named in the RESERVE macro instruction. If the operand is omitted and ECB is not specified, the request is unconditional. (RET and ECB cannot both be specified.) The types of conditional requests are as follows:

  TEST -- tests the availability status of the resources. The requestor is never assigned control of the resources.

  USE -- specifies that control of the resources be assigned to the active task only if the resources are immediately available. If any of the resources are not available, the active task is not placed in a wait condition.

  HAVE -- specifies that control of the resources is requested only if a request has not been made previously for the same task.

ECB=ecb address
: specifies a conditional request for all of the resources named in the RESERVE macro instruction. The address is of an event control block. If the operand is omitted and RET is not specified, the request is unconditional. (ECB and RET cannot both be specified.)

Return codes are provided by the control program only if RET or ECB are designated; otherwise, return of the task to the active condition indicates that control of the resource has been assigned to the task. Return codes are identical to those supplied by the ENQ macro instruction.

UCB=pointer address
specifies either the address of a fullword that contains the address of the unit control block (UCB) for the device to be reserved, or a general register (2-12) that points to a fullword containing the address of the unit control block for the device to be reserved.

To use the shared DASD option in higher level languages, an assembler language subroutine should be written to issue the RESERVE macro instruction. The following information should be passed to this routine: ddname, qname address, rname address, rname length, and RET parameter.

STAE

The STAE macro instruction enables the user to intercept a scheduled ABEND and to have control returned to him at a specified exit routine address. The STAE macro instruction operates in both problem program and supervisor modes.

The STAE macro instruction creates a STAE control block (SCB) which represents a STAE environment tht remains in effect during the execution of the program that issued the STAE or until canceled by a subsequent STAE. When a RETURN or XCTL is issued, the system automatically cancels the STAE environment for that program, unless XCTL=YES is coded in the STAE macro instruction. If XCTL=YES is coded and an XCTL macro instruction is issued, the STAE environment remains in effect for the program that receives control as a result of the XCTL macro instruction.

When a STAE environment is canceled, the last STAE environment that was created and not subsequently overlayed or canceled (if any) becomes the current STAE environment.

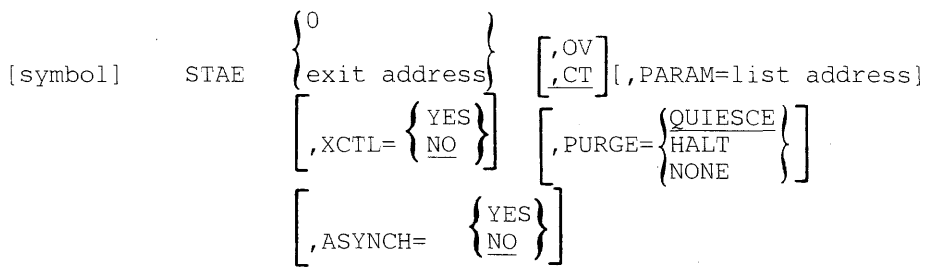
Note that issuing a LINK macro instruction does not cancel the STAE environment and that the user is responsible for canceling the STAE environment if his program does not exit via a RETURN or XCTL. The user cannot cancel or overlay a STAE control block not created by his own program.

Within the STAE exit routine, the user may perform pre-termination functions or diagnose an error. Upon completion of STAE exit routine processing, the user can either allow abnormal termination processing to continue for the task or request that a STAE retry routine be scheduled which would circumvent the scheduled ABEND.

The STAE exit routine cannot contain a STAE or an ATTACH macro instruction. When a STAE retry routine is not to be scheduled, the STAE exit routine should return with a code of zero in register 15.

Entry to a STAE retry routine cancels the STAE environment. If a STAE retry routine causes the task to resume execution, the STAE environment should be reestablished from within the retry routine.

The STAE macro instruction is written as follows:



exit address
specifies the address of a STAE exit routine to be entered if the task issuing this macro instruction terminates abnormally. If 0 is specified, the most recent STAE request is canceled. The address may be loaded into one of the general registers 2 through 12.

OV
indicates that the parameters passed in this STAE macro instruction are to overlay the data contained in the previous STAE request. In the standard form only of the STAE macro instruction, if any of the parameters XCTL, PURGE, or ASYNCH are not specified, the default value for the omitted parameter is assigned.

CT
indicates the creation of a new STAE request. If neither OV or CT is specified, CT is assumed.

PARAM=list address
specifies the address of a parameter list containing data to be used by the STAE exit routine when it is scheduled for execution. The address may be loaded into one of the general registers 2 through 12.

XCTL=YES
indicates that the STAE macro instruction will not be canceled if an XCTL macro instruction is issued.

XCTL=NO
indicates that the STAE macro instruction will be canceled if an XCTL is issued by this program. If neither XCTL=YES or XCTL=NO is coded, XCTL=NO is assumed.

PURGE=
QUIESCE

indicates that all outstanding requests for input/output (I/O) operations will be saved when the STAE exit is taken. At the end of the STAE exit routine, the user can code a retry routine to handle the outstanding I/O requests.

If the PURGE operand is not specified, QUIESCE is assumed. If I/O cannot be quiesced, then I/O is halted (see PURGE=HALT).

HALT
indicates that all outstanding requests for input/output operations will not be saved when the STAE exit is taken.

NONE
indicates that input/output processing is allowed to continue normally when the STAE exit is taken.

Notes: If any IBM-supplied access method, except EXCP, is being used, the PURGE=NONE option is recommended. If this is done, all control blocks affected by input/output processing may continue to change during STAE exit routine processing.

If PURGE=NONE is specified and the ABEND was originally scheduled because of an error in input/output processing, an ABEND recursion will develop when an input/output interruption occurs, even if the exit routine is in progress. Thus, it will appear that the exit routine failed when, in reality, input/output processing was the cause of the failure.

ASYNCH=

YES

indicates that asynchronous interrupt processing is allowed to interrupt the processing done by the STAE exit routine. ASYNCH=YES must be coded if:

- Any supervisor services that require asynchronous interruptions to complete their normal processing are going to be requested by the STAE exit routine.
- PURGE=QUIESCE is specified for any access method that requires asynchronous interruptions to complete normal input/output processing.
- PURGE=NONE is specified and the CHECK macro instruction is issued in the STAE exit routine for any access method that requires asynchronous interruptions to complete normal input/output processing.

Note: If ASYNCH=YES is specified and the ABEND was originally scheduled because of an error in asynchronous exit handling, an ABEND recursion will develop when an asynchronous interruption occurs. Thus, it will appear that the exit routine failed when, in reality, asynchronous exit handling was the cause of the failure.

NO

indicates that asynchronous interrupt processing is not allowed to interrupt the processing done by the STAE exit routine. IF the ASYNCH operand is not specified, NO is assumed.

ISAM Notes: If ISAM is being used and PURGE=HALT is specified or PURGE=QUIESCE is specified but I/O is not restored:

- Only the input/output event on which the purge is done will be posted. Subsequent event control blocks (ECBs) will not be posted.
- The ISAM check routine will treat purged I/O as normal I/O.
- Part of the data set may be destroyed if the data set is being updated or added to when the failure occurred.

Control is returned to the instruction following the STAE macro instruction. When control is returned, register 15 contains one of the following return codes:

Hex Code	Meaning
00	Indicates successful completion of creating, overlaying, or canceling a STAE request.
04	Indicates that STAE was unable to obtain storage for the STAE request.
08	Indicates that the user was attempting to cancel or overlay a nonexistent STAE request, or that the user issued a STAE in his STAE exit routine.
0C	Indicates that the exit routine or parameter list address was invalid.
10	Indicates that the user was attempting to cancel or overlay a STAE request of another user.

STAE -- List Form

The list form of the STAE macro instruction is used to construct a control program parameter list.

The description of the standard form of the STAE macro instruction provides the explanation of the function of each operand.

The format description below indicates the optional and required operands in the list form only.

The list form of the STAE macro instruction is written as follows:

```
[symbol]     STAE     [exit address]  [PARAM=list address]
                      [,PURGE={QUIESCE}]  [,ASYNCH=  {YES}]
                              {HALT    }          {NO }
                              {NONE    }
                      ,MF=L
```

address
is any address that may be written in an A-type address constant.

MF=L
indicates the list form of the STAE macro instruction.

STAE -- Execute Form

A remote control program parameter list is used in, and can be modified by, the execute form of the STAE macro instruction. The control program parameter list can be generated by the list form of the STAE macro instruction. If the user desires to dynamically change the contents of the remote STAE parameter list, he may do so by coding a new exit address and/or a new parameter list address. If exit address or PARM= is coded, only the associated field in the remote STAE parameter list will be changed. The other field will remain as it was before the current STAE request was made.

The description of the standard form of the STAE macro instruction provides the explanation of the function of each operand.

The format description below indicates the optional and required operands in the execute form only.

The execute form of the STAE macro instruction is written as follows:

```
[symbol]     STAE   [ 0          ]  [,OV]
                    [exit address]  [,CT][,PARAM=list address]
                    [,XCTL= {YES}]
                            {NO }
                    [,PURGE={QUIESCE}]      [,ASYNCH=  {YES}]
                            {HALT    }                 {NO }
                            {NONE    }
                    ,MF=(E,{remote list address})
                           {(1)                 }
```

address
is any address that is valid in an RX-type instruction, or one of the general registers 2 through 12, previously loaded with the indicated address. The register can be designated symbolically or as an absolute expression, and is always coded within parentheses.

MF=(E, remote list address)
(1)

indicates the execute form of the STAE macro instruction using a remote parameter list. The address of the remote parameter list can be loaded into register 1, in which case MF=(E,(1) should be coded.

SYNCH

The SYNCH macro instruction permits control program supervisor call (SVC) routines to make synchronous exits to a processing program. It is written as follows:

```
[symbol]     SYNCH  {entry-point}
                    {(15)      }
```

entry-point

specifies the address of the entry point for the processing program that is to be given control.

(15)

if (15) is specified, the entry-point address of the processing program must have been pre-loaded into parameter register 15 before execution of this macro instruction.

SYNCH Macro Definition

```
            MACRO
&NAME       SYNCH           &EP
            AIF             ('&EP' EQ '').E1
            AIF             ('&EP' (1,1) EQ '(').REG
&NAME       LA              15,&EP                  LOAD ENTRY POINT
                                                    ADDRESS.
            AGO             .SVC
.REG        AIF             ('&EP' EQ '(15)').NAMEIT
&NAME       LR              15,&EP(1)               LOAD ENTRY POINT
                                                    ADDRESS.
.SVC        SVC             12                      ISSUE SYNCH SVC
            MEXIT
.NAMEIT     ANOP
&NAME       SVC             12                      ISSUE SYNCH SVC
            MEXIT
.E1         IHBERMAC 27,405
            MEND
```

Programming Notes: In general, the SYNCH macro instruction is used when a control program in the supervisor state is to give temporary control to a processing program routine, and when the processing program is expected to return control to the supervisor state. The program to which control is given must be in virtual storage when the macro instruction is issued. The use of this macro instruction is similar to that of the BALR instruction in that register 15 is used for the entry point address. When the processing program returns control, the supervisor state bit, the storage protection key bits, the system mask bits, and the program mask bits of the program status word are restored to the settings they had before execution of the SYNCH macro instruction.

Example: As a result of an OPEN macro instruction, label processing may be carried out to a point at which a user's processing program indicates that private processing is desired (or necessary). The control program's open routine then will issue a SYNCH macro instruction giving the entry point of the subroutine required for the user's private label processing.

TESTAUTH

The TESTAUTH macro instruction supports the authorized program facility (APF). If is used to determine if a program is authorized to use a program or function restricted in use by APF.

The macro instruction is written as follows:

```
[symbol]     TESTAUTH     FCTN=value
                          [,AUTH=value]
```

FCTN
specifies a value representing whether the function is to be performed. If 0 is specified, no authorization is required for the function. If 1 is specified, authorization is required. The value may be specified as one of general registers 2 through 12 previously loaded with the right-adjusted value. The register may be designated symbolically or with an absolute expression, and is always coded within parentheses.

AUTH
specifies a value representing the authorization level to be tested. If 0 is specified, the job step is not authorized to perform any restricted function. If 1 is specified, the job step is authorized to perform restricted functions. If AUTH is not coded, the authorization of the current job step is used.

Control is returned to the instruction following the TESTAUTH macro instruction. When control is returned, register 15 contains one of the following return codes:

Hex Code	Meaning
00	The task is authorized for the specified function.
04	The task is not authorized for the specified function.
08	The specified values for the function or authorization codes are not valid.

