GC24-5090-1

File No. S370-34

Systems

OS/VS1 Planning and Use Guide

VS1 Release 2

IBM

Preface

This publication has two objectives. The *Planning* portion provides guidance in selecting, planning for, and/or evaluating os/vs1 (virtual storage). The *Use* portion provides guidance in using some of the features or functions of the system. The *Planning* portion is for use by installation personnel who are familiar with IBM System/360 and who are considering the installation of an IBM System/370. The *Use* portion is directed to system programmers who maintain and/or extend the system.

The *Planning* portion contains two sections: *Concepts* and *Considerations*. The *Concepts* section briefly describes the functions, features, and device support provided by vs1. It also describes the operating principles of the system. The *Considerations* section contains suggestions concerning selection of various options of the control program and suggestions concerning operations in various modes (batch, teleprocessing, and graphics).

Appendix A contains a brief theory of operation of the system. A glossary of vs1 terms is also provided.

The reader of the *Planning* portion should be familiar with the *IBM System/370 System Summary*, GA22-7001. The *Introduction to Virtual Storage in System/370*, GR20-4260 will also be helpful, as will the *OS/VS Virtual Storage Access Method Planning Guide*, GC26-3799. For a complete list of available publications, see the *IBM System/360 and System/370 Bibliography*, GA22-6822.

The *Use* portion of the Guide contains information on implementing and/or extending some of the functions of the system. Each section in the *Use* portion is self-contained and deals with a separate capability or function of the control program.

The reader of the *Use* portion is assumed to have the prerequisite background in programming and system maintenance required to implement the procedures contained in the Guide.

Second Edition (January 1973)

This edition applies to Release 2 of OS/VS1 and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Any references that are made to OS/VS2 appear for planning purposes only. Changes are continually made to the information contained herein; before using this publication in connection with the operation of IBM systems, consult the *IBM System/360 and System/370 Bibliography*, GA22-6822, and the current SRL Newsletter for the editions that are applicable and current.

Information on the *Virtual Storage Access Method (VSAM)* and the *Dynamic Support System (DSS)* is included for planning purposes only until they are available. Consult your IBM Branch Office concerning the availability dates.

Level II of TCAM will not run under Release 2 of VS1. The TCAM information in this book is included for planning purposes until the availability of TCAM Level IV.

Summary of Amendments

For a detailed list of changes made in this edition, see page 3.

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

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 reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Programming Publications, Dept. G60, P.O. Box 6, Endicott, New York 13760. Comments become the property of IBM.

Summary of Amendments for GC24-5090-1 OS/VS1, Release 2

Use Guide Reorganization

The *Use* portion of the *Planning and Use Guide* has been rearranged. The sections are now placed in alphabetic sequence for easier access.

TCAM Compatibility Restrictions

Under *Compatibility, Restrictions* (planning portion), the documentation for TCAM object decks has been modified.

Level II of TCAM will not run under Release 2 of VS1. The TCAM information in this book is included for planning purposes until the availability of TCAM Level IV.

VS1 Supported Devices

Additional devices supported under VS1 are included in the planning portion.

Automated System Initialization

An overview of the automated system initialization feature is given in the planning section. A new section, *ASI*, is added to the *Use Guide.*

Remote Entry Services (RES)

A planning overview of RES is included in the planning portion. Utilization of Reader/Writer procedures by RES users is explained in section *PRO* of the *Use Guide*. Information on WTO/WTOR for RES users is included in section *SMI*.

Missing Interruption Checker (MIC)

A brief description of this program is given in the planning portion. Additional information is given under section *FEA* of the *Use Guide.*

V=R Area Specification

The topic *Virtual=Real Storage Availability* has been modified for V=R specification.

Automatic Partition Redefinition

In the planning portion of this manual, the topic *Partition Redefinition* has been updated. In the use portion, section *FEA* includes Automatic Partition Redefinition.

Pageable System Queue Area (PSQA)

Virtual storage area for PSQA is shown in the planning portion in the figures for storage configurations.

12K Dump Area

Virtual storage for a 12K dump area is shown in the figures for storage configurations in the planning portion.

Partition Deactivation/Reactivation

This new topic is included under *Operating Considerations* in the planning portion of this manual.

Page Boundary Loading

Loading on page boundaries is briefly discussed under *General Considerations* in the planning section.

Operator Commands

The new operator commands DUMP, MSGRT, and STOPMN are listed under the topic *Operator Commands.*

OS/MFT—OS/VS1 Differences

Section *DIF* has been updated to include additional MFT—VS1 differences.

Dynamic Dispatching

This optional feature is explained in section *FEA* of the *Use* portion.

Dynamic Support System (DSS) (for planning purposes only)

DSS information is included in section *FEA* of the *Use Guide.*

Greenwich Mean Time

This feature is explained in section *FEA* of the *Use Guide.*

User Modify Logical Cylinder

Using this option is explained in section *FEA* of the *Use Guide.*

ABEND Dump for Reader/Writer

DD statements for Reader/Writer ABEND dumps are included in section *PRO.*

Procedure INITD

Section *PRO* contains an updated INITD procedure.

Checkpointing SYSOUT Data Sets

Information on this topic is included in section *PRO.*

Resident Routines Option

Lists IEABLD00, IEAIGG00, and IEARSV00 (IBM-supplied standard lists for BLDL, RAM, and RSVC) have been modified in section RRO.

Output Separation

End of job output separators information has been added to section *SEP* of the *Use Guide.*

I/O Load Balancing

I/O load balancing for non-specific requests is included in the *Features and Options* (FEA) section.

DEB Validity Checking

A brief explanation of this is given in section FEA.

Message Routing Changes

Changes for multiple-line WTO and miscellaneous items have been made in section MSG.

Real Storage Restriction Removal

The *OLTEP not supported* has been removed for 144K systems.

Miscellaneous

Miscellaneous additions, improvements, and corrections have been made throughout this manual.

Contents

The Use Guide

Adding SVC Routines to the Control Program . SVC 1

How to Use the Tracing Routine . . . . TRC 1

The Time Slicing Facility . . . . TSL 1

Writing System Output Writer Routines . . . . . WWT 1

Appendix A: Theory of Operation . . . . APA 1

Glossary . . . . GLO 1

Index . . . . IND 1

VS1 provides support for IBM System/370 computing systems at the intermediate level. This support is comparable to that provided in the non-relocate environment by the System/360 Operating System with Multiprogramming with a Fixed Number of Tasks (OS MFT). Several enhancements are included in VS1 that make it a more effective and versatile operating system. Two of the more significant of these enhancements are *Virtual Storage* and *Job Entry Subsystem.*

Note: Throughout this manual, the suffix "K" denotes the value 400 (hexadecimal), or 1024 (decimal).

Virtual Storage

Virtual storage provides the user with a storage capacity of up to 16,777,216 bytes, which he can use to operate his computing system. (With the minimum 128K real storage system, the user is limited to two megabytes, or 2,097,152 bytes, of virtual storage.) The virtual storage is contained on auxiliary storage devices (direct access storage devices) in units of 2048 bytes, referred to as *pages*. These pages are transferred into and out of real storage (that part of the system storage from which the CPU can directly obtain instructions and data and to which it can directly return results) with relocation (address translation) to available areas of real storage, as they are needed by the system or by the user's programs. The process is called *paging.* Page-in obtains a page from auxiliary storage and page-out returns a page to auxiliary storage. This paging is handled by the VS1 control program and is completely transparent to the user. (Operating systems that do not support virtual storage are referred to as non-relocate systems.) For a comprehensive discussion of virtual storage and the paging process, refer to the IBM *System/370 System Summary,* GA22-7001, in the section titled *Virtual Storage* and to the *Introduction to Virtual Storage in System/370,* GR20-4260.

Advantages of VS1 Virtual Storage

- Jobs requiring more address space than the available real storage can execute in an VS1 environment.
- Real storage can be dynamically allocated on an as-used or as-required basis.
- Real storage that was unused, by partitions, in an OS MFT system can be recovered with VS1 and used by other partitions, because virtual address space in a partition does not require real storage until it is addressed.
- The small partition scheduling requirements of OS MFT do not exist in VS1 because virtual address space sufficient for scheduler requirements is provided for all partitions. Thus, no partition is forced to wait for scheduling by another.
- Programs with large design points can be tested on machines with smaller real storage. The performance of these programs will be directly related to their storage requirements versus real storage availability.
- Future processors can be written for VS1 with few storage restrictions.
- Infrequently executed system tasks do not require real storage to be permanently reserved for their use. These programs can be paged in on demand.
- Virtual storage can be reserved for unscheduled top-priority jobs.
- Partition redefinition requirement is reduced.

Job Entry Subsystem

The Job Entry Subsystem (JES) is the name of a centralized system facility that provides spooling and scheduling of VS1 primary input and output streams. Priority command routing and new operator commands simplify many operator tasks.

JES performs three basic functions:

1. All primary input streams are read from the input device and stored on a direct access storage device in a format convenient for later processing by the system and by user's programs.
2. System (and selected user) print and punch output is similarly stored on a direct access storage device until a convenient time for printing or punching.
3. If system resources are the objects of contention, JES schedules the activities to assure the highest degree of system availability.

The first two of the preceding functions are referred to as *spooling*. Spooling provides the following advantages:

- Non-sharable devices, generally unit record devices, are used at full rated speed if enough buffers are available.
- Non-sharable devices are used only for the time required to read, print, or punch the data.

Without spooling, the device is occupied for the entire time that a job is reading input or writing output. Thus, the device runs only as fast as the job can accept or generate data.

Because data is stored on a direct access storage device, jobs or their output can be processed in a different order from that in which they were submitted. This ability to control system work is called *job queueing*. Jobs can be scheduled by class, and by priority within class.

The ability to spool input and output and to queue jobs significantly improves system performance and configurability by centralizing and scheduling heavily used functions and by utilizing resources more efficiently. Other improvements, such as reduced system overhead, simpler operator procedures, and increased system availability are possible through the use of the Job Entry Subsystem. Certain portions of JES are more fully discussed in this publication in the *Concepts* section under the headings *Input Readers,* and *Output Writers.*

Compatibility

Current OS MFT object programs will execute in VS1 virtual storage with the exceptions of those noted here in *Restrictions*. Some programs may require virtual=real execution and will not use demand paging. (This is covered more fully in the *Concepts* section of this book under the heading *Virtual=Real Execution.*) Current OS data sets will process in the VS1 system without modification or conversion.

Sequentially organized DOS data sets (SAM) that are portable between DOS and OS MFT without conversion or other user intervention will be fully portable between DOS and VS1.

VS1 is upward compatible to VS2. This compatibility includes source program code, object program code, job control language, and conventions and standards. For a description of OS/VS1–OS/VS2 differences, refer to the *OS/VS2 Planning and Use Guide,* GC 28-0600.

Restrictions

Existing OS programs that will not operate under VS1 without modification include those that:

1. Are time-dependent.
2. Are written to deliberately cause program exceptions.
3. Use machine-dependent data.
4. Use the program status word (PSW) bit 12 (the ASCII bit).
5. Use low-address storage reserved for special purposes (PSW).
6. Depend on devices or facilities not supported or available in System/370 or VS1.
7. Require model-dependent System/360 functions.
8. Attempt to read or write SYSIN or SYSOUT data by other than the SAM access method (that is, EXCP will not work on these data sets).
9. Depend on a valid UCB pointer in the TIOT for SYSIN/SYSOUT data sets.
10. Include TCAM object decks. TCAM message control programs and TCAM message processing programs using the ICOPY, TCOPY, QCOPY, and TCHNG macro instructions must be reassembled and linkage edited. TCAM message processing programs not using any of these macro instructions need only to be re-linkage edited.

The Use part of this publication contains a resume of some of the more significant differences between OS/MFT and OS/VS1. This resume is the section *OS/MFT-OS/VS1 Differences.* It will be of particular interest to system programmers who are involved in an MFT to VS1 conversion.

Real Storage Restrictions

VS1 requires 160K bytes of available real storage to support the standard features listed under *Standard and Optional Features* in this publication. Lack of available real storage reduces the capabilities of VS1. For example, the following restrictions apply to the 128K real storage configuration:

- Only one partition is supported.
- The generalized trace facility, dynamic support system (DSS), and OLTEP are not supported.
- A maximum of two megabytes of virtual storage may be specified.

For the 144K real storage configuration, the following restrictions apply:

- Two-partition support is the recommended maximum.
- The external trace option of the generalized trace facility is not supported.

The Planning Guide

The *Concepts* section of this publication is intended for data processing executives responsible for selection of a system, and planners and system analysts who must understand its operation to plan for its efficient use, and to establish installation procedures. To accomplish this, the *Concepts* section presents a sample sequence of operation describing the initiation, execution, and termination of a set of jobs of various categories and priorities. This sequence of operation is followed by a description of the principles of operation of the VS1 system, describing the operation of each major component of the system.

The *Considerations* section contains information of interest to data processing executives, planners, and system analysts. It also describes VS1 considerations of interest to system programmers who will establish programming conventions for their installations, and machine room supervisors and operators who will establish operating procedures.

Information in the *Considerations* section is presented in these major topics.

1. *General considerations,* which apply to all types of jobs to be run under control of VS1.
2. *Batch considerations,* which apply to "batched" production jobs such as compilation, file maintenance, and report generation.
3. *Telecommunication considerations,* which apply to telecommunications message processing and message switching jobs under VS1.
4. *Graphics considerations,* which apply to jobs which involve the IBM 2250 or 2260 Graphic Displays.
5. CPO *considerations,* which apply to concurrent peripheral operations under VS1.
6. *Typical storage configurations.*
7. *Operating considerations,* which outline system characteristics or actions that are of special interest to the machine operator.

The Use Guide

The *Use Guide* consists of self-contained sections, each of which provides information on how to modify, extend, or implement capabilities of the VS1 control program. It is directed to system programmers who have the responsibility for maintenance of the system. It is assumed that readers of the *Use Guide* are thoroughly familiar with the design of VS1 and with its features.

Although the information in one section is sometimes related to information in another, or to information in the *Planning Guide,* all sections are written as separate and complete units. This organization has been used to reduce cross-referencing and to facilitate the addition of new sections.

Terminology

Unique terminology is explained as it is presented. Additionally, a *Glossary* is provided in the back of the book. Certain basic definitions, however, are essential to understanding the terms as they are introduced. These basic definitions are given in the following paragraphs.

Multiprogramming with a Fixed Number of Tasks

Multiprogramming refers to the concurrent execution of several units of work (tasks), any one of which would, in a single-program environment, occupy the computing system until the task is finished.

Note: Throughout this publication, job refers to an externally specified unit of work (a problem program specified by a JOB card), and task refers to any unit of work that must be performed by the Central Processing Unit (CPU), under control of a Task Control Block (TCB).

The significance of multiprogramming is that it provides increased throughput and better utilization of resources. A typical task makes use of only a small part of the resources available in the system. In a single-program environment, this means that overall application of resources is low. In a multiprogramming environment, however, resource application is markedly improved, because the relatively limited demands of each of several tasks combine to produce a net demand that is more efficient in terms of the system's capabilities.

The phrase "a fixed number of tasks" indicates that the maximum number of tasks the system is capable of performing at one time is determined at system generation. The number of tasks that can be performed at one time can be varied during and after system initialization.

System Initialization

System initialization is the preparation to execute those elements of OS/VS1 that reside in the pageable and non-pageable area of virtual storage. This preparation is performed by the Nucleus Initialization Program (NIP) when the system is brought into real storage through the initial program loading (IPL) procedure, and is supplemented by operator action.

Partitions

Virtual storage is divided into a pageable area and a non-pageable area. The pageable area is further divided, by the user, into a number of discrete areas called partitions. The number of tasks that can execute in each partition depends on whether or not the programmer has made use of the ATTACH facility.

When the ATTACH facility is not used, only one task will execute in each partition. The number of partitions defined determines the number of tasks that can execute concurrently.