WTO/WTOR

The write-to-operator (WTO) and write-to-operator with reply (WTOR) macro instructions have two special operands, MSGTYP and MCSFLAG. Only programmers familiar with multiple console support (MCS) should use these operands, since using them improperly could impede the entire message routing scheme. These operands set flags to indicate that certain system functions must be performed, or that a certain type of information is being presented by the WTO or WTOR macro instruction.

The MSGTYP and MCSFLAG operands may be specified in either the standard or list form of the WTO and WTOR macro instruction. The standard form of the WTO macro instruction is shown below. Only the MSGTYP and MCSFLAG parameters are shown and explained in this description. For ordinary WTO/WTOR macro instruction parameters, see the description in *OS/VS Supervisor Services and Macro Instructions,* GC27-6979.

```
[symbol]     WTO/WTOR     ...[,MSGTYP=      {ACTIVE  }]
                                            {SESS    }
                                            {N       }
                                            {Y       }
                                            {JOBNAMES}
                                            {STATUS  }
                          [,MCSFLAG=(name [, name],...)]
```

MSGTYP=JOBNAMES or MSGTYP=STATUS or MSGTYP=SESS
specifies that the message is to be routed to the console which issued the MONITOR JOBNAMES, MONITOR STATUS or MONITOR SESS command, respectively. When the message type is identified by the operating system, the message will be routed to only those consoles that had requested the information. Omission of the MSGTYP parameter causes the message to be routed as specified in the ROUTCDE parameter.

MSGTYP=ACTIVE
specifies that the multiple-line message is in response to a MONITOR A (MN A) command and should be routed to the console that issued the command.

MSGTYP=Y or MSGTYP=N
specifies that two bytes are to be reserved in the WTO or WTOR macro expansion so that flags can be set to describe what MSGTYP functions are desired (see below). Y specifies that two bytes of zeros are to be included in the macro expansion at displacement WTO + 4 + the total length of the message text, descriptor code, and routing code fields. N, or omission of the MSGTYP parameter, specifies that the two bytes are not needed, and that the message is to be routed as specified in the ROUTCDE parameter. If an invalid MSGTYP value is encountered, a value of N is assumed, and a diagnostic message is produced (severity code of 8).

The bit definitions for MSGTYP=Y follow:

Bit 0: MONITOR JOBNAMES

Bit 1: MONITOR STATUS

Bit 2:MONITOR ACTIVE

Bits 3-4:Reserved

Bit 5:MONITOR SESS

Bits 6-15:Reserved for future system use

When MSGTYP=Y is specified, the issuer of the WTO or WTOR macro instruction that contains the MSGTYP information must set the appropriate message identifier bit in the MSGTYP field of the macro expansion. Prior to executing the WTO or WTOR SVC (SVC 35), the issuer must also set byte 0 of the MCSFLAG field in the macro expansion to a value of hexadecimal 10. This value indicates that the MSGTYP field is to be used for the message routing criteria. When the message type is identified by the system, the message will be routed to all consoles that had requested that particular type of information. Routing codes, if present, will be ignored.

MCSFLAG
specifies that the macro expansion should set bits in the MCSFLAG field as indicated by each name coded. Names and their corresponding bit settings are shown in Figure 30.

Name	Bit	Meaning
-----	0	Invalid entry.
REG0	1	Message is to be queued to the console whose source ID is passed in Register 0.
RESP	2	The WTO is an immediate command response.
-----	3	Invalid entry.
REPLY	4	The WTO macro instruction is a reply to a WTOR macro instruction.
BRDCST	5	Message should be broadcast to all active consoles.
HRDCPY	6	Message queued for hard copy only. This operand is invalid with the multiple-line form of WTO.
QREG0	7	Message is to be queued unconditionally to the console whose source ID is passed in Register 0.
NOTIME	8	Time is not appended to the message. This operand is invalid with the multiple-line form of WTO.
---	9-12	Invalid entry.
NOCPY	13	If the WTO or WTOR macro instruction is issued by a program in the supervisor state, the message is not queued for hard copy. Otherwise, this parameter is ignored.
---	14-15	Invalid entry.
Note: Invalid specifications are ignored and produce an appropriate error message.		

Figure 30. MCSFLAG Parameters

System Overview

The system overview chapter summarizes the VS2 control program, emphasizing job, task, data, and recovery management. Users who wish to learn general concepts about how VS2 operates can use the system overview as a basic learning tool. Users who must learn the logic of VS2 can use the system overview as an index to get to the appropriate logic manuals.

This chapter consists of three sections:

- Method of Operation
- Program Organization
- System Communications

The Method of Operation section consists of seven M. O. (Method of Operation) diagrams which show functions performed by VS2. The M. O. diagrams show initialization, input processing, job processing, output processing, recovery operations, and TSO (time sharing option) processing.

The Program Organization section shows the sequence of operation for a typical batch job stream. This section user cross-references (numbers in parentheses) to the M. O. diagrams to correlate the two sections to one another.

The System Communications section gives the meaning of several important and new control blocks used by VS2. The new control blocks are used for paging data between virtual and real storage.

The preface lists the publications referred to in this chapter.

Method of Operation

This section illustrates the functions performed by the VS2 control program. It consists of seven diagrams:

- Program Structure
- System Initialization
- Processing Input
- Processing Jobs
- Processing Output
- Recovery Management
- TSO

The extended descriptions refer to other IBM publications containing detailed information about the function illustrated, and when appropriate, refer to index words in the referenced publication. The user can look up the index word in the referenced publication, and find detailed information quickly.

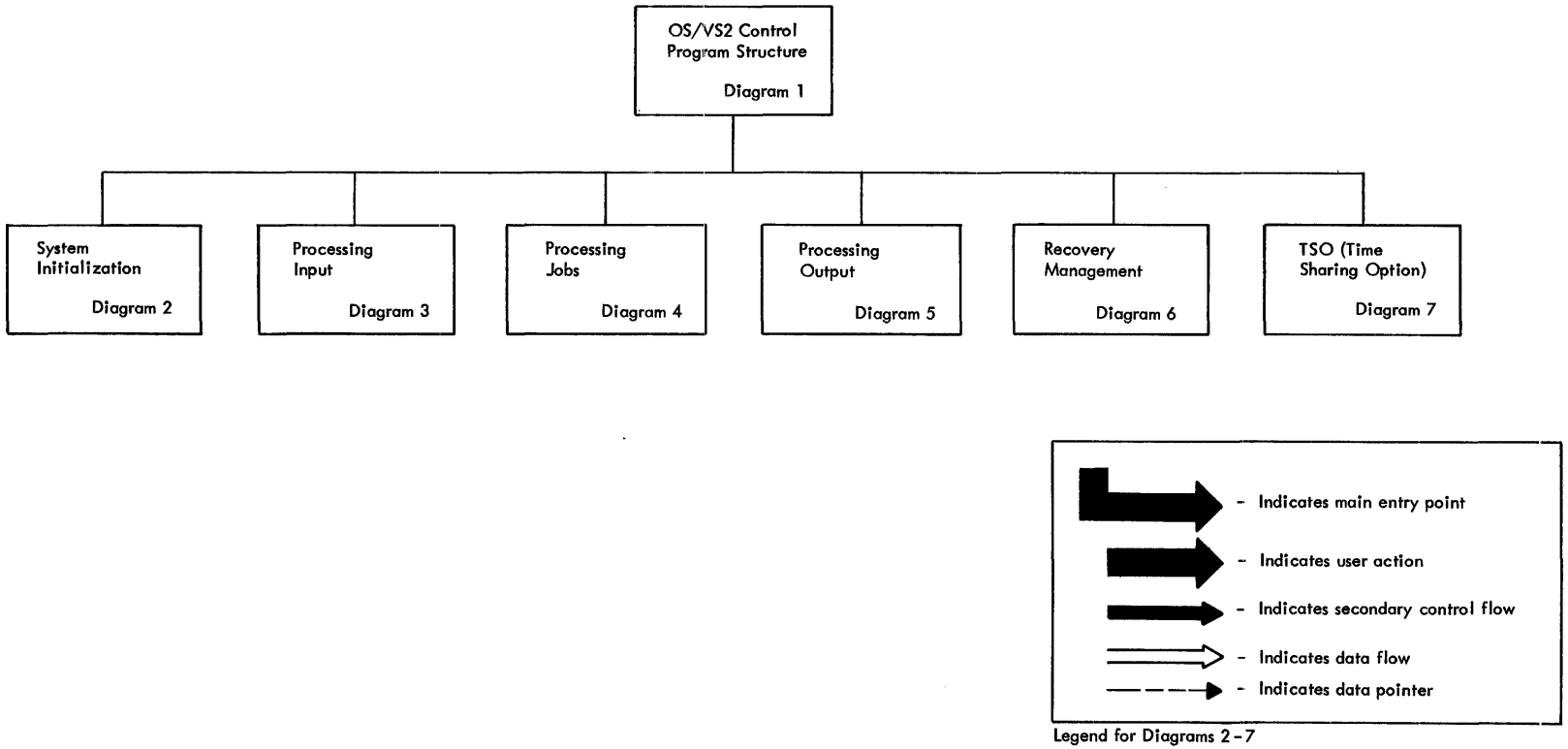


Figure 31. Overview of Method of Operation Diagrams

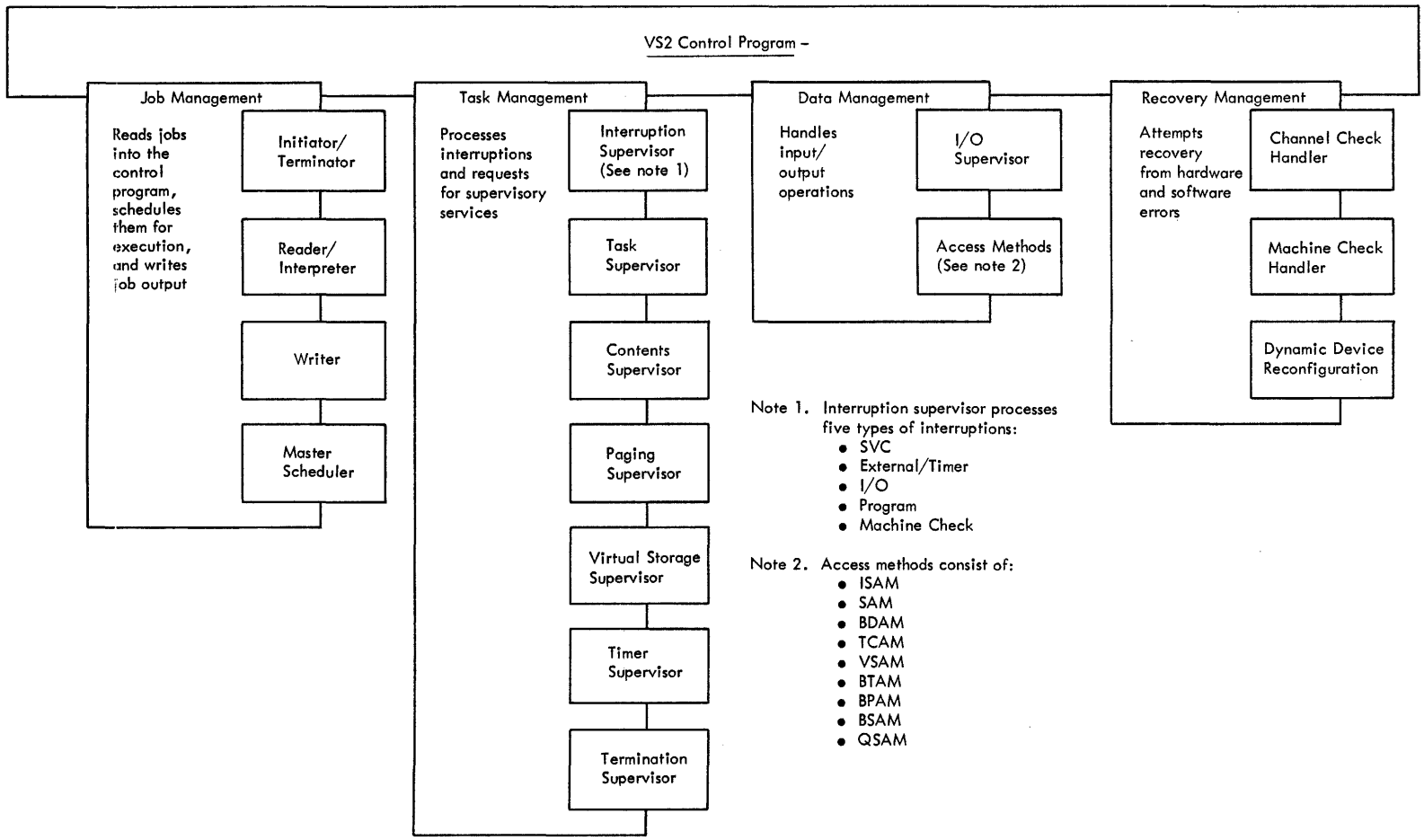


Diagram 1. Program Structure

Diagram 1. Program Structure

Diagram 1 illustrates the major functions that constitute VS2. Since each function consists of many routines, and this system overview is not detailed, only the basic concepts of VS2 are described.