When the ATTACH facility is used, each task can attach any number of subtasks (up to a maximum of between 196 and 249). Each subtask will then execute in the same partition as the attaching task.

The reason for the variation in the number of attachable sub-tasks involves the use of TCBS (task control blocks). Each attached sub-task requires a TCB. A maximum of 255 TCBS is available in the system. Of these, the system requires a minimum of 5. Also, each active partition (or task active in a partition) requires one TCB. In a single partition configuration, with minimum system options, six TCBS are not available for subtasking. This reduces the number available for subtasks to 249. As more system options and/or partitions are added, additional TCBS are required, up to a maximum of 59. This produces the lower limit of 196 available for subtasking.

Each partition has a fixed priority within the system. Partition 0 has the highest priority and partition 51 has the lowest priority. The priority determines which partition will gain control of the CPU first when a wait condition occurs.

The small partition, as it is defined under OS MFT (a partition too small to contain the scheduler), does not exist in VS1.

Concurrent Operation

In a multiprogramming system, tasks are performed *concurrently*. This is a significant programming concept. Execution is not simultaneous, or overlapped, or alternating in a fixed pattern. Each task is contained within a partition. The determination of which task gains control is based on "waits" and "posts". Waiting for an event, such as the completion of an input/output operation, removes the task from contention for control. Posting of the task, which signals that the awaited event has been completed, causes the task to be placed in a "ready" status. The task that becomes active is the highest priority task that is ready. This high-priority task proceeds until another event causes the task to relinquish control. The relinquishing of control by one task, and the gaining of control by another task, is called *task switch*.

Task Switching

There are two ways in which a *task switch* can occur:

1. The active task relinquishes control because it must wait for the completion of an event, such as an input/output operation.
2. Control is seized by a higher-priority ready task as a result of an interruption signaling an event for which it has been waiting.

The first case illustrates how multiprogramming ensures optimum utilization of resources. Whenever one unit of work cannot proceed, another (highest-priority) is advanced. In a single-program environment, no work can proceed while the single task waits for an event. The second case illustrates how an internal balance between the tasks is achieved. When a task has control, it retains control only until a task of higher priority becomes ready to proceed.

VS1 is an IBM System/370 Operating System option that provides extended multiprogramming capabilities and increased flexibility to the Operating System user whose system has 128K bytes or more of available real storage.

The system may reside on any of the following devices:

- IBM 2305-2 Fixed Head Storage Unit
- IBM 2314 or 2319 Direct Access Storace Facility
- IBM 3330 Disk Storage
- IBM 3333 Disk Storage and Control

Systems with VS1 can use the Shared Direct Access Storage Device (Shared DASD) feature. This feature allows two or more independently-operating computing systems to use common direct access storage devices. VS1 can share devices with other configurations of the IBM System/370 Operating System.

As many as 15 multi-step jobs can proceed concurrently with the operation of input readers, output writers, and as many direct system output writers as there are available devices. Each job is associated by job class with a discrete area of virtual storage known as a partition. Partitions are either problem program (maximum of 15) or system task partitions with a combined maximum of 52 partitions. With JES, readers and writers do not run in a partition, and consequently, the RDR/WTR classes of OS MFT are not required in VS1.

When a job must wait for completion of an event such as an input/output operation, another job of lower priority is allowed to proceed. When the higher-priority job is ready to resume, the lower-priority job's processing is suspended and control of the CPU is returned to the higher-priority job. The priority of each job is determined by the partition in which it resides. Jobs are directed to a given partition or group of partitions through the CLASS parameter of the JOB card.

By using the CLASS parameter to denote different *types* of jobs, the user can direct jobs to partitions consistent with the jobs' characteristics. Process-limited jobs, for instance, can be directed to low-priority partitions so that they do not interfere with efficient processing of jobs that do not require the CPU as often. Telecommunications jobs can be directed to higher-priority partitions so that the system response time to the terminal user is minimal. If equal intervals of CPU time are to be allotted to certain graphics or other jobs, these jobs can be directed to time-slicing partitions (if the time-slicing feature is included in the system). If direct system output writers are used, the job class of a job must be a job class that the writer can process. Direct system output writers can handle up to 15 different job classes. The CLASS parameter can be used to establish which jobs are going to be handled by direct system output writers. Additional applications of the CLASS parameter can be established, based on any job characteristics meaningful to the installation.

To use VS1 efficiently, both system and application programmers must understand how it operates. Because other user personnel may be interested in a summary of VS1 operation, without recourse to logic descriptions, this section of the publication contains these major topics:

1. *Minimum System Storage and Device Requirements* describes the minimum system configuration required to support VS1.
2. *Devices supported by* VS1.
3. *Features and Facilities* lists the functional capabilities of VS1 and describes each briefly.
4. *Sequence of Operation* describes the scheduling, initiating, and terminating of a series of jobs.
5. *Principles of Operation* describes in detail how each functional component of VS1 operates, and what it does.

Minimum System Storage and Device Requirements

A computing system using VS1 must have at least 128K bytes of available real storage and the following devices and features:

- Dynamic Address Translation (DAT)
- Standard multiplexor channel with associated input/output devices
- One selector and/or block multiplexor channel with associated input/output devices including an IBM 2314/2319 Direct Access Storage Facility, or an IBM 3330 Disk Storage, or an IBM 3333 Disk Storage and Control
- Storage protection
- Program event recording
- Monitor call

The system configuration must include at least three IBM 2314/2319 disk drives or two IBM 3330 disk drives.

If the shared direct access device feature is selected, the system must also include one IBM 2314 combined with a 2844 Auxiliary Storage Control Unit, or one of the following units equipped with a two-channel switch:

- IBM 2305-2 Fixed Head Storage Unit
- IBM 2314/2319 Direct Access Storage Facility
- IBM 3330 Disk Storage

or

- a 4-channel switch with the IBM 3330 Disk Storage
- a 4-channel switch with the IBM 3333 Disk Storage and Control

Devices Supported by VS1

VS1 provides support for the following hardware devices:

CPUs

IBM Model 135	Processing Unit
Model 145	Processing Unit
Model 155II	Processing Unit
Model 158	Processing Unit

DASD

IBM 2305-2	Fixed Head File
2314	Disk
2319	Disk
2835-2	Control Unit
2844	Control Unit
3330	Disk
3333	Disk Storage and Control
3830-1, -2	Control Unit
Integrated File Adapter for Models 135 and 145	

Tape

IBM 2401/2/3/4	Tape
2415	Tape
2420	Tape
2495	Tape Cartridge Reader
2671	Paper Tape Reader
2803	Control Unit
2804	Control Unit
2816	Tape Switch
2822	Paper Tape Control Unit
3410	Tape
3411	Tape
3420	Tape
3803	Control Unit

MICR/OCR

IBM 1275	OCR
1287	OCR
1288	OCR
1419	MICR (PCI adaptor required)

Displays/Consoles

IBM 1052-7	Console
2150	Console
2250-1, -3	Display/Console
2260-1, -2	Display/Keyboard
2265	Display
2840	Control Unit
2845	Control Unit
2848	Control Unit
3210	Console
3213	Console Printer
3215	Console
Display Console (for Model 3158)	

Printers

IBM 1403-N1, 2, 3, 7	Printer
1404-2	Printer (1403 support only)
1443-N1	Printer
2821	Control Unit
3211-1, -2	Printer
3811	Control Unit
Integrated Printer Adapter for Model 135	

Reader/Punch

IBM 1442-N1, 2	Reader/Punch
2501	Reader
2520	Reader/Punch
2540	Reader/Punch
2596	96 Column Reader (1442 support only)
2821	Control Unit
3505	Reader
3525	Punch

Transmission Control Unit

IBM 2701	Transmission Control Unit
2702	Transmission Control Unit
2703	Transmission Control Unit
2715	Transmission Control Unit
2772	Multi-purpose Control Unit
2955	Data Adapter Unit
3705	Communications Controller
7770	Audio Response Unit
Integrated Communications Adapter for Model 135	

Start/Stop Terminals

IBM 1030	Data Collection System
1050	Data Collection System
1060	Data Collection System
2721	Portable Audio Terminal
2740	Communication Terminal (Models 1 & 2)
2741	Communication Terminal (Model 1)
2760	Optical Image Unit
ATT 83B3	ATT Terminal
WU 115A	Teletype
World Trade Telegraph Terminals	
TWX 33/35 ATT Teletype Terminal	
IBM System/7 Processor Station	

Binary Synchronous Terminals

IBM 1130	Processor Station
1800	Processor Station
2770	Data Communications System
2780	Data Transmission Terminal
2790	Data Communications System
2972-8, -11	General Banking Station
System/3	Processor Station
System/360	Processor Station
System/360, Model 20	Processor Station
System/370	Processor Station
3270	Information Display System
3735	Programmable Buffered Terminal

Console device support provided in VS1 is summarized in Figure 1. SCS represents Single Console Support and MCS represents Multiple Console Support. The numbers in parentheses under MCS indicate the maximum number of devices supported. Blank spaces indicate no support as a system console.

Console Type	System Support	System Console Support		
		SCS	MCS	MCS+DIDOCS
3215	X (current 1052)	X	X(32)	X(32)
3210	X (current 1052)	X	X(32)	X(32)
3213	X (current 1052)			
1403	X	X	X(32)	X(32)
3211	X			
2501	X			
2520	X			
2540	X	X	X(32)	X(32)
3505	X	X	X(32)	X(32)
2250	X			MOD 1, 3 X(32)
2260	X			MOD 1 X(32)
3277	X			X(32)
3158 *	X	X	X(1)	X(1)
2740	X (via BTAM)		X(32)	X(32)

* Display Console for Model 158 Processing Unit

Figure 1. VS1 Console Device Support

Features and Facilities

VS1 makes possible the concurrent execution of up to 15 separate jobs within a single computing system having only one central processor, while continuing to provide all other applicable services of the IBM System/370 Operating System. Other features of VS1 include:

- Automated Initialization.
- Virtual Storage (up to 16,777,216 bytes).
- Independent job scheduling.
- Virtual=real execution facility.
- System Management Facilities (SMF).
- Job/step CPU timing.
- Job step CPU time limiting.
- WAIT time limiting.
- Checkpoint/restart.
- Recovery Management Support
- Redefinition of partition sizes and characteristics during operation.
- System input readers.
- Reading of an input stream from an IBM 2314, 2319, 3330, or tape.
- System output writers.
- Direct system output writers.
- Restarting the system without losing enqueued jobs.
- Concurrent execution of tasks within a partition.
- Remote Entry Services (RES)

Some of the features are described briefly in the following paragraphs and explained in detail under the heading *Principles of Operation.*

Automated Initialization

This feature, when used, makes the initialization process more rapid and flexible. It can reduce the operator's role in the process to a single entry on the console. Flexibility comes from the use of the SYS1.PARMLIB data set (or card reader) to hold members (or control cards) that contain the system initialization parameters. By proper definition of the parameters, each initialization tailors the system to better meet the needs of the anticipated job mixture.

Before initialization, a system programmer enters the needed parameters in SYS1.PARMLIB members by using the IEBUPDTE utility. During initialization, the nucleus initialization program (NIP) requests the operator to "SPECIFY SYSTEM AND/OR SET PARAMETERS". To use automated initialization, the operator simply enters (via the console) a reference to the list of SYS1.PARMLIB members to be used. The list of members may itself be a member of SYS1.PARMLIB, or it may be a card deck. In this way, the operator's role is reduced to a brief response to a system message.

This implementation enables much more rapid initializations than others, which involve the operator's manually entering all the needed parameters via the console. Automated system initialization may be invoked at the user's convenience. That is, unless the operator uses the new keywords for automated system initialization, the manual entry procedure must be followed. This feature requires no changes in the system generation options, but the implementation of automatic start commands is slightly changed. Its use is more fully described in the *Use* portion of this guide under the heading *Automated System Initialization.*

Extended Multiprogramming Capabilities

VS1 permits 15 problem program jobs to operate concurrently in the system. Jobs are scheduled into partitions through use of the CLASS parameter on the JOB statement, in conjunction with the PRTY parameter. The CLASS parameter may designate one of the 15 available job classes, A-O. With the storage protection and fetch protection features, each of these jobs is protected from damage by other jobs, and the system areas are protected from all problem programs.

Independent Job Scheduling

All partitions are independent with respect to job scheduling and initiation. Jobs are scheduled into the first available problem program partition that services the corresponding job class. Jobs are initiated according to the PRTY parameter on their JOB cards.

Virtual = Real Execution Facility

The Virtual=Real execution facility permits the user to have real storage available for any of his programs that will not run in the normally paged environment of VS1. Real storage will be allocated, at job execution time, with real storage addresses equivalent to virtual storage addresses on a byte basis, for the equivalent amount of virtual storage he has defined for the job involved. Multiple virtual=real jobs can execute concurrently if enough real storage is available in the system to accommodate the jobs. Programs that will not run in a paged environment are identified, and the facility is discussed in more detail, in this section of the publication under the heading *Virtual=Real Execution.*

System Management Facilities (SMF)

SMF is optional with VS1. SMF collects and, optionally, records system, job management, and data management information, and provides control program exits to installation-supplied routines that can periodically monitor the operation of a job or job step.

Job/Step CPU Timing

Job/step CPU timing is a standard SMF feature with VS1. The amount of time that each job or job step has control of the CPU is calculated by task management routines. If SMF=BASIC is selected, this value is passed to the user-supplied accounting routine if one is provided; if SMF=FULL is selected, the value is passed to the SMF user exit routines as provided.

Job Step CPU Time Limiting

This feature allows the user to specify the maximum amount of time that a job step can use the CPU. However, if the SMF option is selected and a user exit routine is provided, this routine can extend the time limit so that processing can continue.

Wait Time Limiting

This feature suspends processing of a job step if the job step remains in a wait state for more than an established time limit. If the SMF option is selected, the installation can determine the time limit by providing a user exit routine that can extend the time limit.

Checkpoint/Restart

The checkpoint/restart facility provides an opportunity to restart a job that terminates abnormally due to a hardware, programming, or system error. The restart is permitted either at the beginning of a job step or at a checkpoint within a job step. In either case, the restart may be automatic or may be deferred until the job is resubmitted.

Recovery Management Support

Recovery management is a service provided in VS1 that overcomes the damaging effects that a computer, channel, or I/O device malfunction might otherwise have on a program in progress.