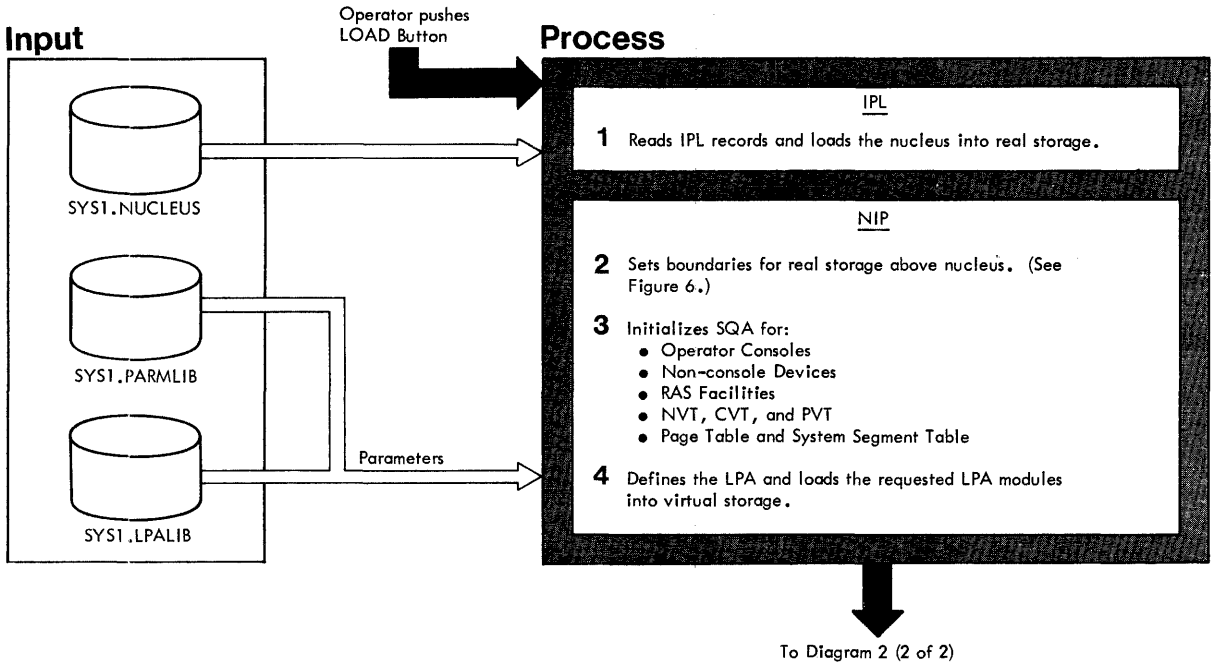


Diagram 2. System Initialization (Part 1 of 2)

Diagram 2. System Initialization

Item	Description	Reference	Index Word
1	After the operator presses the LOAD button, the first IPL record, containing a CCW (channel command word), is read into locations 0-23. The CCW in this record reads, in turn, a chain of CCWs, which read in the remainder of the IPL control section. The doubleword at location 0-15 then becomes the new PSW (program status word). IPL completes its processing by: • Clearing general and floating point registers. • Clearing real storage to the highest addressable location. • Relocating code above the nucleus. • Loading the selected nucleus and transfering control to NIP (nucleus initialization program).	IPL and NIP Logic	record reading nucleus type
2	NIP determines the boundaries for real and virtual storage above the nucleus.	IPL and NIP Logic	storage boundaries
3	NIP initializes important devices, tables, and system facilities, such as the NVT (nucleus vector table), the CVT (communications vector table), the PVT (paging supervisor vector table), and the volumes used for paging.	IPL and NIP Logic	
4	NIP determines the restart characteristics of the system. It then loads the LPA (link pack area) according to the values in the parameter list in SYS1.PARMLIB.	IPL and NIP Logic	quickstart

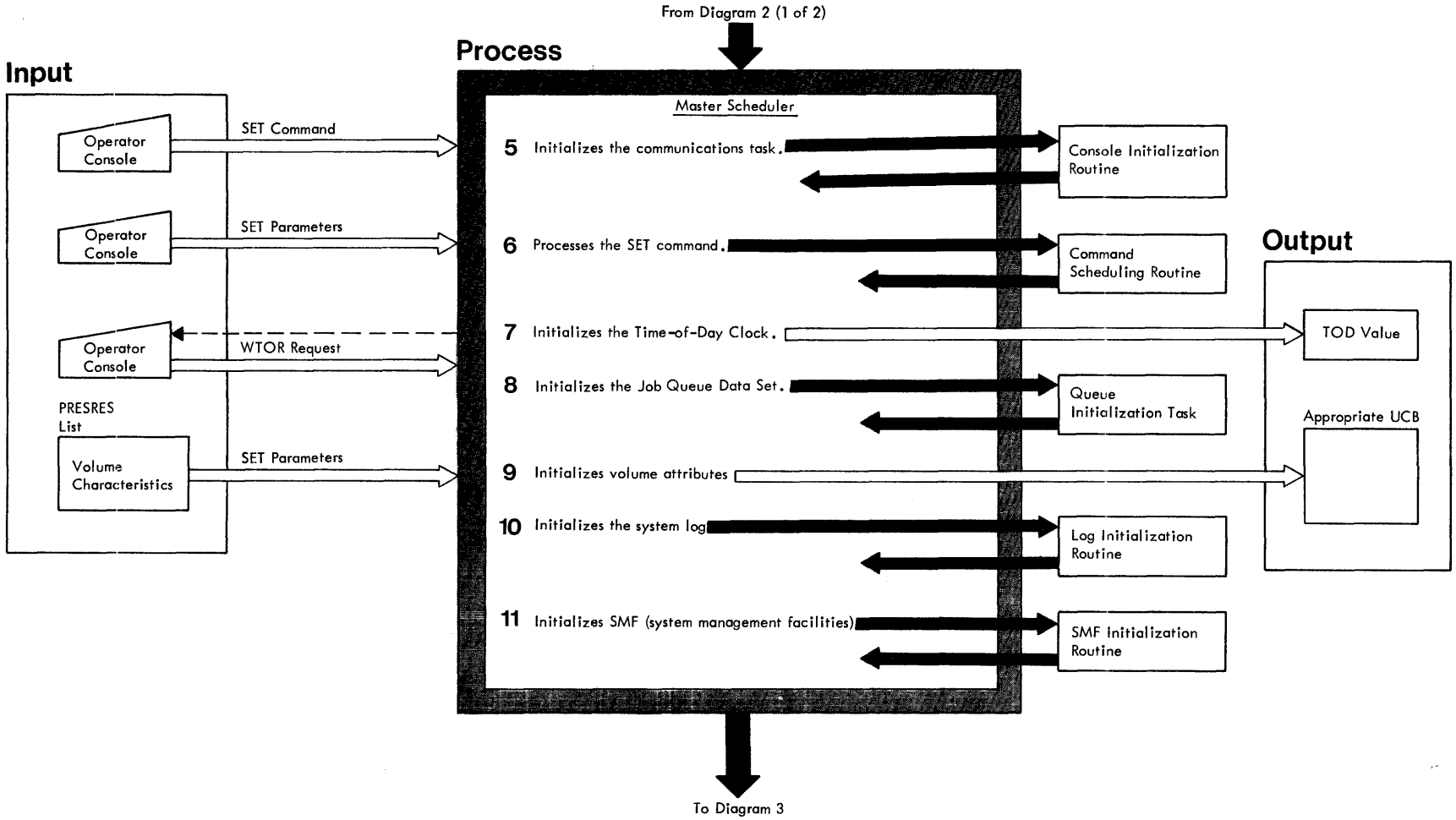


Diagram 2. System Initialization (Part 2 of 2)

Diagram 2. System Initialization

Item	Description	Reference	Index Word
5	The master scheduler IPL routine initializes the console. Then, the IPL routine displays commands available for execution as a result of the AUTO parameter in a SET command.	Job Management Logic	
6	The SET command causes the master scheduler IPL routine to look for the procedure library and the work queue data set. After finding the procedure library data set, the Command Scheduling routine catalogs the procedure library data set into the save volume on which it was found.	Job Management Logic	SET command
7	The master scheduler IPL routine asks the operator for SET parameters to set the TOD (time-of-day) clock.	Job Management Logic	SET command
8	The IPL routine uses an ATTACH macro to pass control to the queue initialization task.	Job Management Logic	
9	The volume-attribute-setting routine ensures that the volumes specified in the PRESRES data set are mounted and that the volume attributes are correct in the proper UCB.	Job Management Logic	
10	The system log data sets are located and initialized. If there is no log in the system, the operator receives a message to that effect.	Job Management Logic	system log
11	SMF initialization includes obtaining and storing SMF parameters, allocating devices for and opening of SMF data sets, establishing the SMF task, initializing a 10-minute timer, and issuing the initial SMF records.	Job Management Logic	SMF initialization

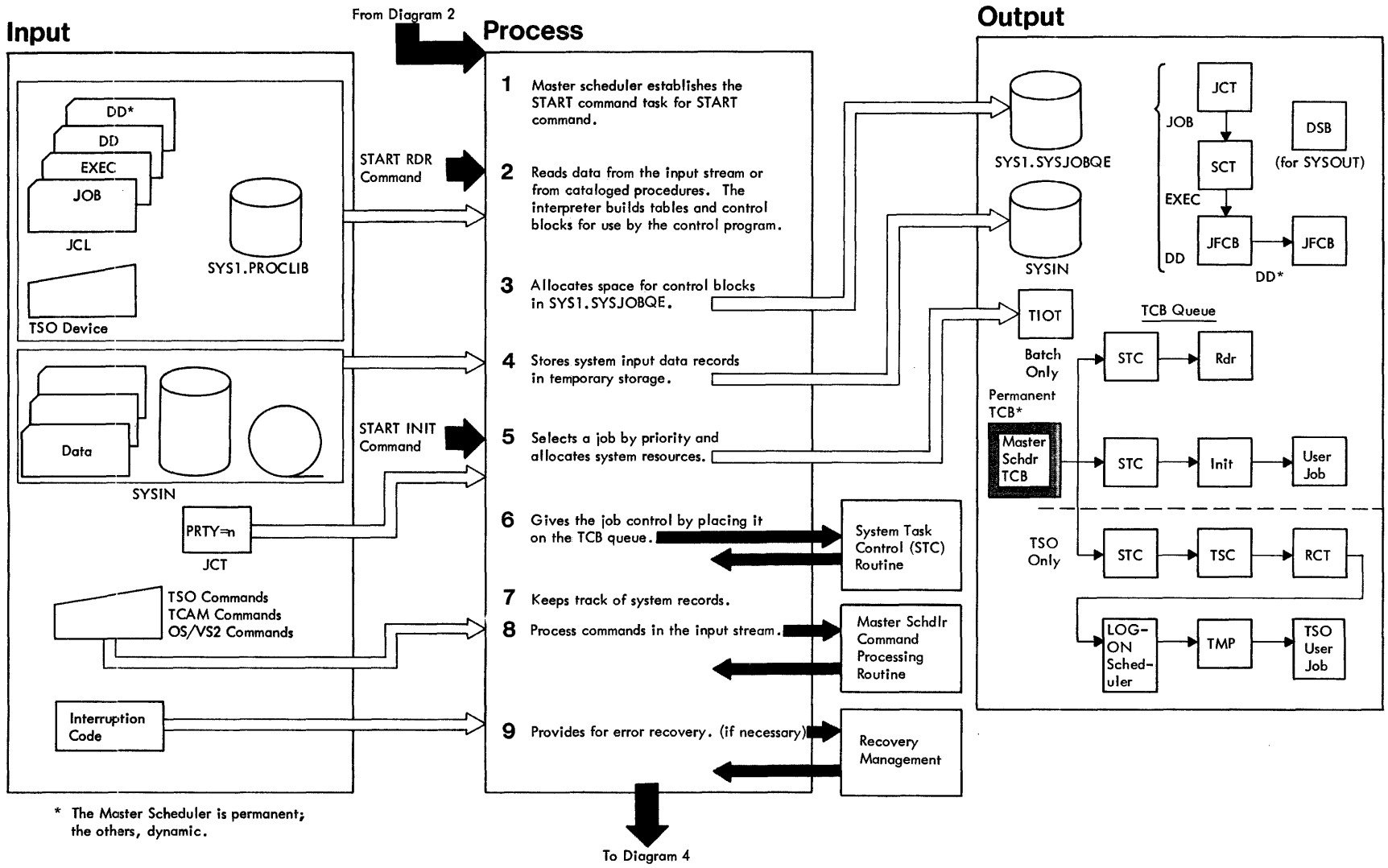


Diagram 3. Processing Input

Diagram 3. Processing Input

Item	Description	Reference	Index Word
1	After master scheduler initialization, the master scheduler prepares for START commands by using the START command task.	Job Management Logic	
2	When the operator issues a START RDR command, the START command task of the master scheduler issues an ATTACH macro instruction to give control to the reader. The input stream can come from several I/O devices: terminals, readers, direct access and tape devices. The input stream consists of JOB, EXEC, and DD statements, data, and commands. The interpreter builds tables and control blocks, such as the JFCB (job file control block), constructed for each DD statement encountered, the SCT (step control table), constructed for the EXEC statement, the JCT (job control table), constructed for the job statement, and the DSB (data set block) for SYSOUT data sets. These control blocks represent the input stream to the operating system. TSO jobs are read and interpreted by the same routines as background jobs, but they are initiated differently. Item 6 explains the procedure used to initiate TSO jobs.	Job Management Logic	control block creating
3	The reader places the JCT, SCT, JFCBs and other control blocks on the SYS1.SYSJOBQE data set to await processing. These records define variables such as device types, access routine methods, and data organization.	Job Management Logic	job queue
4	The reader stores the input data on the SYSIN data set. The JFCB built for the DD* statement points to the input data. When the problem program receives control, it will read the data from the SYSIN data set, after determining its location from information on SYS1.SYSJOBQE.	Job Management Logic	input storage
5	The operator starts the initiator for a particular class by issuing a START INIT command. In OS/VS2, 63 (42 for TSO) initiators can be started concurrently since in the virtual storage environment protection keys are no longer limited to 16. The initiator selects the highest-priority job on the queue and allocates virtual storage, auxiliary storage, I/O devices, data sets, or any other system resources. A TIOT (task input/output table) contains the allocation requests. The initiator determines resource requirements by the values contained in the control blocks built by the reader and interpreter. If the resources requested by a task cannot be acquired, the task waits. (The operator has the option of canceling the task.)	Job Management Logic	resource requests
6	The initiator gives control to the task by placing the task's TCB (task control block) on the TCB queue. (The system task control (STC) routine attaches the task to the TCB queue after receiving control from the initiator.) The TCB queue consists of two types of TCBs -- permanent and dynamic. Permanent TCBs are constructed during system initialization, and consist of the paging supervisor task, the system error task, the dynamic device reconfiguration task, the master scheduler task, and others. Dynamic TCBs are constructed when the ATTACH macro is issued. All tasks in VS2 have their TCBs on this queue. Tasks operating under control of TSO have a similar TCB structure; however, TSO tasks are queued from the TSC (time sharing control) routine, and not the initiator.	Job Management Logic	attaching task command types existing task task creating
7	Various catalog management routines keep system records in order and easily accessible.	Job Management Logic	
8	The master scheduler processes commands entered into the operating system. Commands can be either of two types: task-creating or existing-task commands. Existing-task commands can be processed by the appropriate command processing routines upon entry; task-creating commands must be scheduled for execution. In the second case, the master scheduler builds a CSCB (command scheduling control block) to contain the command request. The command will then be processed when the resources become available.		
9	The VS2 control program contains recovery management support that reduces or cancels the effects of most programming and hardware errors. When an interruption (Diagram 4, item 3) that would stop the operating system occurs, supervisor routines pass control to the appropriate recovery management routines. Diagram 6 details the major recovery management services available to the VS2 user.		

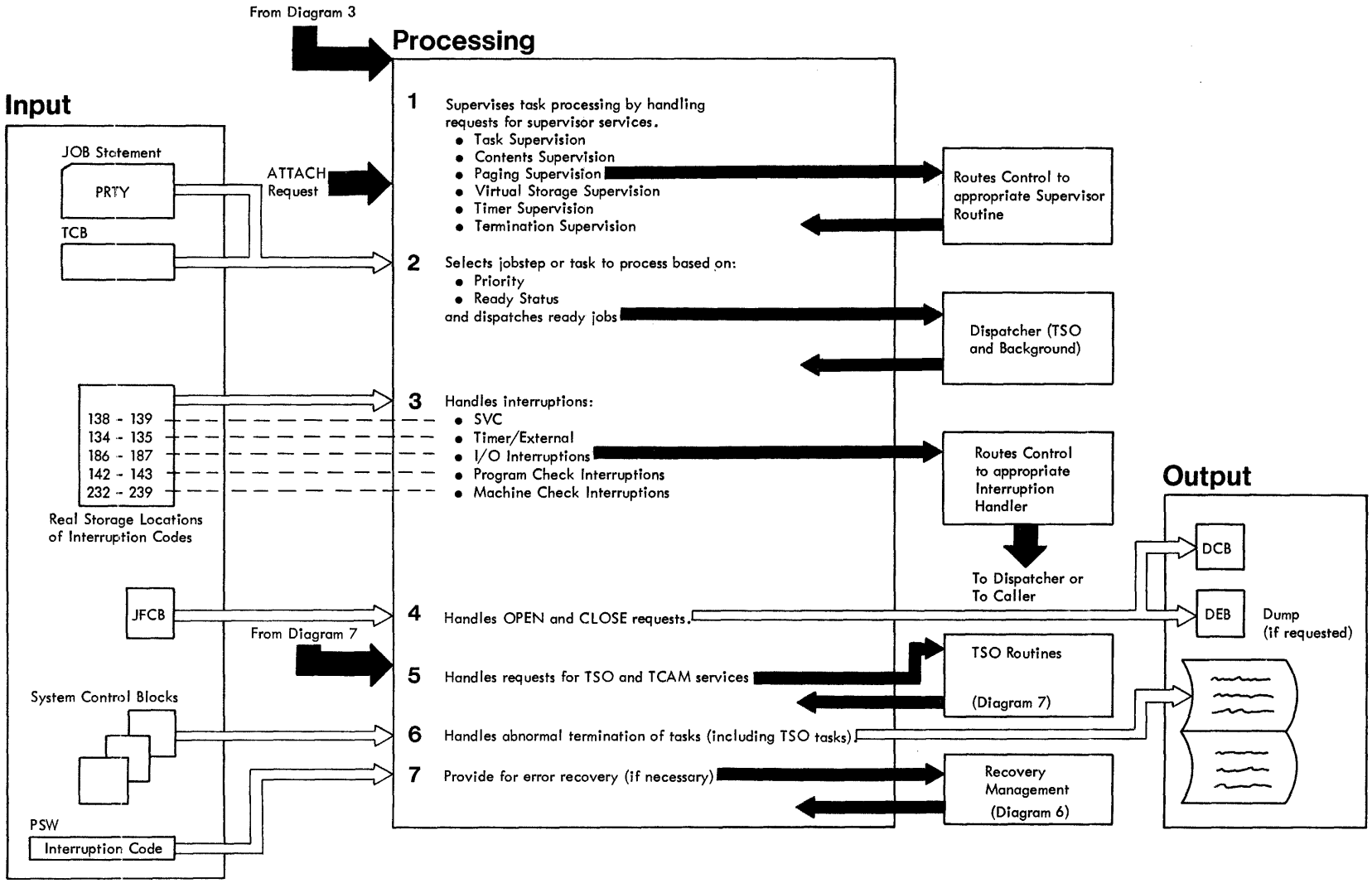


Diagram 4. Processing Jobs

Diagram 4. Processing Jobs

Item	Description	Reference	Index Word
1	The supervisor handles requests for supervisor services. Requests of this nature are in the form of macro instructions. These macros, when expanded, end with an SVC (supervisor call) instruction. The SVC in turn causes an interruption (a machine characteristic) as described in item 3. The supervisor then manipulates the system control blocks to change the scheduling or execution of system tasks.	Supervisor Logic	
2	Each task in the system has an execution priority between 0 (lowest) and 15. System tasks always have the highest priorities. The TCB contains a bit indicating whether or not a task is ready to process. The dispatcher (a supervisor routine that is part of the task supervisor) chooses the highest priority ready task, and marks it as dispatchable. The task can now receive control.	Supervisor Logic	dispatcher TCB
3	The VS2 control program processes five types of interruptions. Each interruption, except for machine check interruptions, is routed through a series of routines that determine the cause of the interruption and the type of processing required to satisfy the interruption. Machine check interruptions go to the Machine Check Handler, a part of recovery management (Diagram 6). When interruptions occur, the current PSW (program status word) contains a record of the system status, and a specific location in real storage contains a code indicating the cause of the interruption.	Supervisor Logic	interruption handling

Item	Description	Reference	Index Word
4	The OPEN routines (part of data management) locate and logically connect the input data sets to the job. Further, they fill in the final information needed by the DCB (data control block), and build the DEB (data extent block). Data management access method routines need this information to transfer data. The CLOSE routines logically disconnect the data sets from the job.	OPEN/CLOSE/ EOV Logic	
5	TSO and TCAM consist of routines that operate under control of the VS2 control program. TSO routines service requests to dynamically change the operation of TSO. TCAM runs under control of VS2 with specific routines handling service requests, in a manner similar to TSO. (See Diagram 7.)	TSO Control Program Logic	
6	When a task fails to finish, supervisor routines terminate it abnormally. Depending on the failure, either the entire job step task or the single task is terminated. If desired, a dump of the task can be taken to help indicate the reason for the failure.	Supervisor Logic	termination
7	The VS2 control program contains recovery management support that reduces or cancels the effects of most programming and hardware errors. When an interruption that would stop the operating system occurs, supervisor routines pass control to the appropriate recovery management routines.		