Partition Redefinition

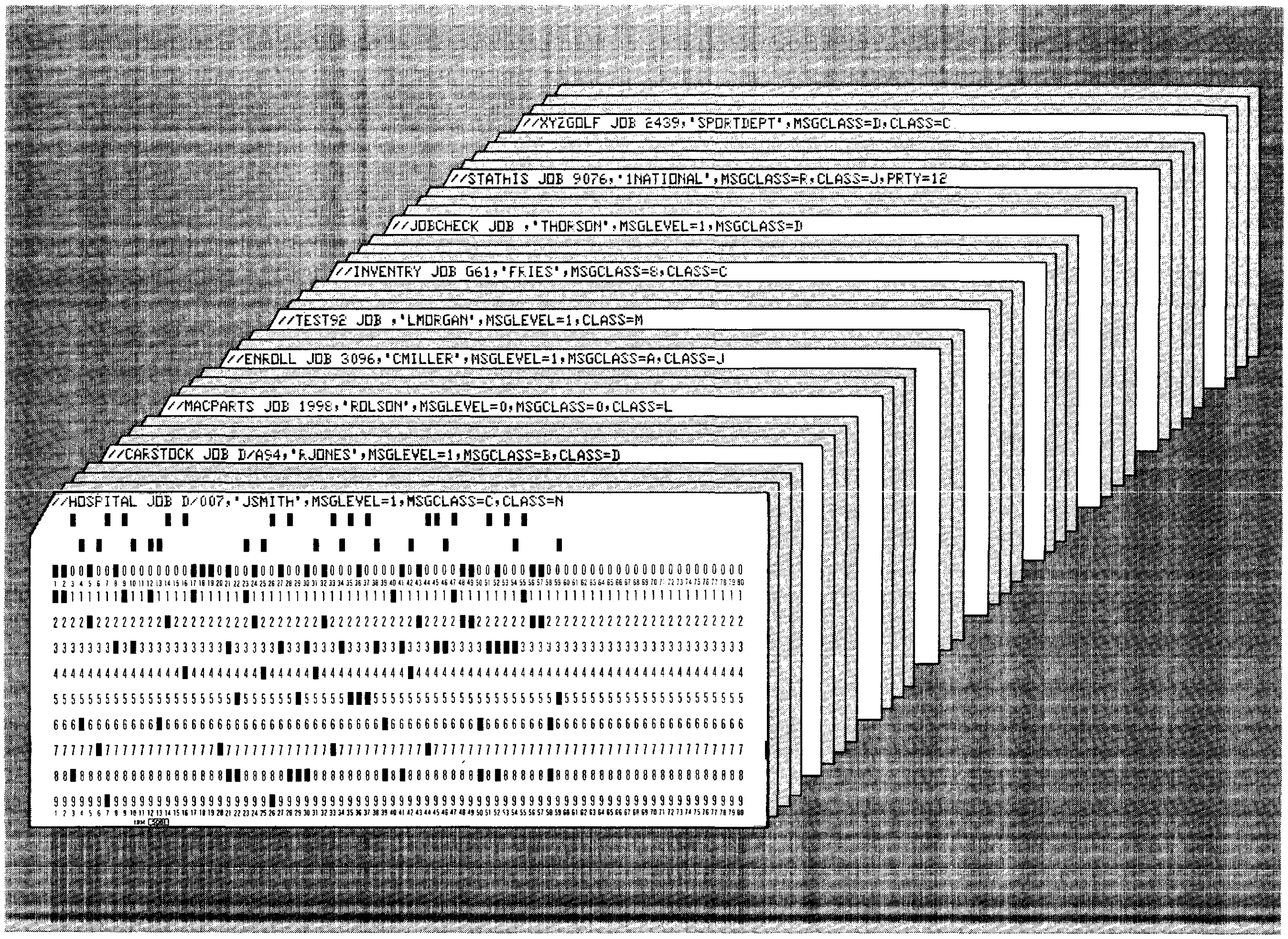
With the facility of partition redefinition, virtual storage can be reconfigured during system initialization or during operation, provided that the partitions to be redefined are contiguous. The partitions will quiesce automatically and require no intervention beyond the DEFINE command. The number of partitions in the system can be decreased, or increased within the limits established at system generation (SYSGEN). Partition attributes may be changed also, including the job class(es), sizes of the partitions, identification, deactivation/reactivation characteristics, and time-slicing attributes.

System Input Readers

System input readers, as they exist in OS MFT, do not exist in VS1. Instead, the Job Entry Subsystem (JES), residing in its own area of virtual storage, processes all system input, reading and spooling the input to the appropriate data sets. JES accepts job control language statements and data in the input stream, including multiple data sets for the same job step, and transcribes that data onto a direct access device for retrieval by the problem program. An input job stream on cards is illustrated in Figure 2. Input to JES from tape or disk can be blocked.

Input Stream from Disk

VS1 allows the user to establish a disk storage drive (IBM 2314, 2319, or 3330) as a system input device. Data in the input stream is permitted, including mul-



```
//XYZGOLF JOB 2439,'SPORTDEPT',MSGCLASS=D,CLASS=C
//STATHIS JOB 9076,'1NATIONAL',MSGCLASS=R,CLASS=J,PRTY=12
//JOBCHECK JOB ,'THORSON',MSGLEVEL=1,MSGCLASS=D
//INVENTRY JOB G61,'FRIES',MSGCLASS=8,CLASS=C
//TEST92 JOB ,'LMORGAN',MSGLEVEL=1,CLASS=M
//ENROLL JOB 3096,'CMILLER',MSGLEVEL=1,MSGCLASS=A,CLASS=J
//MACPARTS JOB 1998,'ROLSON',MSGLEVEL=0,MSGCLASS=0,CLASS=L
//CARSTOCK JOB D/A94,'RJONES',MSGLEVEL=1,MSGCLASS=B,CLASS=D
//HOSPITAL JOB D/007,'JSMITH',MSGLEVEL=1,MSGCLASS=C,CLASS=N
```

Figure 2. Input Job Stream

tiple data sets for the same job step, providing the facility for:

1. Reading input from sequentially organized data sets.
2. Deblocking of blocked input records.
3. Automatically switching volumes if end-of-volume is detected on a data set extending across volumes, or if concatenated data sets are being processed.
4. Starting more than one reader for the same disk storage unit.

System Output Writers

System output writers, resident in partitions as they are in OS MFT, do not exist in VS1. Instead, system output, with the exception of direct system output, is written by JES output writer(s), resident in the pageable supervisor area of virtual storage. Each output writer can handle as many as eight different output classes. More than one writer can service the same output class.

Direct System Output Writers

Direct system output writers are a job scheduler function. They enable a job's output data sets to be written directly to an output device during execution of the job.

Direct system output writers operate in problem program partitions. As many writers can operate in a partition as there are devices available. Each writer must be assigned to only one device and can process one system output class. Problem program output can be handled by a direct system output writer if the job class of the problem program is an eligible job class and the problem program is executing in the same partition as the writer.

System Restart

It is sometimes necessary to shut down the system while needed information still exists on the job queue data set (SYS1.SYSJOBQE). Such occasions might be end-of-shift, end-of-day, scheduled maintenance, system malfunction, or power failure. System restart permits all enqueued input and output jobs that had been entered in the job queue to remain there for subsequent retrieval by the system. Jobs that were being processed at the time of shutdown must be restarted.

VS1 Subtasking

VS1 allows the user to have any number of tasks (up to a maximum of between 196 and 249) executed concurrently within a partition. This feature provides the user with:

- An ATTACH function that can create subtasks.
- A DETACH function that removes completed tasks requested by the ATTACH routine.
- A CHAP function that allows a problem program to change dispatching priorities of the tasks within the partition.

Remote Entry Services (RES)

Remote entry services (RES) is an extension of JES and provides the remote terminal (work station) users with the same batch computing facilities that are available at the central installation. It allows jobs to be submitted from a work station and output to be routed back to the same work station and/or to other work stations. Besides accepting job input and routing output, RES facilities provide for sending messages between users and showing job, terminal, or work station status.

RES support consists of:

- Additions to existing JES components
- Modifications to the command processor for the commands LISTBC, LOGON, LOGOFF, ROUTE, and SEND
- The RES data sets SYS1.UADS and SYS1.BRODCAST
- The Remote Terminal Access Method (RTAM)
- Extensions to WTO and WTOR functions
- IKJRBBMP—a new standard utility used to create and build SYS1.UADS and SYS1.BRODCAST.

Except for RTAM, all changes to VS1 to support RES are always in the system. SYS1.UADS and SYS1.BRODCAST are required if remote users are to use the system. SYS1.UADS is also required if the central operator LOGON/LOGOFF function is to be used. Otherwise, these data sets may or may not be specified (if present, they are used).

RTAM is the only part of the RES package that is not present in any way in the system unless it is specifically included by a separate RTAM system generation. It is the only teleprocessing access method used by RES. RTAM resolves all work station, communication line, and transmission code dependencies. It is started on a communication line basis rather than on a terminal/work station basis.

To initiate processing, the central operator issues a START RTAM command, which, in turn, starts one or more communications lines. Additional lines may be started later by entering a MODIFY command specifying lines previously allocated to RTAM.

The priority of a job read in from a remote user may be determined by the QID entry built at IPL time. The QID table is built from the SYS1.UADS data set.

Jobs entered from remote work stations are placed on the system input queues. However, for each remote user, there are 36 output queues (classes A-Z and 0-9) and 1 output hold queue in addition to the system output and hold queues. This, of course, requires additional space on the job queue data set (SYS1.SYSJOBQE).

For details on RES and RTAM, including machine and additional space requirements, see the OS/VS1 RES *System Programmer's Guide,* GC28-6878.

Sequence of Operation

To illustrate the concepts of VS1, a sample sequence of operation is described here. (For an overview of the processing performed by various components of the control program, refer to *Appendix A.*)

In the job stream shown in Figure 2, the following CLASS parameters appear on the JOB cards:

1. CLASS=N
2. CLASS=D
3. CLASS=L
4. CLASS=J
5. CLASS=M
6. CLASS=C
7. None
8. CLASS=J,PRTY=12
9. CLASS=C

The system is loaded by use of the normal IPL procedure and initializes itself by use of the nucleus initialization program (Figure 3). After system initialization, the contents of virtual storage are as shown in Figure 4. The job class identifiers assigned to each partition at system generation are the alphabetic characters shown in the upper right corner of the partitions. Partitions 0 and 1 are shown as not active. Figure 5 illustrates the input work queues established at system initialization. When a START command is entered for the reader, the reader begins reading the job stream and entering jobs into the input work queues for each CLASS (Figure 6). The START commands for the initiator and writer are also entered.

The initiator in P4 now schedules the first job (Figure 7). Because N is the primary job class assigned to P4, the initiator searches the CLASS=N queue

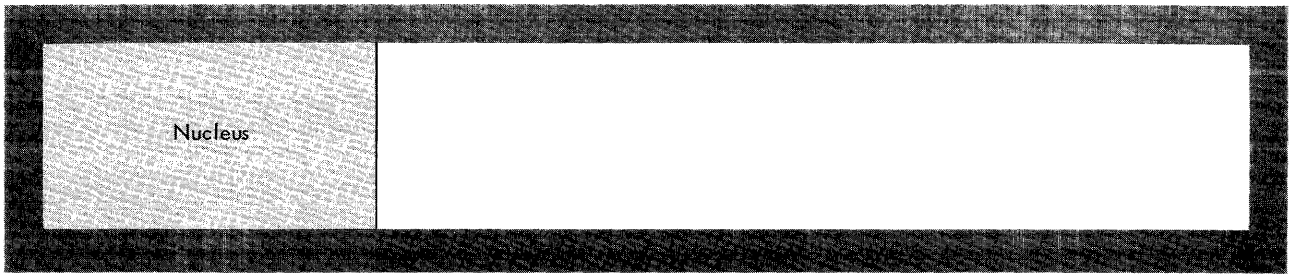


Figure 3. Contents of Virtual Storage after Nucleus Initialization

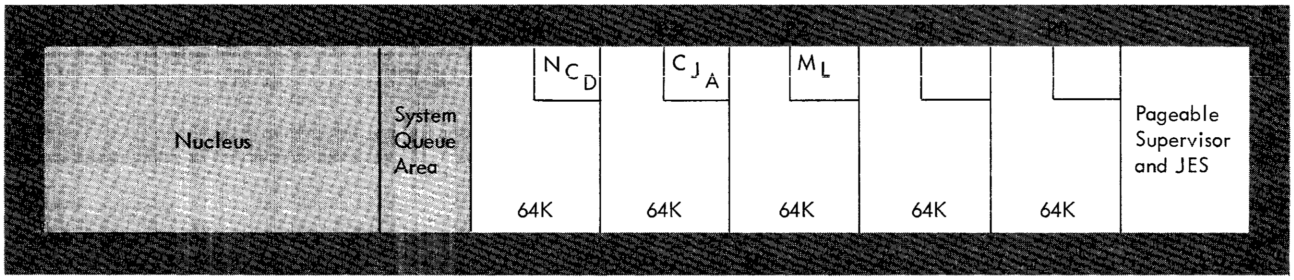


Figure 4. Contents of Virtual Storage after System Initialization

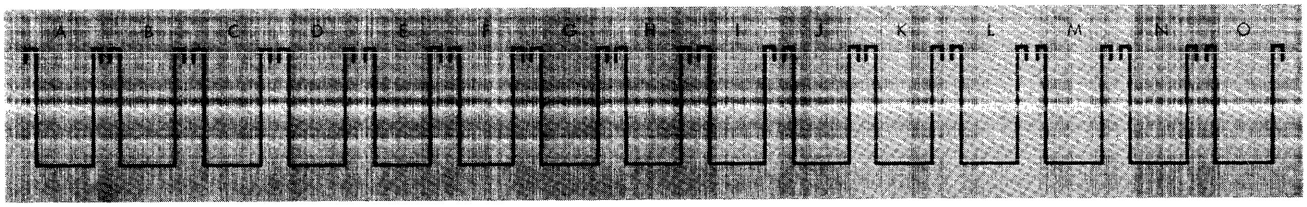


Figure 5. Input Work Queue after System Initialization

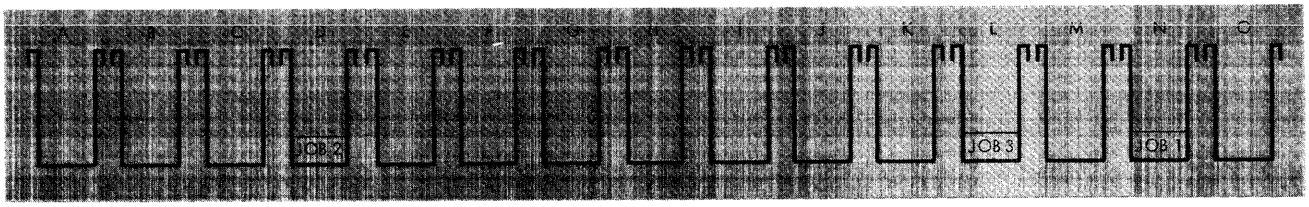


Figure 6. Input Work Queues after First Three Jobs Have Been Entered

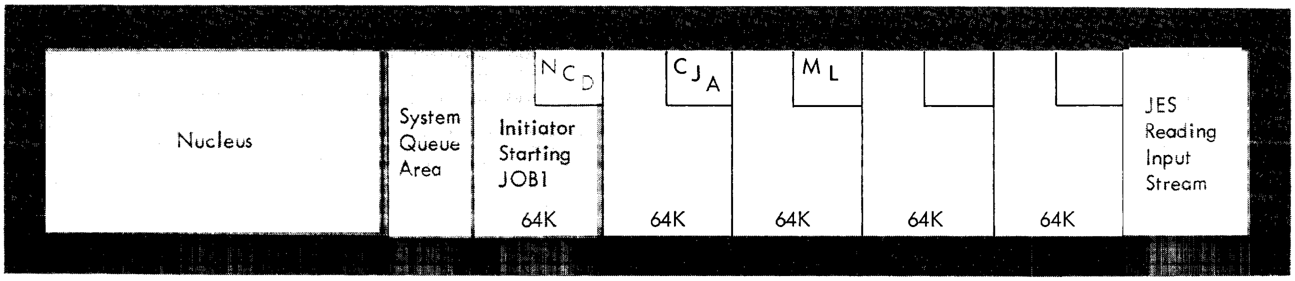


Figure 7. Contents of Virtual Storage after START Commands

first (job 1 has been placed on the queue by this time), and job 1 is initiated and given control (Figure 8). The reader continues reading the input stream, placing jobs in their appropriate queues. As soon as a job is completely read in, it may be scheduled if a partition servicing the particular job class is available. Job 3 will be scheduled into P2 (because L is the secondary class for P2) and job 4 will be scheduled into P3 (because J is the secondary class) when they have been read in. Because job 7 has no CLASS parameter, it is placed on input work queue A (the default job class). Job 8, with a PRTY parameter specifying priority 12, would be placed on input queue J ahead of job 4 if job 4 had not already been scheduled. At this point jobs 1 through 9 have been read and placed on their appropriate queues (Figure 9).

When job 1 has finished processing, a scheduler is brought into P4 to terminate the job. The scheduler now searches input work queue N for a job for P4. Because the CLASS=N queue is empty, the CLASS=C queue is searched. Job 6 is waiting on the queue; therefore, it is scheduled into P4. At this point, the contents of virtual storage are as shown in Figure 10.

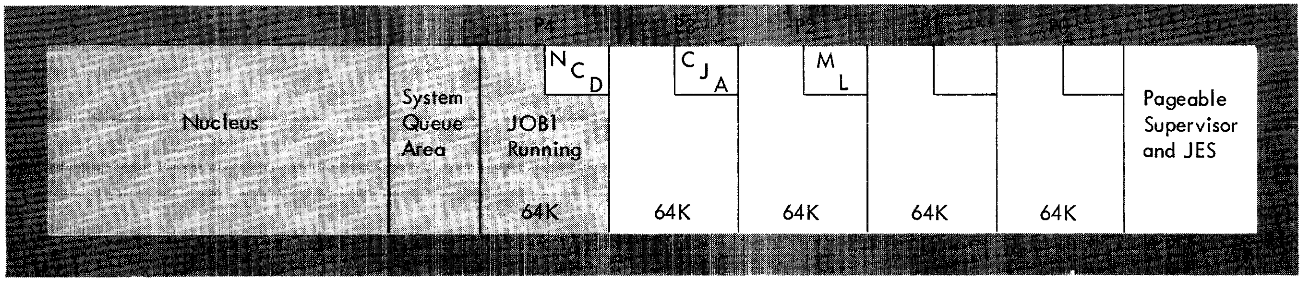


Figure 8. Contents of Virtual Storage after First Job has been Scheduled

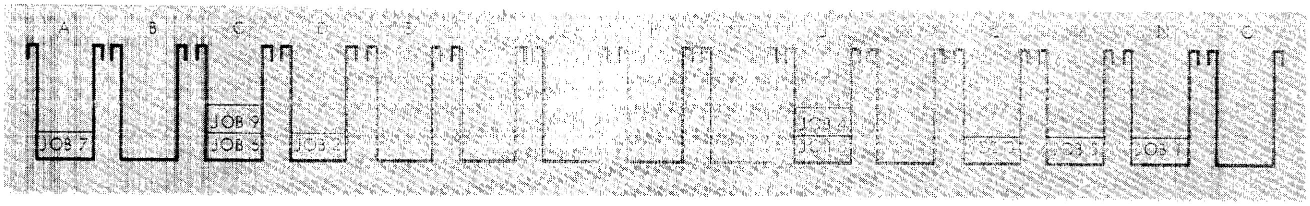


Figure 9. Input Work Queues after All Nine Jobs Have Been Entered

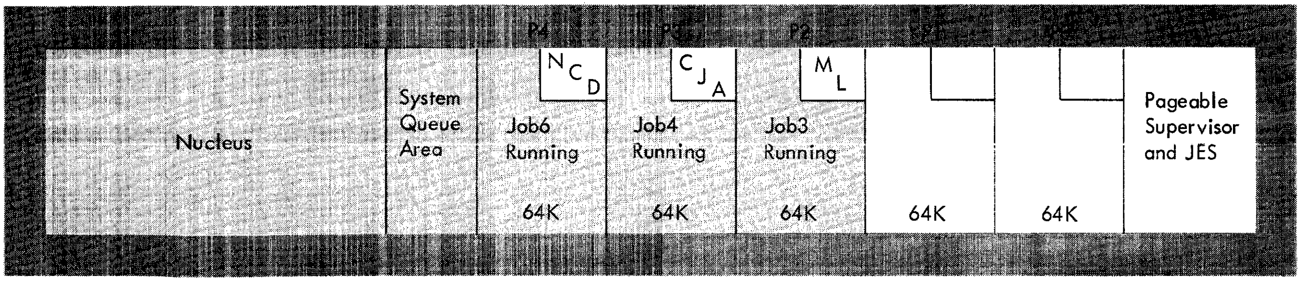


Figure 10. Contents of Virtual Storage with Three Partitions Active

Principles of Operation

This topic describes the principles of operation of VS1. Included are:

- Partition job class facility.
- Priority within job class.
- Job/step CPU timing
- Virtual storage organization.
- Checkpoint/restart.
- Partition redefinition.
- System input readers.
- Scheduling process.
- System output writers.
- Direct system output writers.
- System restart.

Figure 11 illustrates a configuration referred to throughout the remaining discussion of *Principles of Operation*. P/P denotes problem program partition.

Partition Job Class Facility

The partition job class facility allows one or more partitions to be assigned to selected jobs. During system generation, a partition must be assigned to service each job class that will be used. These assignments may be modified later. (See *Partition Redefinition* in this section.) Each problem program partition may be assigned as many as 15 job classes designated by the alphabetic characters A through O. These job class designations have no inherent meaning; they can be used to denote any job characteristic, which would influence the choice of partitions for the job, meaningful to the installation. More than one partition may be assigned to service the same job class(es). In Figure 11, P3 is assigned to job classes C, J, and A; P4 is assigned to N, C, and D. These partition job class identifiers are used by the system to determine which input queue is searched first by an initiator servicing a specific partition. (See *Problem Program Partitions* in this section.) The sequence in which jobs are selected from each input work queue is determined by the PRTY parameter. (See *Enqueuing Jobs by* CLASS *and* PRTY in this section.)

The partition(s) in which a job executes is controlled by using the CLASS parameter on the JOB statement. The format of this keyword parameter is:

CLASS=job class

where job class is the alphabetic identifier (A-O) assigned to the job. If this parameter is omitted from the JOB card, a job class of A is assigned by the system. All 15 job classes may be used, provided at least one partition has been assigned to each of the classes specified. When a job class for a particular job is designated (by the CLASS parameter), the job is executed only in a partition that has been assigned to service that class. If more than one partition is assigned to service that job class, the job is executed in the first available problem program partition. A typical JOB card may be specified as follows:

```
//JOBPAY   JOB   661,'JDOE',CLASS=C,PRTY=12
```

In the configuration illustrated in Figure 11, this JOB card causes the job to execute in either P3 or P4, whichever is available first.

System Management Facilities

SMF is included at system generation by specifying SMF= in the SCHEDULR macro instruction. (This parameter replaces ACCTRTN= in OS MFT.) There are three options for SMF:

SMF=NOTSUPPLIED—specifies that the user does not want any SMF processing.

SMF=BASIC—specifies that user-written accounting routines are being provided. JES accounting information and two additional exits are also provided.

SMF=FULL—specifies that the full system management facilities routines are to be provided. VS1 provides two additional exits and additional JES accounting information.

The collected information is written onto data sets on direct access devices.

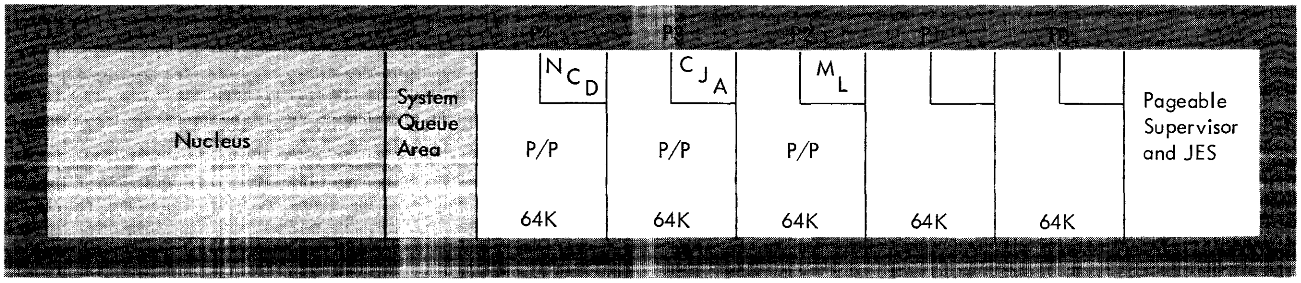


Figure 11. Sample Five-Partition Configuration

Job/Step CPU Timing

Job/step CPU timing is a standard SMF feature in VS1. CPU timings are calculated for each job step. This value is then passed, along with an accumulated value for the *entire* job, to a user-supplied accounting routine for further processing.

Note: The CPU timings include only the *active* time that the CPU has control of the job step. It does not include the *wait* time or the time used by the initiator and terminator for that job.

Functions of the Control Program with VS1

The control program routines of VS1 have four major functions: job management, task management, data management, and recovery management.

Job Management

Job management is the processing of communications from the programmer and operator to the control program. There are two types of communications:

1. Operator commands, which start, stop, and modify the processing of jobs in the system.
2. Job control statements, which define work being entered into the system.

Processing of these commands and statements is referred to as command processing and job processing, respectively.

Task Management

Task management routines monitor and control the entire operating system, and are used throughout the operation of both the control and processing programs. Task management has seven major functions:

- Interruption supervision.
- Task supervision.
- Virtual storage supervision.
- Page supervision.
- Contents supervision.
- Overlay supervision.
- Timer supervision.

The task management routines are collectively referred to as the "supervisor."

Data Management

Data management routines control all operations associated with input/output devices: allocating space on volumes, channel scheduling, storing, naming, and cataloging data sets, moving data between real and auxiliary storage, and handling errors that occur during input/output operations. Data management routines are used by processing programs and control program routines that require data movement. Processing programs use data management routines primarily to read and write required data, and also to locate input data sets and to reserve auxiliary storage space for output data sets of the processing program.

Data management routines are:

- Input/output (I/O) supervisor, which supervises input/output requests and interruptions.
- Access methods, which communicate with the I/O supervisor.
- Catalog management, which maintains the catalog and locates data sets on auxiliary storage.
- Direct access device space management (DADSM), which allocates auxiliary storage space.
- Open/close/end-of-volume, which performs required initialization for I/O operations and handles end-of-volume conditions.

The Virtual Storage Access Method, VSAM, is announced but not available. VSAM is an access method for use with direct access storage devices on IBM System/370 with VS. VSAM creates and maintains two types of data sets. One 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 is sequenced by the time of arrival of each record into the data set and is called an *event-sequenced* data set. Data records are located by using the records displacement from the beginning of the data set. The displacement is called the relative byte address (RBA). The RBA is 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. VSAM stores data records in a new data format designed for long term data stability and for data base applications. Data in both types of data sets can be accessed either sequentially or directly.

VSAM enhances many ISAM capabilities including device independence, concurrent processing, and the kinds of accessing supported. It provides additional password security protection. VSAM creates and maintains separate VSAM catalogs that contain specialized information about each VSAM data set and are used to link a data set with its index. VSAM includes a multifunction utility program that will define, delete, print, copy, and provide backup and portability of VSAM data sets and maintain the separate catalogs. An interface routine to allow most ISAM programs access to VSAM data sets is also provided. For a more detailed explanation of VSAM, see the *OS/VS Virtual Storage Access Method Planning Guide,* GC26-3799.

Recovery Management

Recovery management services are provided through the following programs:

Machine-Check Handler: This program processes machine-check interruptions. Depending on the severity of the malfunction, the machine-check handler (1) restores the system to normal operation, (2) terminates tasks associated with the malfunction so the system can resume processing, or (3) places the system in a wait state. In all cases, the machine-check handler program writes diagnostic messages and error records.

Channel-Check Handler: This program receives control after the detection of a channel data check, channel control check, or interface control check. For channel control checks and interface control checks, CCH:

- Indicates the results of the analysis of the error for later use by the error recovery procedures when they set up for a retry of the I/O operation.
- Constructs a record of the error environment. When this record is later recorded, a message is issued to inform the operator that a channel-detected error has been recorded on LOGREC.

For channel data checks, CCH constructs a record of the error. The error recovery procedures do not require information from CCH to retry I/O operations on which channel data checks occurred.

Dynamic Device Reconfiguration: This program, upon receiving a request from the operating system or from the operator, permits a demountable volume to be moved from one device to another and repositioned if necessary. This method is used to bypass various I/O errors, and is done without abnormally terminating the affected job or reperforming an initial program load (IPL).

Alternate Path Retry: This program allows an I/O operation that has developed an error on one channel path to be retried on another channel path, if another channel path is assigned to the device performing the I/O operation. Alternate path retry also provides the capability to vary a path to a device online or offline.

Missing Interruption Checker (MIC): This program polls active I/O operations to determine if a channel end and/or device end interruption has been pending for more than an installation-specified period of time. Also, it provides a reminder message for mount requests. For more detailed information, see the *Features and Options (FEA)* section in the *Use* portion of this manual.

Control Program Organization

The control program is on auxiliary storage in three partitioned data sets created when the system is generated. These data sets are:

- The NUCLEUS partitioned data set (SYS1.NUCLEUS), which contains the Nucleus Initialization Program (NIP) and the resident portion of the control program.
- The SVCLIB partitioned data set (SYS1.SVCLIB), which contains nonresident SVC routines, nonresident error-handling routines, and the access methods routines.
- The LINKLIB partitioned data set (SYS1.LINKLIB), which contains other nonresident control program routines and IBM-supplied processing programs.

Resident Portion of the Control Program

The resident portion (nucleus) of the control program is in SYS1.NUCLEUS. It is made up of those routines, control blocks, and tables that are brought into storage at initial program loading (IPL) and are never overlaid by another part of the operating system. The nucleus is loaded into the non-pageable area of virtual storage.

The resident task management routines include:

- Interrupt supervision.
- Virtual storage supervision.
- Page supervision.
- Timer supervision.

They also include portions of the routines that perform:

- Task supervision.
- Contents supervision.
- Overlay supervision.

The resident job management routines are those routines of the communications task that receive commands from the operator, and the master scheduler task.

Communications Task: The communications task processes the following types of communication between the operator and the system:

- Operator commands, issued through a console.
- Write-to-operator (WTO) and write-to-operator with reply (WTOR) macro instructions.
- Interruptions caused when the INTERRUPT key is pressed, to switch from the primary console to an alternate console.

Master Scheduler Task: The master scheduler task processes job queue manipulation commands and partition definition. For example, a HOLD or DEFINE command is processed by the master scheduler task.

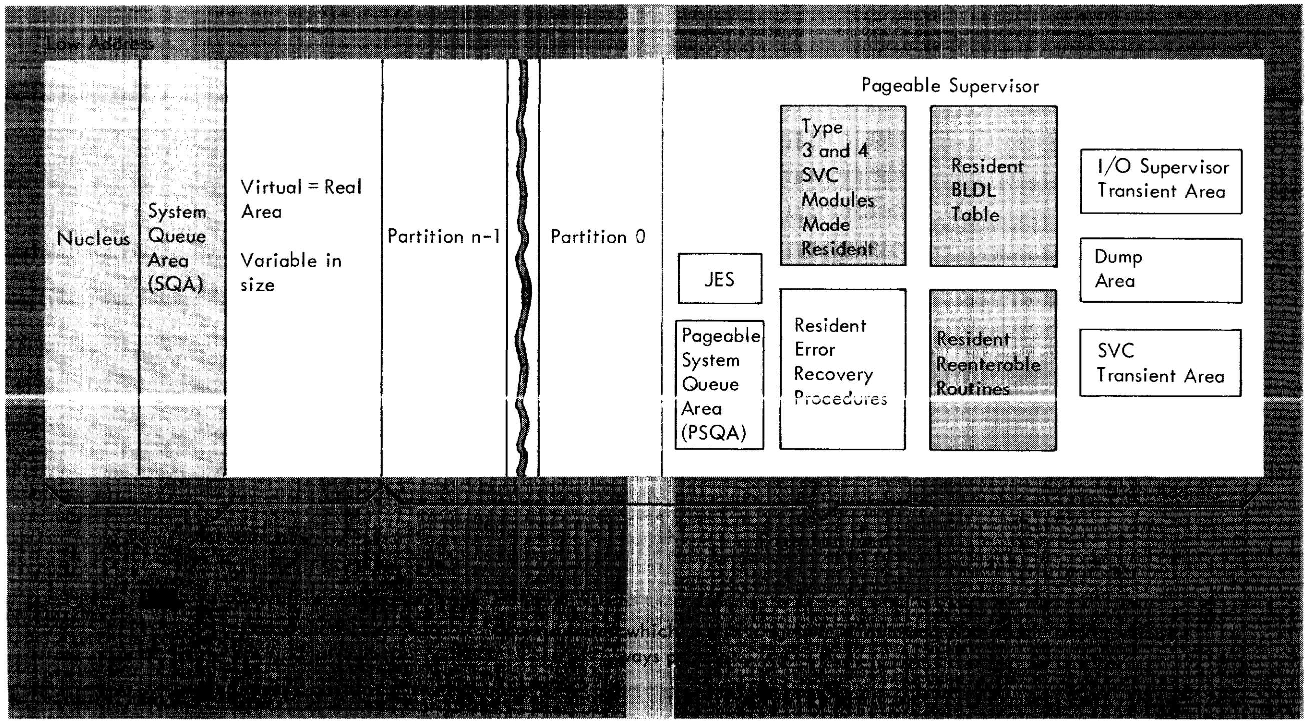


Figure 12. Virtual Storage Organization

The resident data management routines are the input/output supervisor and the BLDL routines of the partitioned access method.