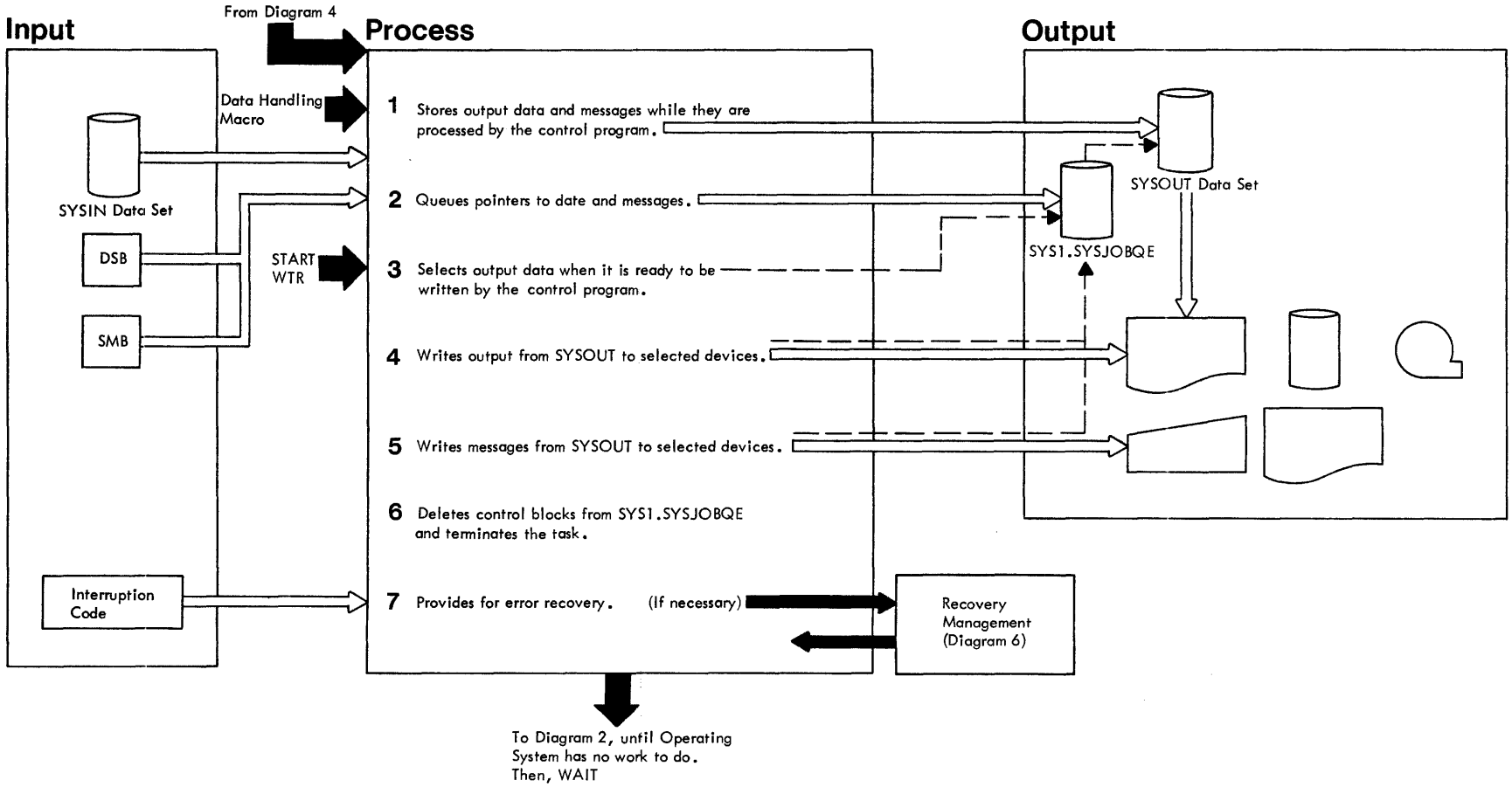


Diagram 5. Processing Output

Diagram 5. Processing Output

Item	Description	Reference	Index Word
1	Each task issues data handling macros during its processing, and the control program stores the data in the SYSOUT data set. System messages issued to the programmer are also stored in the SYSOUT data set. TSO and TCAM transfer data by using message handling programs created by TCAM macro instructions. The TCAM Logic and the TSO Control Program Logic publications describe message handling programs.	Job Management Logic	reading records
2	Pointers to data, such as DSBs (data set blocks) and SMBs (system message blocks) are located on the SYSJOBQE data set. These control blocks contain addresses and other system control information.	Job Management Logic	
3	The operator issues a start command (START WTR) to start a writer for any of 36 output classes. After the writer receives control (via an ATTACH macro), it selects data from its class. The writer will print the data as output from that class until the operator issues a STOP command or until there is no data. More than one writer can be active at a time, with more than one writer active for each class.	Job Management Logic	
4	The writer writes the data to the device specified by the installation.	Job Management Logic	
5	The writer writes the messages to the programmer as specified.	Job Management Logic	
6	Supervisor routines disconnect the control blocks associated with the task, and then terminate the task normally.	Supervisor Logic	termination
7	The VS2 control program contains recovery management support that reduces or cancels the effects of most programming and hardware errors. When an interruption that would stop the operating system occurs, supervisor routines pass control to the appropriate recovery management routines.	Job Management Logic	

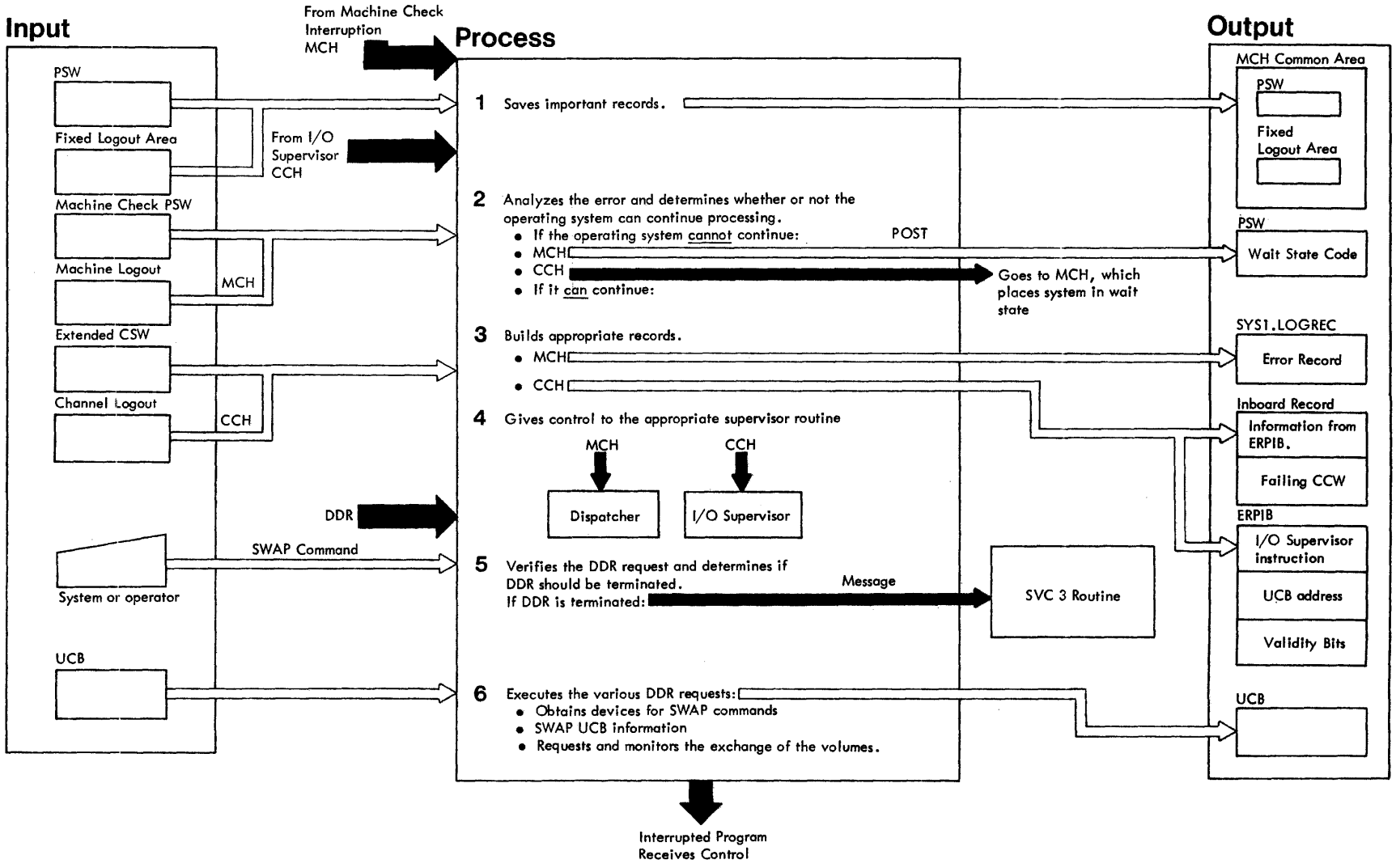


Diagram 6. Recovery Management

Diagram 6. Recovery Management

Item	Description	Reference	Index Word
1	MCH (machine check handler) saves the machine check PSW (program status word) and the fixed logout area. This allows control to be returned to the CPU (central processing unit) at the point where the error occured, if necessary.	Recovery Management Support Logic	
2	MCH receives control when the machine-check interruption code indicates: • System damage, where the error cannot be attributed to the instruction referred to by old PSW. • Instruction processing damage, where the instruction referred to by the old PSW failed. • Hardware retry successful, where the CPU instruction was retried successfully. • ECC (error correction code), where the ECC facility repaired the error. • Time facility damage, where the TOD clock failed. • Timer damage, where the high resolution timer contained a parity error.	Recovery Management Support Logic	error analysis
	CCH (channel check handler) receives control from the I/O supervisor when the extended CSW (channel status word) indicates: • A channel data check • A channel control check • An interface control check. Unlike MCH, CCH cannot place the system in a wait state. CCH gives control to MCH for system termination conditions.		channel failure
3	MCH performs two types of error recording: one for formatted error recording where the records go to SYS1.LOGREC for eventual printing; and the other for emergency recording where the records go to SYS1.LOGREC when the system cannot continue. The MCH error record consists of the abbreviated record, the fixed logout, and the damage-assessment field of the MCH common area.	Recovery Management Support Logic	recovery paths
	CCH prepares input for the I/O supervisor, which attempts channel retry. The input consists of the CCH error record, which contains an ERPIB (error recovery procedure interface block) and a channel inboard record.	Recovery Management Support Logic	ERPIB
4	MCH gives control to the dispatcher or to the interrupted program. CCH passes control to the I/O supervisor, which uses the CCH records to attempt channel retry operations.	Recovery Management Support Logic	I/O supervisor processing termination
5	DDR (dynamic device reconfiguration) allows the operator to move a volume from an I/O unit that has recurring failures to another. The SWAP command initiates DDR. DDR verifies the SWAP request, controls the execution of the request, and then performs the request.	Recovery Management Support Logic	SWAP command
6	DDR prepares for volume changing by updating the UCB (unit control block) information for the devices indicated by the request.	Recovery Management Support Logic	

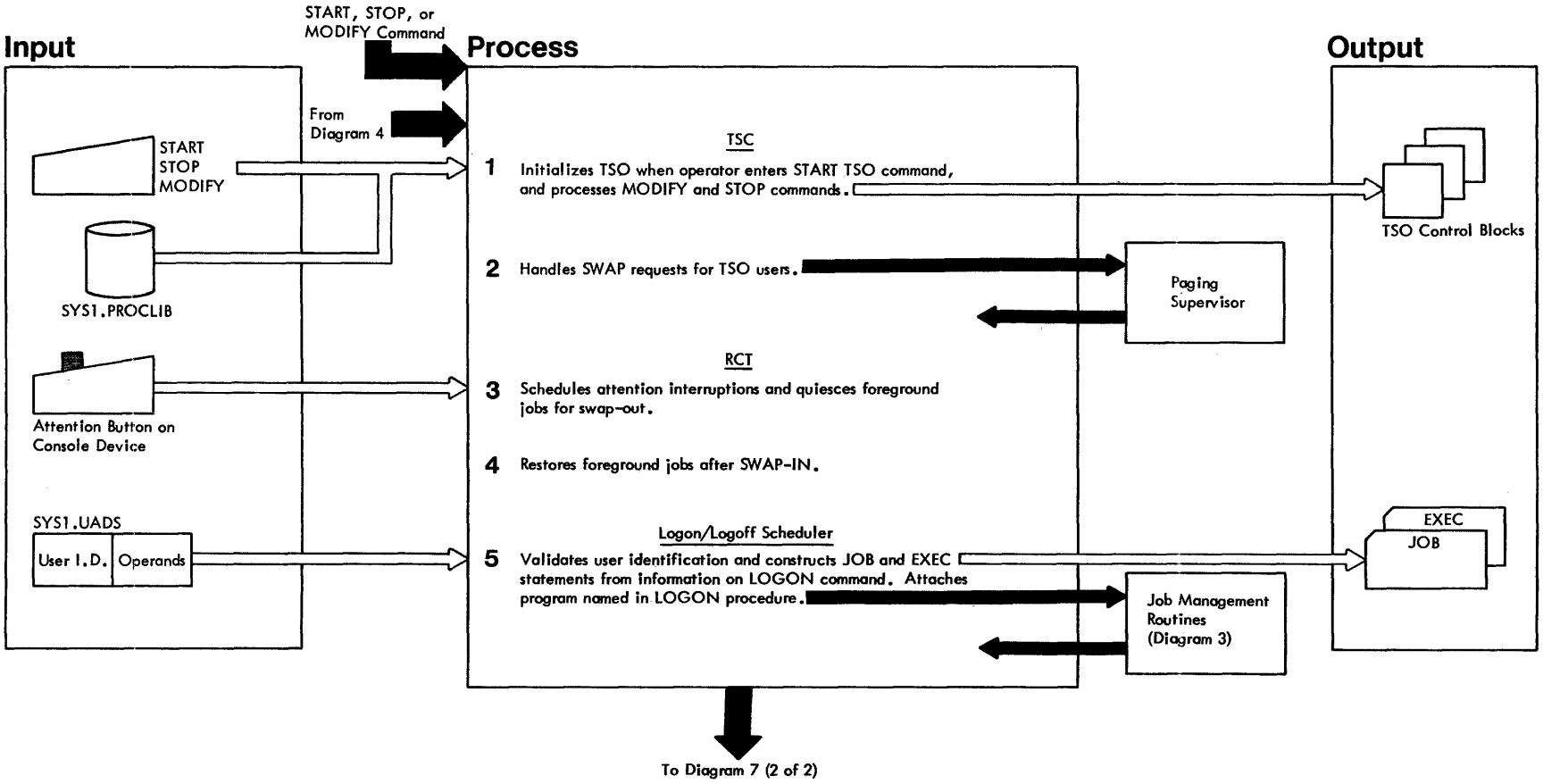


Diagram 7. TSO (Part 1 of 2)