Optionally, other access method routines may be made resident.

The user may also select resident reenterable routines, which are access method routines from SYS1.SVCLIB, and other reenterable routines from SYS1.LINKLIB. At system generation, the user specifies that he wants such routines resident in virtual storage. At IPL, he identifies the specific routines desired in the RAM=entry. The selected routines are loaded during system initialization and reside in the high end of virtual storage (see Figure 12). The resident BLDL table is standard. It contains directory entries for selected modules from the linkage or SVC libraries.

Normally transient SVC routines (that is, types 3 and 4 SVC routines) can be made resident through the RSVC option, specified by the user. NIP loads these routines adjacent to the resident reenterable routines. If there are no resident reenterable routines, the routines are loaded adjacent to the transient areas. (See Figure 12.)

Nonresident Portion of the Control Program

The nonresident portion of the control program comprises routines that are loaded into virtual storage as they are needed, and which can be overlaid after their completion. The nonresident routines operate from the partitions and from two sections of the pageable supervisor area called transient areas.

The non-resident routines which perform job management are collectively referred to as the scheduler.

Virtual Storage Organization

In a single task environment, virtual storage is divided into two areas: the non-pageable area, and the pageable area. In multiprogramming with a fixed number of tasks, the pageable area is divided further into as many as 52 discrete areas called partitions. Figure 12 shows the division of virtual storage.

The non-pageable area, located in the lower portion of virtual storage, contains the resident portion of the control program, and control blocks and tables used by the system. The size of the non-pageable area depends on the number of partitions established by the user, and the control program options selected at system generation.

The VS1 system nucleus occupies an area containing at least 54K bytes in virtual storage.

Partitions are defined within the pageable area, located in the upper portion of virtual storage, at system generation. The number of partitions may be varied within the number specified at system generation, and the sizes and job classes of partitions may be redefined at system initialization *or* during operation. Each partition may be occupied by a processing program, or by control program routines that prepare job steps for execution (job management routines), or that handle data for a processing program (access method routines).

Provided the total number of partitions does not exceed 52 and enough computing system resources are available, VS1 provides for the concurrent execution of as many as 15 problem programs. Each program is located in its own partition of virtual storage, with input readers and output writers, under control of JES, being located in the JES portion of the pageable supervisor area. The VS1 system provides for task switching among the tasks in the partitions, and between those tasks and the communications task and master scheduler task in the system area.

Non-pageable Area

The non-pageable area is that part of virtual storage into which the nucleus and SQA is loaded at IPL. The storage protection key of the non-pageable area is zero so that its contents can be modified by the control program only.

System Queue Area: The system queue area is located in the resident supervisor area of virtual storage. As SQA is needed, it is extended dynamically in both real and virtual storage. It contains ENQ/DEQ control blocks and command scheduling control blocks (CSCBS). In addition, if the communications task cannot obtain WTO buffer space, SQA is used.

The size of the system queue area is initially established at system generation (via the SYSQUE parameter of the CTRLPROG macro instruction). It can be modified at IPL time.

The system queue area also contains task related control blocks for each active subtask. In this case, the size of the system queue area is determined by: the number of partitions, and the number of subtasks that can be concurrently active. The size of the system queue area, established during system generation, should be retained.

The system queue area (SQA) is established by NIP adjacent to the fixed area and provides the virtual storage space required for tables and queues built by the control program. The SQA must be at least 4K bytes for a minimum one-partition system. Its storage protection key is zero so that it can be modified by control program routines only. Data in the system queue area indicates the status of all tasks.

Pageable Area

Figure 13 shows how the contents of each partition in the pageable area are organized and how they are related to the rest of virtual storage. Routines are brought into the high or low portion of a VS1 partition. Job management routines, processing programs, and routines brought into storage via a LINK, ATTACH, or XCTL macro instruction, are loaded at the lowest available address. The highest portion of the partition is occupied by the user parameter area and user save area. The next portion of the partition is occupied by the task input/output table (TIOT), which is built by a job management routine (I/O device allocation routine). This table is used by data management routines and contains information about DD statements.

Each partition may be used for a problem program as well as for system tasks. When the control program requires virtual storage to build control blocks or work areas, it obtains this space from the partition of the processing program that requested the space. Access method routines and routines brought into storage via a LOAD macro instruction are placed in the highest available locations below the task input/output table.

Working storage and data areas are assigned from the highest available storage in a partition.

The high portion of the pageable area is occupied by the pageable supervisor routines, the dump area, pageable SQA, the JES routines, and it also contains two transient areas into which certain nonresident routines are loaded when needed: the SVC transient area (2048 bytes) and the I/O supervisor transient area (1024 bytes). These areas are used by nonresident SVC routines and nonresident I/O error-handling routines, respectively, which are read from SYS1.SVCLIB.

Each transient area contains only one routine at a time. When a nonresident SVC or error-handling routine is required, it is read into the appropriate transient area. The transient area routines operate with a protection key of zero, as do other routines in this area.

System Input Readers

System input readers, resident in partitions as they exist in OS MFT, do not exist in VS1. Instead, system input is handled by the JES reader(s), resident in the pageable supervisor area of storage. For additional information concerning the JES reader, see the explanation under the heading *Input Readers* in this section.

Problem Program Partitions

VS1 permits up to 15 partitions to be specified for problem programs. Each partition may have up to 15 job class identifiers; more than one partition may be assigned to service the same job class(es). Problem program partitions must be at least 64K in virtual size.

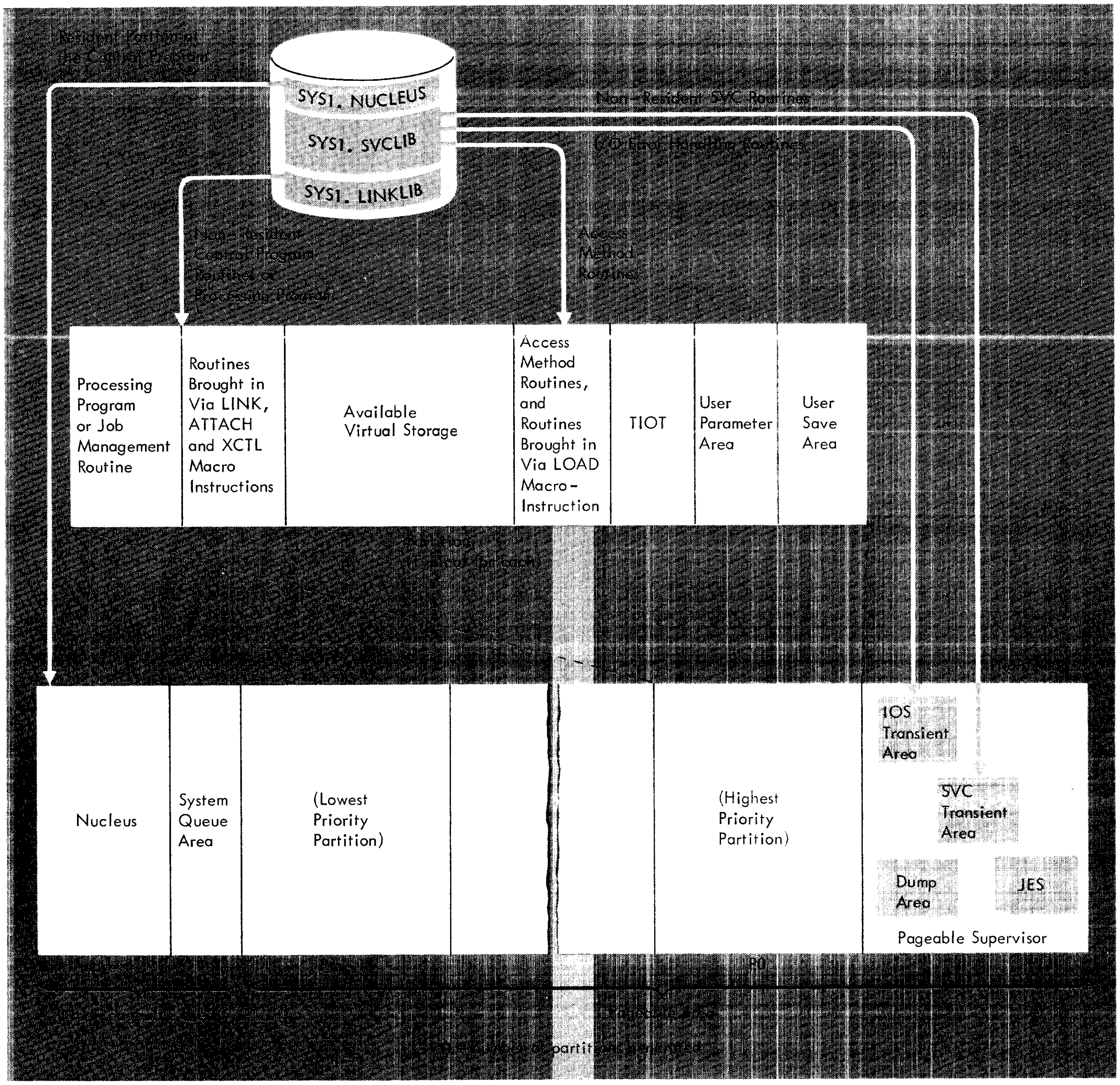

Resident Portion of the Control Program
SYS1. NUCLEUS
SYS1. SVCLIB
SYS1. LINKLIB
Non-Resident SVC Routines
I/O Error Handling Routines
Non-Resident Control Program Routines or Processing Program
Access Method Routines
Processing Program or Job Management Routine
Routines Brought in Via LINK, ATTACH and XCTL Macro Instructions
Available Virtual Storage
Access Method Routines, and Routines Brought in Via LOAD Macro-Instruction
TIOT
User Parameter Area
User Save Area
Partition (typical for each)
Nucleus
System Queue Area
(Lowest Priority Partition)
(Highest Priority Partition)
IOS Transient Area
SVC Transient Area
Dump Area
JES
Pageable Supervisor
P0
Pageable Area
partitions generated

Figure 13. Division of Virtual Storage

Problem programs run concurrently with system readers and writers. When a problem program in a partition is terminated, a scheduler is brought into that partition to retrieve another job from the input work queue for the appropriate job class(es). Control is then given to the appropriate task; for example, if a problem program is retrieved from an input queue, control is given to the program for execution.

For example, in Figure 11, P4 is assigned job classes N, C, and D. If a job of class N has just been terminated, the initiator first searches the job class N input work queue. If no class N jobs exist, the initiator searches for job class C jobs; if no class C jobs exist, the job class D input queue is searched. If all three queues are empty, the partition remains dormant until another job with class N, C, or D is read into the system and scheduled.

System Task Partitions

Up to 37 system task partitions are available in VS1. They are specified as system task partitions, by the user, by the C-S parameter of the PARTITNS macro instruction. They may be used in VS1 to ensure that partitions are available to:

- Start and stop readers and writers.
- Process mount procedures.
- Run conversational remote job entry readers.
- Run the generalized trace facility.

System task partitions are not required to run readers and writers in VS1 as they were in OS/MFT. Readers and writers in VS1 (other than direct system output writers) operate from the pageable supervisor area of storage.

Virtual=Real Execution

Real storage in VS1 consists of a non-pageable section and a pageable section. The resident supervisor, including the nucleus and the system queue area occupy the non-pageable area. The pageable area is occupied by pages of programs (user's and systems) which are brought into it for execution and are then returned to auxiliary storage. This is referred to as a *paged environment.*

Two types of programs exist that cannot run in a paged environment:

1. EXCP programs that modify the channel program while it is active, such as OLTEP.
2. Programs that are highly time-dependent, such as 1419 programs.

To permit these programs to operate under VS1, a facility is provided that permits the user to specify how much real storage he wants available for his programs in a non-paged environment. He does this by specifying ADDRSPC=REAL and REGION=nnnK, where nnnK is equal to the real storage required for his job, on the JOB or EXEC JCL statement. Storage will be allocated based on the REGION= parameter or, if REGION= is omitted, on the specification in his reader procedure. Storage will be allocated at job execution time as needed. Virtual addresses will still be processed by the dynamic address translation feature, but page tables will be built so that the translated addresses are the same as the untranslated addresses. Figure 14 shows the contents of virtual storage in a virtual=real situation. The problem program in partition 2 requires virtual=real storage.

Unit Record Applications: Because of the programming overhead in IOS due to virtual storage support, certain unit record applications may take longer in VS1 than in MFT. To avoid this situation the following applications can be run in a virtual=real (V=R) environment.

OCR Applications—1287-1288 programs which do not take advantage of JES run slower in a paged environment than they would under OS/MFT. Testing has indicated that OCR applications from OS/MFT have no appreciable degradation when run V=R on VS1.

Card Reader Applications—A job accessing a card reader and using the CNTRL macro may run slower. This is due to the fact that these applications add processing time to the system overhead. This can be avoided by using the V=R facility.

Unless you have these specific types of programs in your library, and have verified that the preceding situations are present, running unit record jobs in V=R mode is not recommended and will have an adverse effect on system performance.

Virtual=Real Storage Availability

The system default for the size of the V=R area is 512K or the real storage size of the machine, whichever is less. For systems larger than 512K, the user can specify a V=R area greater than 512K and less than or equal to the real storage size of the machine. He must specify this value at system initialization time in response to message IEA101A.

No firm rules or proven formulas exist for determining the amount of real storage that will be available for virtual=real execution in the system at a given time. This is due to the number of system operations that affect the availability of real storage space. However, the following considerations should help the user to understand the system use of real storage and to plan his work load and workflow accordingly.

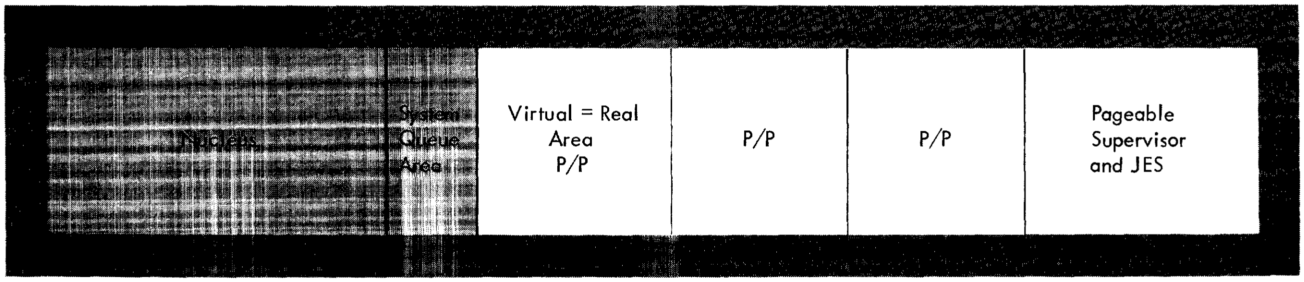


Figure 14. Contents of Virtual Storage with Virtual=Real Specified

- The size of the sysgened nucleus affects the amount of real storage available for V=R execution. The basic nucleus for a 128K real storage system operating one partition requires 64K. This includes 6K for Recovery Management Support (RMS) and 4K for System Queue Area (SQA). As more features, options, or support are sysgened in, the nucleus size increases and less real storage is available for V=R operations.

- When a request for V=R area is received, the system scans the V=R area for the requested amount of contiguous real storage. If an area is located that contains unused pages, the request for V=R space can be honored immediately. If a conditionally suitable area (containing no long-term fixed pages) is located, it is assumed to be conditionally available to honor the V=R request. Pages within this area are tagged for interception and subsequent use for V=R execution. When all pages in the area are no longer required for currently executing programs and have been paged out or relocated in real storage, the area becomes available and the V=R request can then be honored.

- Real storage for SQA, when specified at sysgen or NIP time, is appended to the nucleus. This real storage, like the nucleus and RMS area, is not available for V=R execution. If too little SQA has been specified, the system will dynamically extend the SQA, in increments of 2K bytes, as required. The extension will probably last until the next IPL. This extension further reduces the V=R area and may additionally cause real storage fragmentation, since the 2K extension will be located in what the system deems the optimum location. This location may not be the most optimum from the standpoint of V=R execution. As a worst case example, the extended SQA might be located in the middle of the V=R area, which would reduce the contiguous available storage in the area by half.

- Protected Queue Area (PQA) space also affects the amount of available V=R space. At initial partition definition time, a minimum of 2K of real storage is reserved for PQA for each partition defined. If additional PQA is required during processing, the system also extends the PQA in increments of 2K bytes. These increments are also placed in the optimum location from the system standpoint and this may cause the same sort of fragmentation as noted for SQA extension. In all events, the PQA is not available for V=R execution. As with SQA, the PQA extensions will probably last until the next IPL.

- PCI FETCH, if used, can cause a fixed PQA or SQA overflow. Each time a module must be brought into virtual storage, a 1600-byte work area is required and must be fixed. If the request is from a problem program, the work area is taken out of fixed PQA. If the request is from the system, the work area is taken from fixed SQA. Extensions of PQA and SQA are handled as previously described.

- JES (Job Entry Subsystem) requires one page of real storage to be long-term fixed. This page will be fixed right after NIP time and will remain for the duration of the IPL.

- Some components of the system cannot tolerate page faults during their operation. A page fault occurs when a "page not in real storage" is referred to by an active page. To eliminate the possibility of page faults occurring, these components may "long-term fix" required pages in real storage. These pages cannot be moved within real storage or paged out to auxiliary storage. These pages are fixed by the system in optimum locations (for system use) and may cause the same storage fragmentation and reduction of V=R area as previously noted. These pages remain fixed until they are freed by the component, at which time additional contiguous V=R area may become available.

- The system reserves an area of 36K that is used for EXCP I/O (short-term fixes) and paging. Two pages of this area are reserved as a "safety valve" in case PQA or SQA needs to be extended due to an overflow condition. This area is reserved to reduce the possibility of the system being unable to continue processing due to lack of real storage to accept pages needed for system functions. The area may, or may not be used by the system, but it is reserved and further reduces the area available for V=R requests. The reservation remains the same, regardless of the real storage size of the system involved. On a small real storage size system, this can be a significant factor in V=R area availability.

- VS1 is a paging system. Real storage is managed by the control program in a manner that will prove most efficient for currently executing programs. This may prove to be the least efficient for V=R operations. The system will make all effort possible to honor requests for V=R area, but current operations still take precedence.

With the number of variables that can occur in real storage usage during system processing, it is impossible to predict V=R storage availability at any given time. Each installation will have to depend upon experience to determine the most feasible allocations for SQA and PQA based on job mix and job flow for the installation. Experience should also determine when it is most feasible to run jobs with V=R requirements. In theory, they should be run as soon as possible after IPL to assure the maximum provision of available V=R area.

System Output Writers

System output writers, resident in partitions as they are in OS MFT, do not exist in VS1. Instead, system output, with the exception of direct system output, is written by JES output writer(s), resident in the pageable supervisor area of storage. For additional information concerning the JES writer, see the explanation under the heading *Output Writers* in this section.

Direct System Output Writer Partitions: Direct system output writers operate in partitions. To control the writing of a job's output, the direct system output writer must be operating in the same partition as the job; also, the job's class must be an eligible job class. An eligible job class is one that has been assigned to the direct system output writer when the writer was started. Direct system output writers can handle up to 15 different job classes.

Checkpoint/Restart

The checkpoint/restart facility permits a restart either at the beginning of a job step (step restart) or at a checkpoint within a job step (checkpoint restart). A checkpoint is requested by issuing a CHKPT macro instruction; a step restart is requested by including special parameters in the job control statements for the job.

In a checkpoint restart, the restart must be executed in the same virtual storage area as was used for the original execution. The required virtual storage must be contained within one partition. Furthermore, the partition must be a problem program partition. Restart may also be delayed if a DEFINE command entered by the operator changes the boundaries of the partition. In a step restart, there are no such restrictions.

A CHKPT macro should not be issued by a subtask or by a job step task that has active subtasks.

Partition Redefinition

Partition redefinition allows the operator to change the number of partitions, their size, and their job classes at any time after initial program loading (IPL). Adjacent partitions may be combined to accommodate jobs with large storage requirements; these partitions may be reestablished subsequently (within system generation limits) when the need for a large partition has passed. Job classes assigned to a partition may be changed also, to accommodate changes in the work load for one or more job classes.

In addition, if the time-slicing feature is included in the system, the number of time-slicing partitions can be decreased, or increased within system generation limits, the range of the highest and lowest partition number to be time-sliced can be changed, or the amount of time to be allotted to each task can be modified. All time-slicing attributes may also be canceled.

Partition redefinition is invoked in any of three ways, depending on whether it is invoked during or after system initialization. At system initialization, the partition configuration may be changed by replying 'YES' to message IEE801D 'CHANGE PARTITIONS?'. Alternatively, partition redefinition may be invoked after system initialization by entering the operator command, DEFINE, either with or without the keyword PARM=membername. Without the keyword, the operator redefines the partitions manually. With the keyword, the system redefines the partitions automatically by referencing a member in SYS1.PARMLIB. For further information, see the *VS1 Features and Options* section.

Note: The DEFINE command cannot be entered through the input stream.

Partition Combination

Adjacent partitions may be combined as soon as their jobs have been terminated. If an unending job is being executed in a partition that is to be combined with an adjacent partition, the unending job must first be terminated with a CANCEL or STOP command. All other partitions that are to be combined are made quiescent by the system as soon as their current tasks are completed. Any number of *adjacent* partitions may be combined. For example, in Figure 11, P2 and P3 may be combined into one larger partition of 128K. However, P2 and P4, P1 and P3, may not be combined. When P2 and P3 are combined, the new configuration is as shown in Figure 15. P2 *or* P3 may be made the inactive partition. When P2 and P3 are combined, the job classes to be serviced by the new partition (P2) must be determined. If no change in job class(es) is specified, the classes currently being serviced by P2 remain as the job class assignments of the new partition. (See *Identity Change.*) The inactive partition (P3) is made nondispatchable until it is recovered.

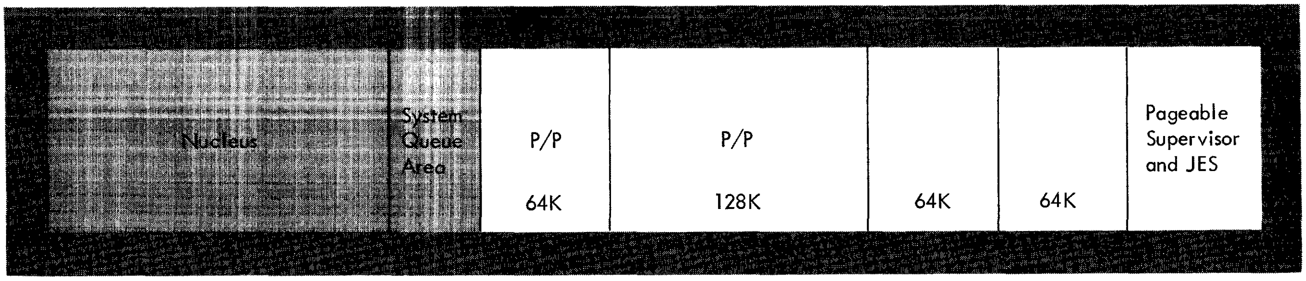


Figure 15. Partition Configuration after Combination

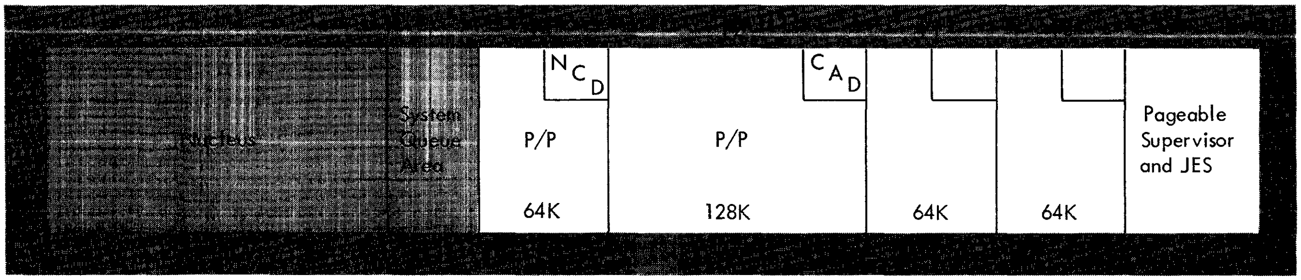


Figure 16. Partition Identification after Combination

With the storage protection and fetch protection features, a unique protection key is available for each problem program partition. A list is kept of each available key for subsequent reassignment to combined or recovered partitions. When partitions are combined or recovered, the first available protection key on the list is assigned to them.

Note: Storage assignment increases through partition redefinition should be made in increments of 64K bytes. If they are not, the system rounds the value to the next 64K increment.

Identity Change

Partition redefinition also allows the job classes specified at system generation or at system initialization to be changed. When partitions are combined or recovered, the job class(es) that will be assigned to the resulting partitions must be determined. In Figure 15, P2 and P3 were combined into the larger partition P2. However, the original partitions each had three job classes. Therefore, the decision must be made whether to choose new job classes or some combination of the six old classes. For example, P2 could be assigned job classes C, A, and M, or a new job class could be specified, such as O. A new configuration is illustrated in Figure 16. If a new job class identifer(s) is not included during partition combination, the job class(es) originally assigned to the partition which remains active is unchanged. As a result, some jobs already enqueued on the input queue may not have a partition assigned to service them.

Partition Recovery

Partitions that were combined may be reestablished, or recovered. In Figure 15, P3 is now inactive, but is to be recovered. Once again the decision must be made as to which job classes will be assigned to both P2 and P3. P2 and P3 need not retain their original size, nor their previous job classes. With P3 recovered, Figure 17 shows a possible new configuration.

Note: When recovering partitions, a job class(es) must be specified for the partitions being recovered, since only the currently active partition has a job class(es) assigned.

Partition Definition Processing

As shown in Figure 18, when the operator enters either DEFINE or the reply 'YES' to the 'CHANGE PARTITIONS?' message, the system requests that the definitions be entered. If 'LIST' was specified, the system lists the current partition configuration, including the time-slicing specifications, if this feature was chosen. (The operator must remember to CANCEL all affected unending jobs before redefining the system. If he does not, the new partition definitions do not take effect.) After definitions are entered, the system checks their validity and also inhibits scheduling subsequent jobs into the affected partitions. The system will stop any active direct system output writer in the affected partition. When the current jobs have been terminated, the new definitions are implemented. The *Considerations* section contains operating considerations associated with partition redefinition.

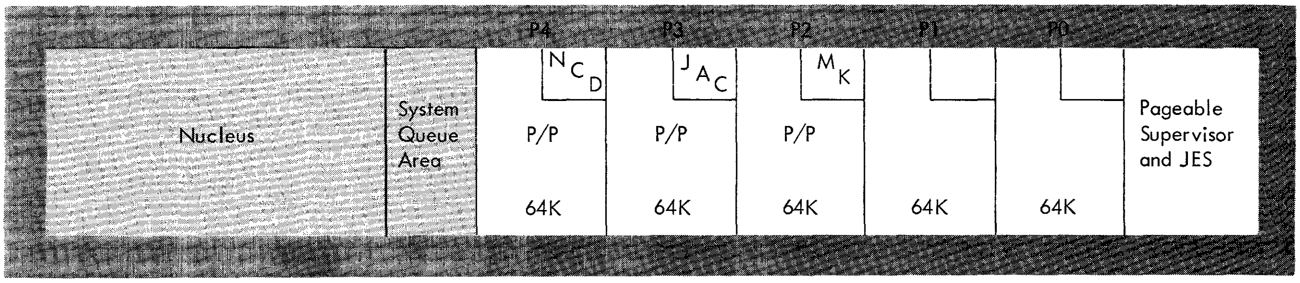


Figure 17. Partition Configuration after Recovery

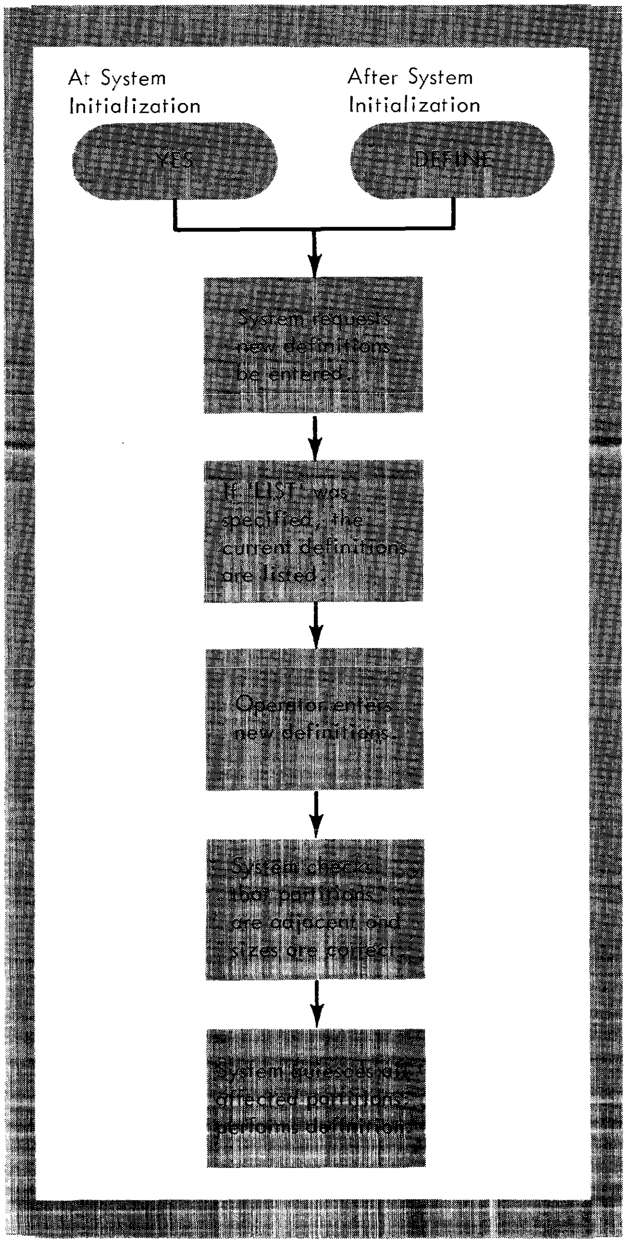

At System Initialization
After System Initialization
YES
DEFINE
System requests new definitions be entered.
If 'LIST' was specified, the current definitions are listed.
Operator enters new definitions.
System checks that partitions are adjacent and sizes are correct.
System quiesces all affected partitions; performs definition.