Diagram 7. Time Sharing Option

Item	Description	Reference	Index Word
1	The operator starts TSO with a START TSO command. The TSC (time sharing control task) receives control after the START command and obtains a foreground region by using VS2 task management routines (see Diagrams 1 and 4). The TSC also handles STOP and MODIFY commands, performing the requested functions.	TSO Control Program Logic	operator action
2	TSO does not use separate page data sets; rather, it relies on the paging supervisor for swapping TSO users to and from external page storage. The TSC interfaces with the paging supervisor and signals it to perform swapping.		paging
3	The TSC attaches an RCT (region control task) for each region. The TSC serves two purposes: scheduling attention interruptions, and quiescing and restoring foreground jobs. Before a foreground job can be swapped out, all I/O activity must be quiesced-stopped in an orderly manner-and control blocks must be removed from system queues. Since most control blocks for regions in OS/VS2 reside in the LSQA (local system queue area), they can be swapped out with the rest of the region after the foreground job has been quiesced.		RCT TSC
4	The RCT logically restores foreground jobs by reestablising pointers to control blocks and by reactivating intercepted I/O requests.		
5	The Logon/Logoff scheduler validates the user identification and builds JOB and EXEC statements from operands on the LOGON command. The Logon/Logoff scheduler passes these statements to reader/interpreter and initiator routines (Diagram 3) which allocate resources to the job the same as for batch jobs. This technique ensures compatibility between foreground and background jobs. The Logon/Logoff scheduler invokes the TMP (terminal monitor program).		Logon/Logoff scheduler

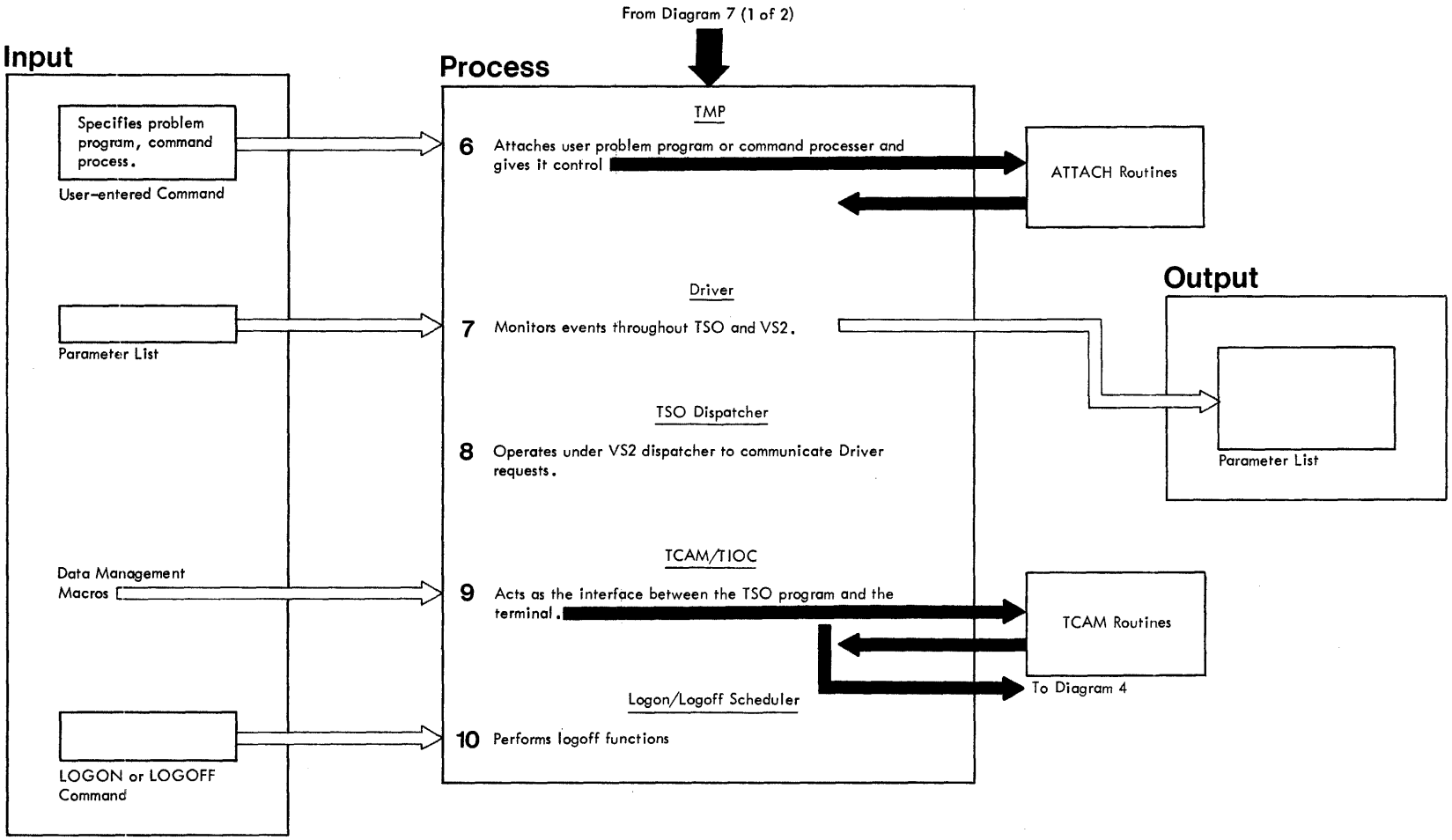


Diagram 7. TSO (Part 2 of 2)

Item	Description	Reference	Index Word
6	The TMP attaches problem programs or command processors requested by the terminal user. Once the attached foreground program gains control of the CPU, it has control for a set amount of time called a time-slice. This ensures efficient use of system resources by both background and foreground jobs. The Driver (Item 7) controls the length of the time-slice.		
7	The Driver receives information from other TSO routines and the control program, then uses this information to assess the workload of the system, and monitors efficient operation of TSO. One aspect is controlling the length of TSO time-slices.		driver
8	The TSO Dispatcher communicates the TSO Driver's requests for status changes in TSO to the TSC and the RCT. The TSO Dispatcher also rearranges the TCB ready queue to indicate the current priority of tasks in the system (time sharing tasks and background tasks) as determined by the Driver.		
9	TSO uses TCAM (telecommunications access method) to communicate with the terminal user. Several routines provide the function of initialization and I/O transfer: TIOC (terminal I/O coordinator) acts as the interface between TCAM and the terminal user of a problem program. TIOC routines move data between TSO buffers and problem program buffers. TSO subroutines of TCAM move data from TCAM buffers to TSO buffers for input processing, and vice-versa for output processing. TCAM message handler routines process messages to and from terminals. TCAM message handler routines consist of assembled TCAM macros. The TCAM Concepts and Facilities publication explains how to assemble message handler routines.	TCAM Logic TSO Control Program Logic	terminal communications
10	The Logon/Logoff scheduler is brought into a region when a LOGOFF command or when a second LOGON command is entered during the terminal session. The logoff routines terminate the user's session and perform statistics updating.		

Program Organization

This section shows the sequence of operation for a job in a typical batch job stream. To simplify basic concepts of VS2, the figure does not consider IPL and NIP, nor does it consider TSO processing. IPL and NIP processing and TSO processing are described in M. O. diagrams 2 and 7 respectively; the numbers in the lower right-hand corner of each block refers to the appropriate M. O. diagram.

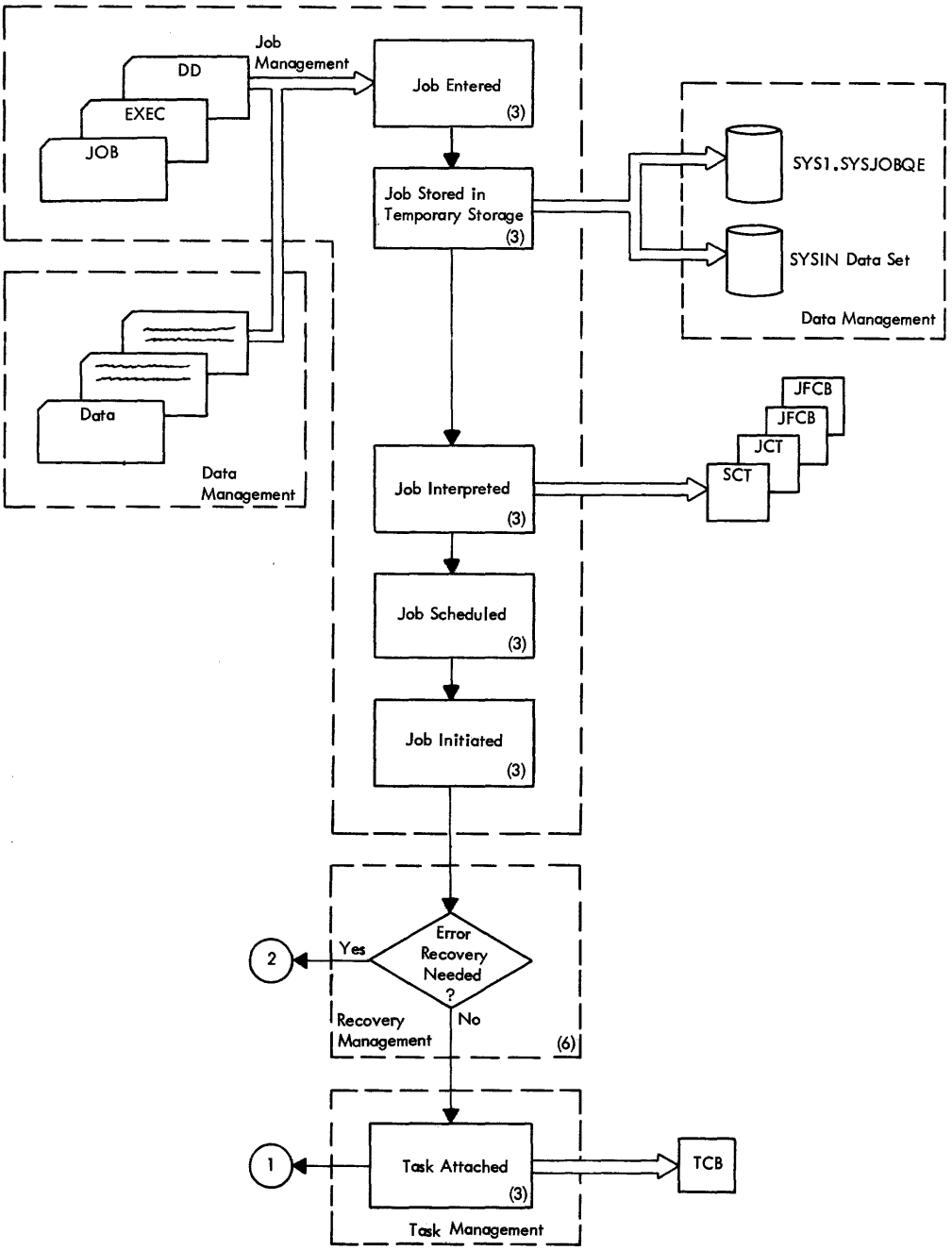


Figure 32. Sequence of Operation (Part 1 of 2)