Figure 18. Partition Definition Processing

Input Readers

VS1 allows the user to specify as many input readers as he desires, within the limits of the virtual storage allocation he has made for his JES area at sysgen time. The readers are part of JES, are resident in the pageable supervisor area of storage, and operate concurrently with writers and with problem programs. The maximum number of readers required is specified at sysgen time in the RDR parameter of the JES macro instruction or at IPL time by the JES reconfiguration facility. One reader is assumed if the parameter is omitted.

All input to VS1, except console-entered commands, passes through the JES reader(s). The JES reader is started for each input stream by entering a START command, and it continues to service an input device until terminated by a STOP command or until end-of-file is encountered on a device other than a card reader.

When the JES reader is initialized, the Job Entry Peripheral Services (JEPS) monitor, a part of JES, initializes work area storage for input stream dependent information and attaches the reader as a subtask for each input stream started. The reader examines the records and separates them into commands, job control, and data.

The functions of the JES reader are to:

1. Scan each job statement for valid jobname, class, type of run, and priority keywords.
2. Obtain a time stamp containing the start time as each job is encountered in the input stream.
3. Give an internal VS1 identification consisting of a number concatenated with the job name.
4. Send to the command processing section of the JES reader all commands not within a job. The command is processed according to a specified disposition that may include:
   - Execute the command.
   - Write the command on the console and execute.
   - Write the command on the console and ask the operator whether or not to execute.
   - Ignore the command.
5. Put all the JCL for the job into a JCL spool data set.

6. Take a job-end time stamp when the job has been completely read. The real time in the JES reader is calculated for the job. The total number of statements read for the job is available at this time. If no error has been encountered, the job is enqueued by class and priority on the job queue. If an error has been encountered, the job is deleted and, if an unrecoverable I/O error occurs, the reader is closed.
7. Bypass all input from the input stream, if requested, until a job with a specified name is encountered.
8. Complete processing for the current input stream when an end-of-file is encountered on all devices except a card reader. The JEPS monitor is called to indicate that the reader has been stopped.

Figure 19 is a graphic representation of JES input reader processing.

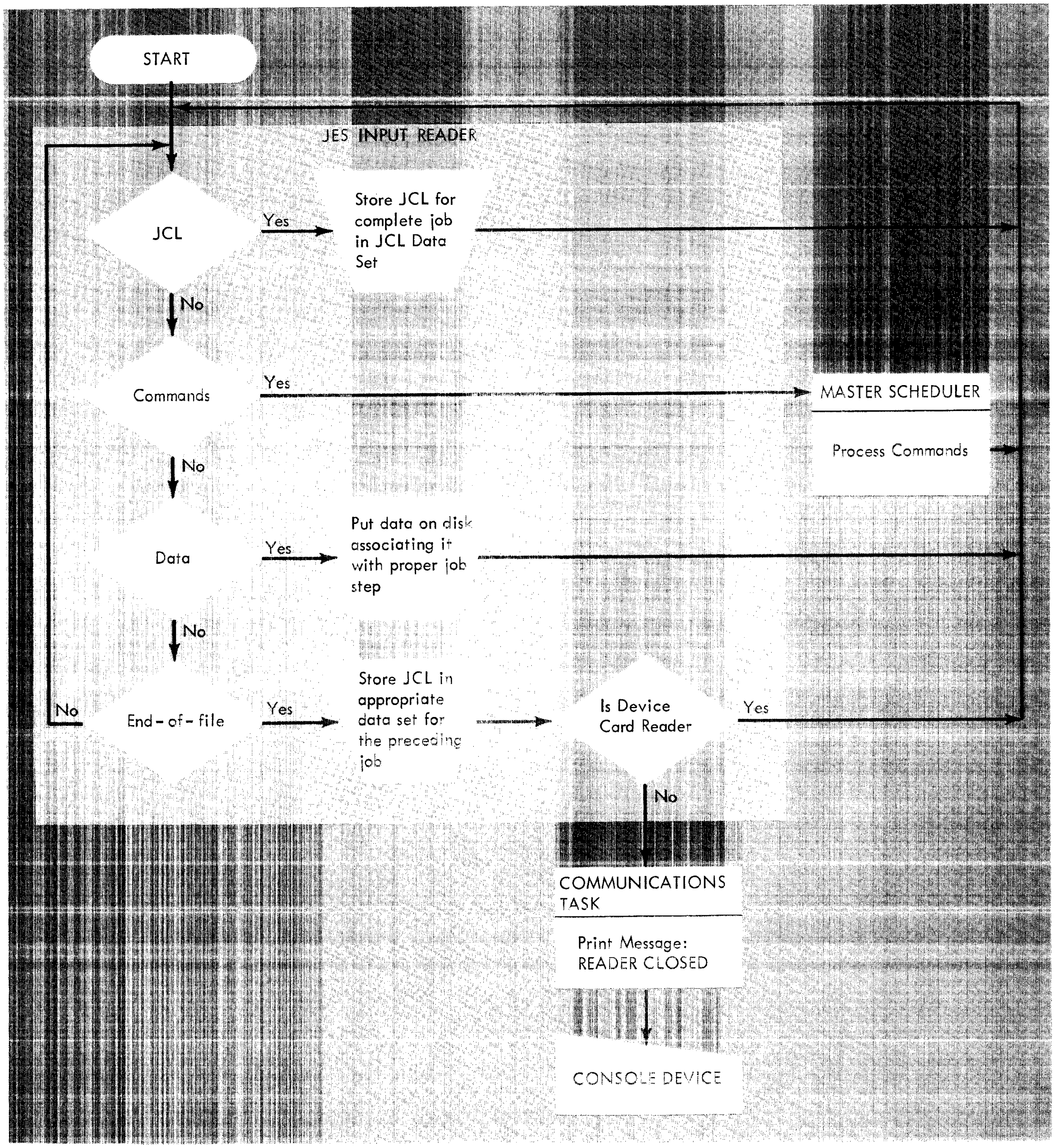

START
JES INPUT READER
JCL
Yes
Store JCL for complete job in JCL Data Set
No
Commands
Yes
MASTER SCHEDULER
Process Commands
No
Data
Yes
Put data on disk associating it with proper job step
No
No
End-of-file
Yes
Store JCL in appropriate data set for the preceding job
Is Device Card Reader
Yes
No
COMMUNICATIONS TASK
Print Message: READER CLOSED
CONSOLE DEVICE

Figure 19. JES Input Reader Processing

Enqueuing Jobs by CLASS and PRTY

A job definition, containing a disk entry record and accounting records, is created for each job read by the input readers and is placed in the queue specified by the CLASS parameter. Jobs are entered into the input work queues for each class according to the PRTY parameter (PRTY values range from a low of 0 to a high of 13). One input work queue exists for each of the 15 job classes. Jobs having the same class and priority are placed in the queue first-in/first-out (FIFO). When the input work queue for a job class contains one or more jobs, termination of a job in any partition assigned to service work for that job class is followed by selection of the next highest-priority job from the input queue. Selection and initiation of the new job does not require operator intervention.

For example, if CLASS=D, PRTY=12 is specified on the JOB statement, the job is placed on the input work queue for job class D, behind any previously enqueued PRTY=12 jobs, but ahead of all jobs of lower priority.

Note: If no PRTY parameter is specified on the JOB card, the job is assigned the default priority specified in the reader procedure.

The PRTY parameter applies only to *initiation* priority, not to dispatching priority. Dispatching priority determines which job should be given control of the CPU. In VS1, dispatching priority is derived from the relative position of the partitions: P0 has highest priority, P51 lowest.

The dispatching priority of a task is determined by the relative position of the task control block (TCB) on the dispatching queue. (The dispatching queue is the chain of TCBS indicated by the TCBTCB fields.)

If subtasking is not used, all TCBS are established in the nucleus at system generation. These are ordered to provide a dispatching priority starting with resident system task TCBS, through the job step task TCB of the highest priority partition (P0), to the successively lower priority partitions' TCBS (P1-P51). Control of the CPU is given to the program represented by the highest-priority ready TCB.

If subtasking is used, TCBS established at system generation in the nucleus represent the resident system tasks and the job step task of each partition. However, each job step task can attach subtasks, each of which will have a TCB located in the partition's protected queue area. The dispatching priority is initially the same as the partition's priority. The dispatching priority differs from the partition's priority when a job step task issues a CHAP (change priority) macro instruction to change its dispatching priority. If dispatching priorities are not changed, each partition's job step task is dispatched before its subtasks, which are then dispatched in the order in which they were attached. When all of a job step's subtasks have been dispatched, the job step task of the next lower partition can be dispatched.

If the time-slicing feature was specified at system generation, the effective dispatching priority of a group of time-sliced partitions can be altered. Time-slicing allows the user to establish one group of consecutive partitions in which the task in each of the partitions is assigned a uniform interval of time to retain control of the CPU. When the allotted time has elapsed, the next lower-priority ready task gains control of the CPU for its allotted time. This process continues until either all tasks are waiting and completed, or until a task of higher-priority becomes ready.

Job Initiation and Termination

The job scheduler contains the initiation, interpretation, allocation, and termination portions of the control program. As illustrated in Figure 20, the job initiation portion selects jobs from input work queues and determines which type of output writer to use for each output class. As each problem program is executed, it retrieves its input (SYSIN) data from the direct access device where it was previously stored by the input reader. (*Note* that this retrieval takes place at direct access speeds, which is faster than reading input data directly from a card reader.) During problem program execution, output data directed to an output class is recorded on a direct access device, or written directly to the output device, depending on the type of output writer selected.

Jobs are scheduled for execution according to:

1. Their job class identifier.
2. Their priority within the job class queue.
3. An available partition corresponding to the appropriate job class.

When a job is complete, the scheduler performs the required termination and informs a system output writer that the data produced by the problem program is ready to be written on the specified device.

Job Initiation

An initiator may be started in each problem program partition that is defined at sysgen time. When each initiator is started, job management allocates an area on DASD called SWADS (scheduler work area data set). Each SWADS holds the tables for the problem program currently handled by that particular initiator. Thus, it is a sequential data set that is reusable; that is, the tables for a new job overlay those from the previous job. The SWADS is deallocated when the STOP command for that initiator is processed. (For a description and discussion concerning SYS1.SYSJOBQE, see section JQF in

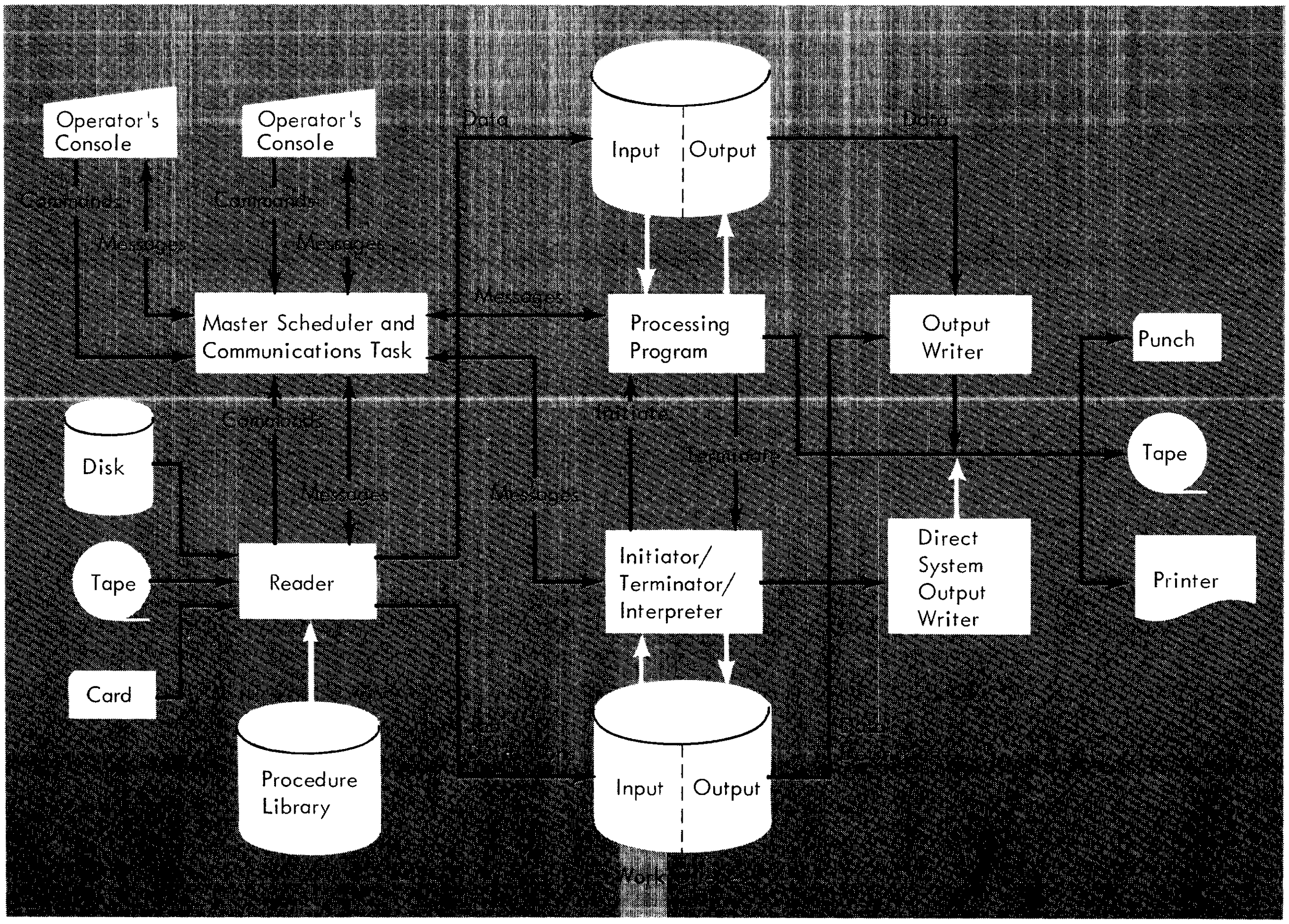


Figure 20. The VS1 System

the *Use* portion of this manual.) The initiator selects a job from the input work queues established by the system input reader and schedules it for execution. Initiators obtain jobs for partitions based on the job classes assigned to the partitions, and the priority of the jobs within their job classes. The initiator interfaces to allocation procedures for device allocation. The job is then scheduled for execution. An initiator is given control after a job has been terminated.

When the job terminates, the initiator then schedules the next available job into its partition and passes control to the first step of that job. When system output writers are used, output data sets are placed on direct access storage devices while the job is being processed. The output data sets are enqueued by output class and subsequently retrieved by system output writers.

Job Termination

The termination portion of the control program determines first whether step termination or job termination is to be performed. Step termination includes disposing of data sets, deallocating input/output devices, processing condition codes, and executing an accounting routine. If the job contains additional steps, control is returned to the initiator to schedule the next job step. Job termination is performed after the last step of a job has been terminated. An accounting routine is executed; data set disposition and input/output deallocation that could not be done at step termination are completed.

If the job used direct system output writers, its output was written directly to an output device or devices: no intermediate device had to be used and no output work is enqueued.

If the job used system output writers, its output is entered in the output work queue for processing by the system output writer. Output work queue members are enqueued within output class according to the PRTY parameter on the JOB card. Jobs having the same output class and priority are placed in the queue FIFO. For example, if a single output class is specified for system messages and all output for a particular problem program, the output work queue for that class includes, at job termination:

1. All system messages produced at job initiation, such as allocation messages.
2. All system messages produced during job termination.
3. All problem program output.

This output is transferred to the specified output device in the order shown. Different types of output, such as system messages and problem program data, are never intermixed.