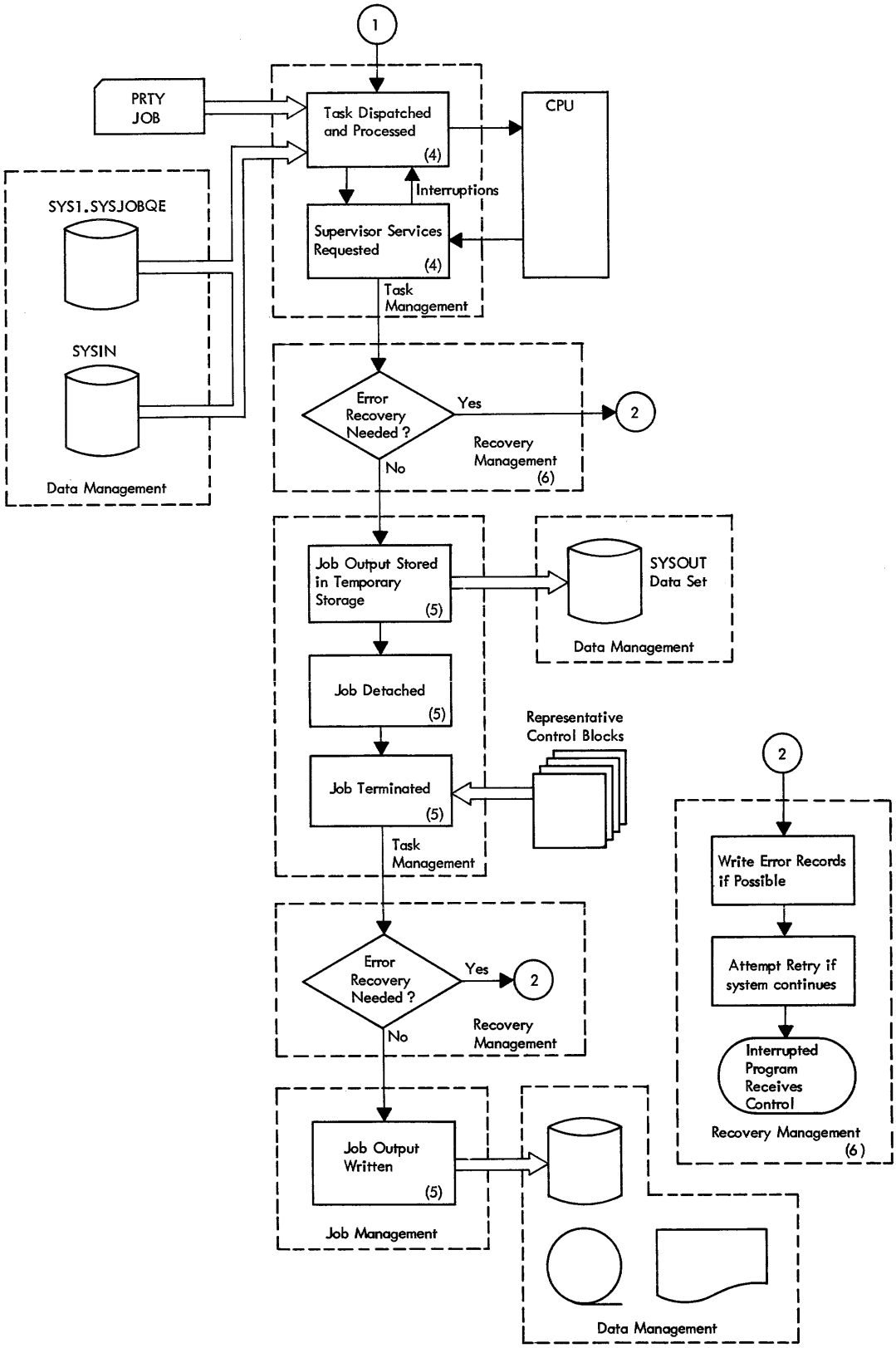


Figure 32. Sequence of Operation (Part 2 of 2)

System Communications

This section describes the purpose of several control blocks and tables used by the VS2 control program. The description of control blocks is not meant to be exhaustive. Control blocks are described in more detail in the *VS2 System Data Area* publication and in the "Data Areas" section of logic manuals.

System Control Blocks

The method of operation diagrams referred to various control blocks, such as the UCB and the DEB. Control blocks contain information about a particular system resource. For example, the UCB describes I/O devices and the TCB contains information about a task. In short, control blocks communicate information needed by the various components of the control program. Figure 33 lists and describes the following system control blocks:

- Communications Vector Table (CVT)
- Task Control Block (TCB)
- Task Input/Output Table (TIOT)
- Data Extent Block (DEB)
- Data Control Block (DCB)
- Unit Control Block (UCB)
- Input/Output Block (IOB)
- Event Control Block (ECB)
- Page Table (PGT)
- Segment Table (SGT)
- External Page Table (XPT)
- Page Frame Table (PFT)

Figure 34 illustrates the relationship between system control blocks.

CVT	The CVT provides a means of communication between nonresident routines and the control program nucleus. It points to two words of TCB addresses that locate the TCB of the active task.
TCB	The operating system keeps pointers to all information related to a task in a TCB. Diagram 3 illustrates the TCB queue, showing permanent TCBs and dynamic TCBs, and represents tasks attached and created with the ATTACH macro. The TCB contains pointers to other system control blocks. These control blocks contain such data as which I/O devices are allocated to the task, which data sets are open, and which load modules are requested.
TIOT	The TIOT contains pointers to control blocks used by I/O support routines. It points to the addresses of unit control blocks allocated to the task, the job and step name, the ddnames associated with the step, and the status of each device and volume used by the data sets. Job management constructs a TIOT for each task in the system.
DEB	A DEB describes a data set's auxiliary storage assignments and contains pointers to DCBs and UCBs. The DEB is created and queued to the TCB at the time a data set is opened. Each TCB contains a pointer to the first DEB on its that points to data sets which are opened for the task at a given time, what extents are occupied by open data sets, and where the DCB and UCB are located.
DCB	The DCB is the place where the operating system and the problem program store all pertinent information about a data set. The DCB points to the IOB. It may be completely filled by operands in the DCB macro instruction, or partially filled in and completed when the data set is opened, with subparameters in a DD statement and/or information from the data set label. The format of DCBs differs slightly for each of the access methods and device types.
UCB	The UCB describes the characteristics of an I/O device. One UCB is associated with each I/O device configured into a system.
IOB	The IOB is the source of information required by the I/O supervisor. It is filled in with information when an I/O operation is requested. The IOB points to the DCB associated with the I/O operation and to the channel command associated with a particular device.
ECB	The ECB is a 1-word control block created when an EXCP macro instruction is issued, initiating an asynchronous I/O operation. At the completion of the I/O operation, the access method routine posts the ECB. By checking this ECB, the completion status of an I/O operation can be determined. In all access methods, the ECB is the first word of a larger block, the data event control block.
PGT	The PGT converts the page number to a real storage address. Each page number in the PGT has a real storage address associated with it. The PGT consists of page table entries, which are created by the create page table routine.
SGT	The SGT translates the segment number to the correct page table number. Each segment number in the SGT has a page table number associated with it. VS2 uses two segment tables: one with a protection key of 0 for system programs; and the other with a protection key of one for problem programs. The SGT consists of segment table entries, which are created by NIP routines.
XPT	The XPT keeps track of every page on external page storage. For each PGT entry, there is a corresponding XPT entry. The XPT consists of external page table entries, which are created by the create page table routine.
PFT	The PFT contains a list of the page frames available in real storage, and keeps track of all of the virtual storage area available for paging. Each page frame starts on a 4K boundary and is 4K bytes long. To expedite paging operations, PFT entries contain a bit signifying whether or not that page was changed while in real storage. Pages that were changed are paged out to make room for pages needed for execution. Pages that were not changed are overlayed directly, since a copy of them exists on external page storage.

Figure 33. Description of System Control Blocks

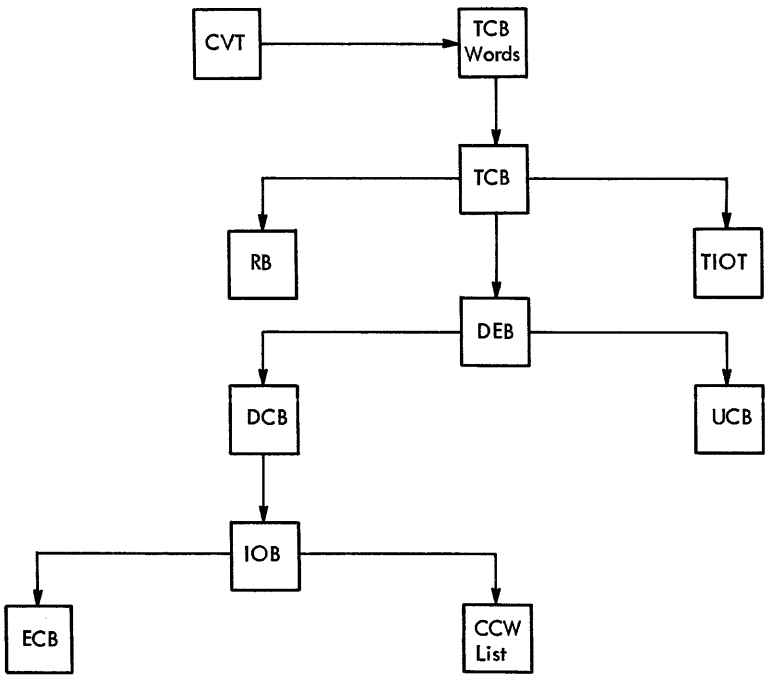


Figure 34. Relationship Between System Control Blocks

Request Blocks

Frequently, the routines that constitute a task are not all brought into the dynamic area with the first load module. Instead, they are requested by the task as it requires them. This dynamic loading capability necessitates using a control block to describe each load module associated with a task -- a request block (RB). An RB is created by the control program when it receives a request from the system or from a problem program to fetch a load module for execution. Figure 35 lists and describes these RBs used in VS2:

- Interrupt Request Block (IRB)
- Program Request Block (PRB)
- Supervisor Interrupt Request Block (SIRB)
- Supervisor Request Block (SRB)
- Task Interrupt Request Block (TIRB)

IRB	An IRB is created each time an asynchronous exit routine is executed. It is associated with an event that can occur at any time during program execution. The IRB is filled at the time the event occurs, just before control is given to the exit routine.
PRB	A PRB is created whenever an XCTL, ATTACH, SYNCH, or LINK macro instruction is issued. It maintains information concerning non-supervisory routines that must be executed to perform a task.
SIRB	An SIRB is similar to an IRB, except that it is associated only with IBM-supplied input/output error routines. SIRBs maintain information concerning I/O error handling routines. Associated error routines are fetched from the SYS1.SVCLIB data set.
SVRB	An SVRB is created each time a type II, III, or IV supervisor call is issued. This block is used to store information if an interruption occurs during execution of these SVC routines.
TIRB	A TIRB is created by the ATTACH routines for each task. The TIRB represents control program services that must be performed asynchronously.

Figure 35. Description of Request Blocks

With the possibility of many RBs subordinate to one task, it is necessary that queues of RBs be maintained. VS2 maintains an active RB queue. The active RB queue is a chain of request blocks describing active load modules and SVC routines. This queue can contain PRBs, SVRBs, IRBs, SIRBs, and under certain conditions, LPRBs. Figure 36 illustrates the active RB queue.

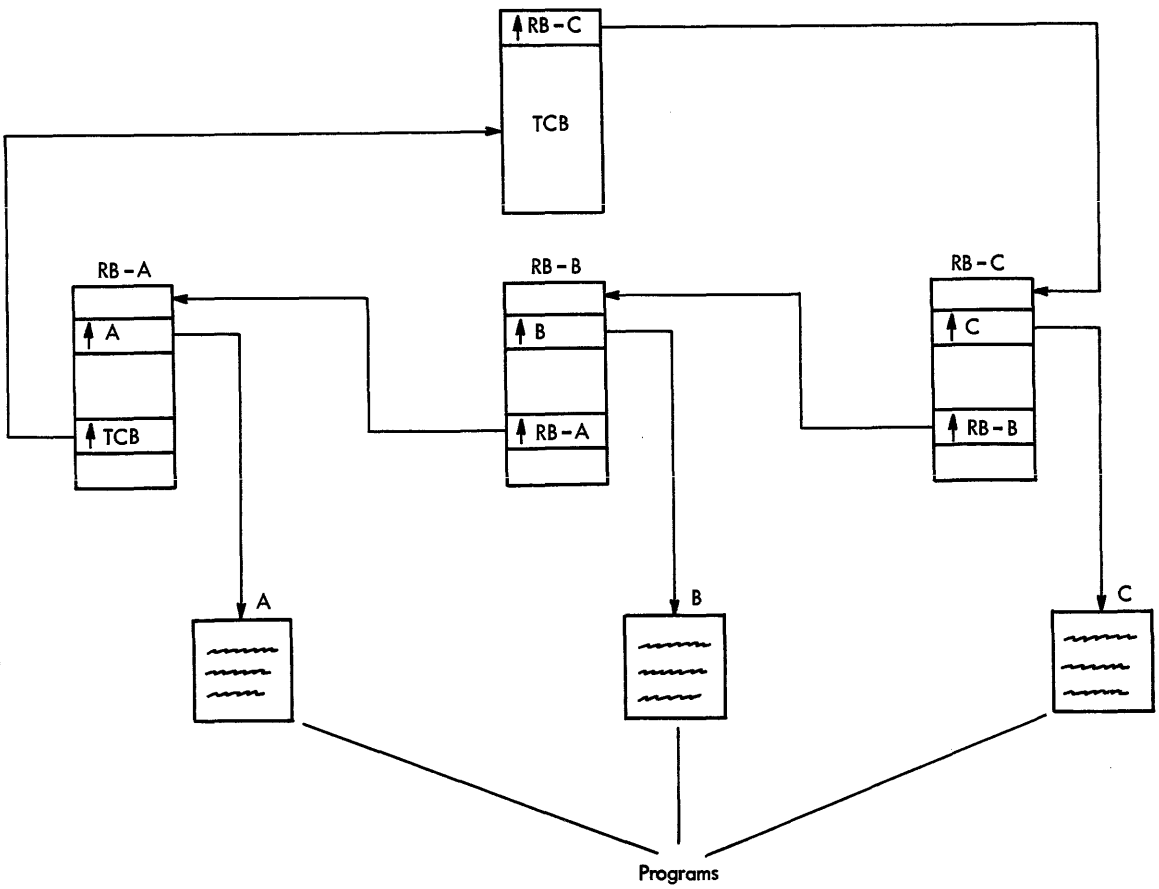


Figure 36. Active RB Queue

Contents Directory

The contents directory is made up of four separate queues: the link pack area control queue (LPAQ); the job pack area control queue (JPAQ); the load list, and the link pack area directory (LPD). Figure 37 describes the following queues used in VS2:

- Link Pack Area Queue (LPAQ)
- Job Pack Area Queue (JPAQ)
- Load List
- Link Pack Area Directory (LPD)

LPAQ	The LPAQ is a record of every program in the system link pack area. This area contains reenterable routines specified by the control program or by the user. The routines in the system link pack area can be used repeatedly to perform any task of any job step in the system. The entries in the LPAQ are contents directory entries (CDEs).
JPAQ	The JPAQ represents each job step in the system that uses a program not in the link pack area. The JPAQ, like the LPAQ, is made up of CDEs. It describes routines in a job step region. The routines in the job pack area can be either reenterable or not reenterable. These routines however, cannot be used to perform a task that is not part of the job step.
Load List	The load list is a chain of request blocks or LLEs (load list elements) describing load modules invoked by a LOAD macro instruction. The load list differs from the active RB queue in that RBs and associated load modules are not deleted automatically. They remain intact until a DELETE macro instruction deletes them, or job step termination occurs. The load list represents routines that are brought into a job pack area or are found in the link pack area. Each load list element is associated with a CDE in the JPAQ or the LPAQ; the programs represented in the load list are thus also represented in one of the other contents directory queues. Load list elements also contain a count field. Each time a LOAD macro instruction is issued for a load module already represented on the load list, the count is incremented by one. As corresponding DELETE macro instructions are issued, the count is decremented until it reaches zero.
LPD	The LPD contains a record of every program in the pageable link pack area. The LPD consists of link pack directory entries (LPDEs) that point to LPA programs. The LPD resides in the LPA, and it can be paged.

Figure 37. Description of Contents Directory Queues

Dynamic Area Control

Storage control information is kept in a series of control blocks called queue elements. Figure 38 describes the following queue elements used in VS2:

- Partition Queue Element (PQE)
- Free Block Queue Element (FBQE)

PQE	The PQE associated with a region resides in the system queue area. PQEs head a chain of FBQEs that describe free space in a region. PQEs are connected to the TCBs for all task s in the job step through a dummy PQE located in the system queue area. Virtual storage supervisor routines create PQEs.
FBQE	The FBQEs chained to a PQE reflect the total amount of free space in a region. Each FBQE is associated with one or more contiguous 4K blocks of free storage area. FBQEs reside in the lowest part of their associated area. As the amount of free space within the region changes, FBQEs are added to and deleted from the the free block queue. Virtual storage supervisor routines create FBQEs.

Figure 38. Description of Dynamic Area Queues

Subpool Storage Control

The virtual storage in a region used by problem programs and system programs is assigned by one or more numbered subpools. (Subpools can also be shared by tasks.) Subpools are designated by a number assigned to the area through a GETMAIN macro instruction. Subpool numbers available for problem program use range from 0 to 127. Subpool numbers 128 through 255 are either unavailable or used by system programs. Figure 39 lists and describes the following queues used in VS2:

- Subpool Queue Element (SPQE)
- Descriptor Queue Element (DQE)
- Free Queue Element (FQE)

SPQE	SPQEs describe the subpools created for a task. SPQEs reside in the system queue area and are chained to the TCB that uses the subpool. They serve as a link between the TCB and the descriptor queue (made up of DQEs), and may be part of a subpool queue if the task uses more than one subpool. If a subpool is used by more than one task, only one SPQE is created. Virtual storage supervisor routines create SPQEs.
DQE	DQEs pointed to by each SPQE reflect the total amount of space assigned to a subpool. Each DQE is associated with one or more 4K blocks of storage set aside as a result of a GETMAIN macro instruction. Each DQE is also the starting point for the free queue (made up of FQEs). Virtual storage supervisor routines create DQEs.
FQE	The FQE describes a free area within the blocks described by a DQE. As area distribution with the set of blocks changes, FQEs are added to and deleted from the free queue. Virtual storage supervisor routines create FQEs.

Figure 39. Description of Subpool Queues

Load Module Storage Control

Each load module in the dynamic area is described by the following entries that tell how much space it occupies. Figure 40 describes the following elements used in VS2:

- Contents Directory Element (CDE)
- Extent List (EXLST)
- Allocated Queue Element (AQE)

CDE	The contents directory is a group of queues, each of which describes an area of virtual storage. The CDEs in each queue represent the load modules residing in the associated area. There is a CDE queue for the link pack area and one for each region, or job pack area.
EXLST	The total amount of virtual storage occupied by a load module is reflected in an EXLST. EXLSTs contain the address of the virtual storage block. IDENTIFY, Laoder, and Program Fetch create EXLSTs.
AQE	The AQE points to allocated storage in the SQA (system queue area) or in an LSQA (local system queue area) with either the requesting task or the job-step task. Virtual storage supervisor routines create AQEs.

Figure 40. Description of Storage Control

Glossary

This glossary defines new OS/VS2 terms that are used in this publication. For definitions of terms not included in this glossary, see *IBM Data Processing Glossary,* GC20-1699.

address translation: The process of changing the address of a data item or an instruction from its virtual address to its real storage address. See also dynamic address translation.

automatic priority group: A group of tasks at a single priority level that are dispatched according to a special algorithm that attempts to provide optimum use of CPU and I/O resources by these tasks. See also dynamic dispatching.

basic control (BC) mode: A mode in which the facilities of a System/360 computing system and additional System/370 features, such as new machine instructions, are operational on a System/370 computing system. See also extended control (EC) mode.

BC mode: Basic control mode.

DAT: Dynamic address translation.

demand paging: Transfer of a page from external page storage to real storage at the time it is needed for execution.

disabled page fault: A page fault that occurs when I/O and external interruptions are disallowed by the CPU.

DSS: Dynamic support system.

dynamic address translation (DAT): (1) The change of a virtual storage address to a real storage address during execution of an instruction. See also address translation. (2) A hardware feature that performs the translation.

dynamic area: The portion of virtual storage that is divided into regions that are assigned to job steps and system tasks. See also pageable dynamic area, nonpageable dynamic area. Contrast with nondynamic area.

dynamic dispatching: A facility that assigns priorities to tasks within an automatic priority group to provide optimum use of CPU and I/O resources.

dynamic support system (DSS): An interactive debugging facility that allows authorized maintenance personnel to monitor and analyze events and alter data.

EC mode: Extended control mode.

enabled page fault: A page fault that occurs when I/O and external interruptions are allowed by the CPU.

extended control (EC) mode: A mode in which all the features of a System/370 computing system, including dynamic address translation, are operational. See also basic control(BC) mode.

external page storage: The portion of auxiliary storage that is used to contain pages.

fixed: Not capable of being paged out.

fixed BLDL table: A BLDL table that the user has specified to be fixed in the lower portion of real storage.

fixed link pack area: An extension of the link pack area that occupies fixed pages in the lower portion of real storage.

fixed page: A page in real storage that is not to be paged out.

frame: Same as page frame.

link pack area (LPA): An area of virtual storage containing reenterable routines that are loaded at IPL time and can be used concurrently by all tasks in the system.

local system queue area (LSQA): One or more segments associated with each virtual storage region that contain job-related system control blocks.

LPA: Link pack area.

LSQA: Local system queue area.

nondynamic area: The area of virtual storage occupied by the resident portion of the control program (the nucleus and the link pack area). Contrast with dynamic area.

nonpageable dynamic area: An area of virtual storage whose virtual addresses are identical to real addresses; it is used for programs or parts of programs that are not to be paged during execution.

page: (1) A fixed-length block of instructions, data, or both, that can be transferred between real and external page storage. (2) To transfer instructions, data, or both between real storage and external page storage.

page data set: A data set in external page storage, in which pages are stored.

page fault: A program 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 nonpageable so that it remains in real storage.

page frame: A block of real storage that can contain a page. Synonymous with frame.

page-in: The process of transferring a page from external page storage to real storage.

page-out: The process of transferring a page from real storage to external page storage.

page table (PGT): A table that indicates whether a page is in real storage and correlates virtual addresses with real storage addresses.

page translation exception: A program interruption that occurs when a virtual address cannot be translated by the hardware because the invalid bit in the page table entry for that address is set. Synonymous with page fault. See also segment translation exception, translation specification exeception.

pageable dynamic area: An area of virtual storage whose addresses are not identical to real addresses; it is used for programs that can be paged during execution.

paging: The process of transferring pages between real storage and external page storage.

paging device: A direct access storage device on which pages (and possibly other data) are stored.

paging supervisor: A part of the supervisor that allocates and releases real storage space (page frames) for pages, and initiates page-in and page-out operations.

PER: Program event recording.

primary paging device: An auxiliary storage device that is used in preference to secondary paging devices for paging operations. Portions of a primary paging device can be used for purposes other than paging operations.

program event recording (PER): A hardware feature used to assist in debugging programs by detecting program events.

real address: The address of a location in real storage.

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.

secondary paging device: An auxiliary storage device that is not used for paging operations until the available space on primary paging devices falls below a specified minimum. Portions of a secondary paging device can be used for purposes other than paging operations.

segment: A continuous 64K area of virtual storage, which is allocated to a job or system task.

segment table (SGT): A table used in dynamic address translation to control user access to virtual storage segments. Each entry indicates the length, location, and availability of a corresponding page table.

segment translation exeception: A program interruption that occurs when a virtual address cannot be translated by the hardware because the invalid bit in the segment table entry for that address is set. See also page translation exception, translation specification exception.

SQA: System queue area.

swapping: In VS2 with TSO, a paging technique that writes the active pages of a job to external page storage and reads pages of another job from external page storage into real storage.

system queue area (SQA): An area of virtual storage reserved for system-related control blocks.

translation specification exception: A program interruption that occurs when a page table entry, segment table entry, or the control register pointing to the segment table contains information in an invalid format. See also page translation exception, segment translation exception.

virtual address: An address that refers to virtual storage and must, therefore, be translated into a real storage address when it is used.

virtual equals real (V=R) storage: Same as nonpageable dynamic area.

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 amount of auxiliary storage available, rather than by the actual number of real storage locations.

V=R storage: Same as nonpageable dynamic area.

Index

Indexes to OS/VS publications are consolidated in the **OS/VS Master Index**, GC28-0602, and the **OS/VS Master Index of Logic**, GY28-0603. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

OS/VS2 Planning and Use Guide

GC28-0600-2

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

What is your occupation? _____

Number of latest Technical Newsletter (if any) concerning this publication: _____

Please indicate in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. Elsewhere, an

Cut or Fold Along Line

Cut or Fold Along Line

Your comments, please . . .

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

First Class
Permit 81
Poughkeepsie
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

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)

OS/VS2 Planning and Use Guide Printed in U.S.A. GC28-0600-2

OS/VS2 Planning and Use Guide Printed in U.S.A. GC28-0600-2

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)