Data sets for a job are enqueued on the output work queue according to output class and priority, so that they can be written by an output writer. These data sets may include data sets produced during a job step, as well as control program messages. Depending on its characteristics and the way it is to be processed by the control program, a data set may be assigned to any one of 36 output classes (A-Z, 0-9) defined at an installation. A particular output class may reflect such characteristics as priority of the data, type of device to record it, or location or department to which it is to be sent. (See *Choosing Output Classes* in the *Considerations* section.)

Direct System Output Writers

Direct system output writers write problem program and system messages, produced by the initiator/terminator, directly to system output devices. Valid output devices are: printer, punch, and magnetic tape.

Direct system output writers are started in problem program partitions and are initialized before the problem program gets control of the partition. When the problem program writes its output, the output will go directly to the output device assigned to the direct system output writer. System output writers can handle as many as 15 different job classes. If no job class(es) is specified in the START command, job classes to be processed are obtained from:

First, the partition information block; then, the direct system output procedure.

If job classes are specified in the START command, they override those specified in the other two sources. For example: if the operator enters the command,

```
START DSO.P3,283,,(JOB CLASS=ABC,OUTCLASS=B)
```

any job running in partition 3 with a jobclass of A, B, or C and an output class of B will have its output written directly to tape device 283.

The job class and output class of a direct system output writer can be changed by use of the MODIFY command. For example: if the operator enters the command,

```
MODIFY 283,JOBCLASS=DE,OUTCLASS=A
```

any direct system output writer writing to output device 283 will process jobs with a jobclass of D or E, and an output class of A.

Direct system output writers can be stopped by a STOP command. The STOP command may specify a partition ID, which will cause all DSO in the partition to be stopped, or it may specify a device, which will cause only DSO to the particular device to be stopped.

A user-supplied DSO procedure may be used, but it must execute the IBM-supplied direct system output writer.

Output Writers

Output writers write system output data sets created by problem programs, and system messages produced by the initiator/terminator tasks. All system output data sets are written on direct access storage devices. These are used as the input data sets for the output writers. Valid output devices for an output writer are printer, punch, and magnetic tape.

When a job is terminated, system messages and output data are enqueued in the appropriate system output queue. One system output queue exists for each output class. Queues are serviced in the order specified in the START command for the writer. As many as eight output classes may be specified. These classes override those in the writer cataloged procedure. The writer dequeues the first entry from the primary queue. If there are no entries in the primary queue, the writer dequeues the first entry in the secondary queue. This continues through the eighth queue, or until the writer finds work.

VS1 allows the user to specify as many system output writers as he desires, within the limits of the virtual storage allocation he has made for his JES area at sysgen time. The writers are part of JES, are resident in the pageable supervisor area of storage, and operate concurrently with readers and with problem programs. The maximum number of writers required is specified at sysgen in the WTR parameter of the JES macro instruction or at IPL time by the JES reconfiguration facility. One writer is assumed if the parameter is omitted.

All VS1 output data, with the exception of direct system output data, must pass through the JES writer. The data includes output data sets and data sets containing system messages and job control language. The writer is invoked for each output stream by entering a START command. It continues to service an output device until terminated by a STOP command.

When the JES writer is initialized, the JEPS monitor initializes work area storage for output stream dependent information and attaches the writer as a subtask for each output stream started.

The JES writer uses job entry central services to obtain output data and system messages in the order they appear in the specific output class queue. The writer processes one copy of the job's JCL, system messages, and output data set, unless otherwise specified. Multiple copies can be provided by so specifying in the job's JCL, or by the WRITER command.

When all output for a job is processed, an accounting linkage routine is called for final job accounting and the job is completely removed from the system. At this time, the space held by the job on the spool and queue devices is freed.

A user-written output writer procedure may also be used. This procedure may execute a user-written writer program or the IBM-supplied writer. If a user-written writer procedure is used, it must be placed in SYS1.PROCLIB and named in the START command.

System Restart

Because it is sometimes necessary to shut down the system (end-of-shift, end-of-day, normal maintenance, or system malfunction), system restart allows the system to resume operation without having to reenter jobs that have been enqueued. Information concerning jobs on the input, hold, and output queues, and jobs in interpretation, initiation, execution, or termination, is preserved for use when the system is reloaded (see Figure 21). When the system is restarted, the operator receives messages describing the status of each job in the system. If the job was not enqueued, the jobname is written out at the console. If the job was under control of a scheduler, the jobname *and* stepname are given. In addition, the operator is informed of whether allocation was being performed for the job, or whether the job was being executed or terminated.

Invoking System Restart

After the system is reloaded, and after nucleus initialization, system restart may be invoked by *omitting* the "F" suffix from the Q=(unitname,[F]) parameter of the SET command, or by omitting the Q=parameter

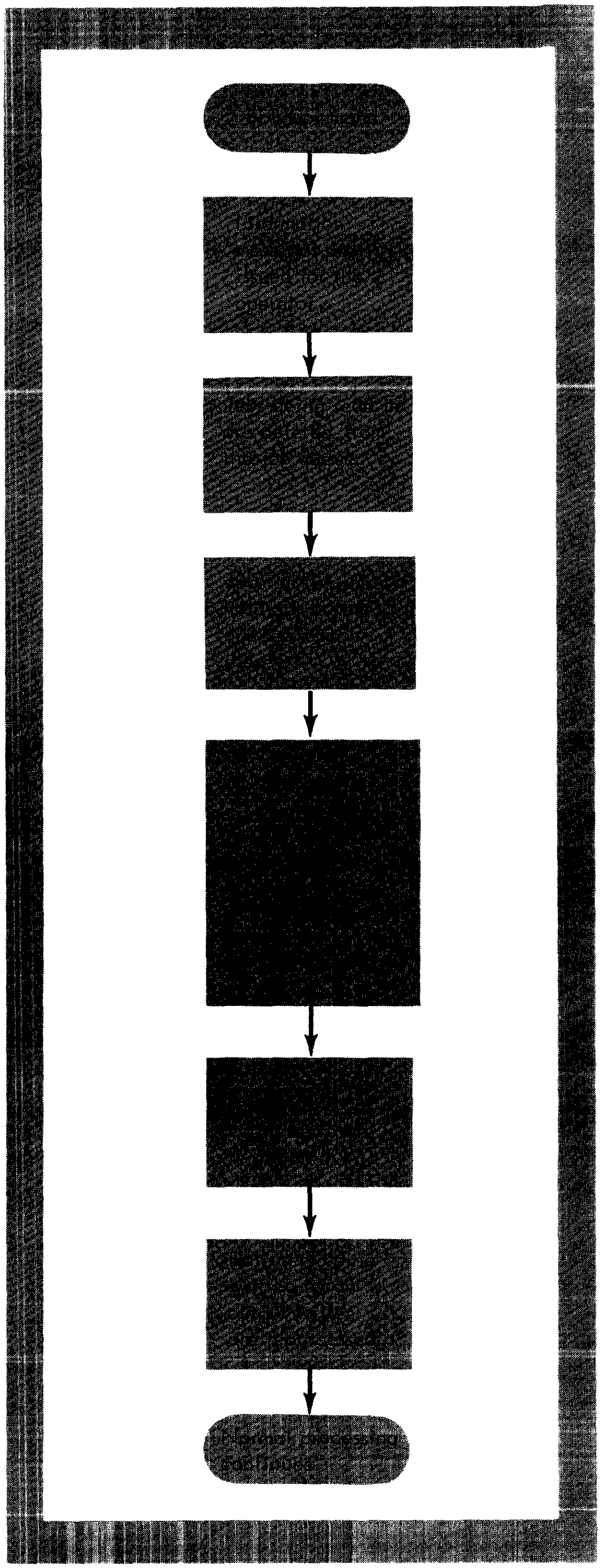


Figure 21. System Restart Processing

entirely. This is valid only for the *initial* SET command, and cannot be done at any other time. This omission indicates to the system that the job queue data set (SYS1.SYSJOBQE) already exists in proper format, and requires only initialization.

Jobs that Were Being Read In

When the system must be restarted, jobs that were being read in are deleted from the job queue and must be resubmitted.

Jobs on Input, Hold, and Output Queues

All jobs that were enqueued on their appropriate job class, hold, or output class queues, *remain* there for subsequent processing when the system is restarted. No operator action is required.

Jobs that Were Dequeued

Jobs that were dequeued from an input or hold queue may or may not have been completely interpreted. If interpretation was not complete, system restart reenqueues the job where it will be subsequently dequeued by an initiator and reinterpreted. If interpretation was complete, the job will be scheduled for restart if possible. Otherwise, the job will be canceled, and output data sets created for the job will be enqueued.

With the restart facility, any job that was in step termination at restart time is executed starting at the next step of the job, after system restart is complete. Any job that requested restart is restarted at the current step, after system restart is complete, if the following conditions are satisfied:

- The step completed the allocation phase (that is, the phase between allocation and step termination).
- System completion code 2F3, designating system restart, was specified as eligible for restart.
- The operation replied YES to the verification request (message IEF225D) to restart the job.

System Output Processing

Output jobs that were being processed by a system output writer are requeued to reprocess any data sets that had not been written completely at the time the system was shut down. No operator action is required. All system messages and data sets that had not been processed are written by the first eligible output writer started. System log data sets are queued to output class A by system restart routines.

Standard and Optional Features

The standard features included in VS1 are:

1. Same as OS/MFT, Release 20.1.
2. Linkage Editor/Loader.
3. System Assembler.
4. Storage Protection.
5. Jobstep Interval Timer.
6. Identify.
7. Job Entry Subsystem.
8. Recovery Management:
   a. MCH (Machine Check Handler).
   b. CCH (Channel Check Handler).
9. Page Supervisor.
10. Access Methods:
    a. BSAM.
    b. QSAM.
    c. BDAM.
    d. BPAM.
11. Utilities:
    a. System Utilities

       IEHATLAS
       IEHDASDR
       IEHINITT
       IEHIOSUP
       IEHLIST
       IEHMOVE
       IEHPROGM
       IFHSTATR

    b. Data Set Utilities

       IEBCOMPR
       IEBCOPY
       IEBDG
       IEBEDIT
       IEBGENER
       IEBISAM
       IEBUPDTE
       IEBPTPCH
       IEBTCRIN

    c. Independent Utilities

       IBCDASDI
       IBCDMPRS
       ICAPRTBL

12. Service Aids:
    HMAPTFLE
    HMASPZAP
    HMBLIST

HMDPRDMP
HMDSADMP
IFCDIP00
IFCEREP0
IMCJOBQD
Generalized Trace Facility (GTF)

13. Multitasking.
14. On-Line Test Executive Program (OLTEP).
15. Remote Entry Services (RES).
16. Automated System Initialization.
17. Partition Deactivation/Reactivation.
18. Missing Interruption Checker.
19. Automatic Partition Redefinition.
20. User Modify Logical Cylinder Facility.
21. Greenwich Mean Time (GMT).

The optional features available for VS1 are:

1. Selected resident SVC routines.
2. Selected resident modules from LINKLIB.
3. PCI FETCH.
4. Operator communication during initialization.
5. System Log.
6. Dynamic Device Reconfiguration (DDR).
7. Checkpoint/Restart.
8. Multiple Console Support (MCS).
9. System Management Facility (SMF).
10. Shared Direct Access Storage Device (DASD).
11. Device Independent Display Operators Console Support (DIDOCS).
12. Time Slicing.
13. Access Methods:

    BTAM
    TCAM
    GSP/GAM
    BISAM
    QISAM
    VSAM
    RTAM

14. Alternate Path Retry.
15. Automatic Volume Recognition (AVR).
16. Integrated Emulators.
17. Conversational Remote Job Entry (CRJE).
18. I/O Load Balancing.
19. DEB Validity Checking.
20. Dynamic Dispatching.
21. Fetch Protection.
22. Dynamic Support System (DSS) (for planning purposes only).
23. Direct Access Volume Serial Number Verification.

Considerations

In preparing for the use of VS1, data processing planners and system programmers should evaluate not only the characteristics and requirements of the jobs to be processed by the system, but the characteristics and facilities of the system that influence how a job is processed once it has been presented to the system. Some of these characteristics are general and apply equally to all types of jobs. Others are related directly to job type. A third category of characteristics, although related to job types, is exhibited primarily in system operation, and must be considered by machine room supervisors and machine operators.

In this section, the topic *General Considerations* describes items of interest primarily to planning personnel, that should be considered before generation of a VS1 system. The next four topics—*Batch Processing, Telecommunications, Graphics,* and *Concurrent Peripheral Operation*—describe considerations important to the systems programmer and the application programmer. These four topics are organized similarly, in "checklist" fashion, so that the reader interested in a given job type need read only *General Considerations* and the topic corresponding to the job type in which he is interested, to learn all considerations pertinent to that job type. Because partition configurations depend on the amount of real and virtual storage available as well as on the types of jobs to be run, the topic *Typical System Configurations* describes partition arrangements for systems with 128K bytes, 192K bytes, and 384K bytes of real storage. These configurations are general, but should be helpful for planning. The seventh topic, *Operating Considerations* briefly describes characteristics of VS1 that may affect operating procedures.

General Considerations

Several considerations apply to all phases of the system; these must be considered regardless of the type of job that is being run. They include:

- Estimating storage requirements.
- Using resident reenterable routines.
- Placing system data sets on direct access devices.
- Sharing direct access devices with other systems.
- Choosing the number and size of partitions.
- Specifying appropriate job classes.
- Assigning job names.
- Formatting problem program messages.
- Choosing output writers.
- Avoiding system interlocks.

Estimating Storage Requirements

Refer to the publication *OS/VS1 Storage Estimates,* GC24-5094, for information on estimating the storage requirements for your system.

Single Console vs. Multiple Consoles

Through multiple console support (MCS), an installation may use one primary (or master) console and multiple secondary consoles where each console is dedicated to one or more system functions (for example, tape library, disk library, or teleprocessing control). MCS services all consoles concurrently, creating an environment for operator/system interaction that gives each console the appearance of being the only console on the operating system. Each console operator receives only those messages from the system that are related to the commands that he enters and to his assigned functions.

MCS provides the mechanism to:

- Route messages to selected functional areas.
- Allow a user-written exit routine to modify the message's routing and descriptor codes prior to the issuance of the message.
- Switch to an alternate console if a primary console should fail.
- Allow automatic message deletion on devices such as display tubes (graphics).
- Support a hard copy log for the recording of routed messages, operator commands, and system responses.

Selective message routing, provided under MCS, is the ability to route both problem program and system-initiated messages to functional areas and the SYSLOG device. Messages appear only on consoles that have been specifically designated to receive the messages. In this manner, a console whose function is to receive tape messages, for example, is prevented from receiving messages not pertinent to that function.

A system generation option is provided to permit insertion of a resident, user-written exit routine in the communications task. The exit routine receives control prior to routing any WTO and WTOR messages whose routing codes will be used by the operating system.

The exit routine may examine but not modify the message text; however, the exit routine may modify the message's routing and descriptor codes. Messages will only be sent to those locations specified in the modified routing codes.

MCS permits console switching, which can be initiated automatically, by operator command, or by the operator manually pressing the interrupt key on the system control panel:

- Automatic console switching to an alternate console occurs when permanent hardware errors are detected by the operating system.
- Command-initiated console switching occurs when the system accepts a valid VARY operator command. Command-initiated console switching is used to restructure the system console configuration and the hard copy log. Console switching to an alternate console can be performed by placing the original console either online or offline.
- Manual switching is limited to the master console (primary console device) and is initiated by pressing the interrupt key on the system control panel. Manual switching to a new master console is used when the master console is inoperative and the hardware failure cannot be detected by the system.

Messages can be automatically deleted from the screen on the operator console with CRT display by using the DOM macro instruction. When a system or problem program no longer requires that a message be displayed—for example, if a WTO macro instruction was issued and the message is no longer needed—a DOM macro instruction should be issued to delete the message from the screen.

MCS allows buffered or immediate hard copy. Thus, no information is lost when messages and operator commands are deleted from graphic displays. In addition, the hard copy device can be used as a collection point for all messages and commands. Routing codes and the time, if the timer option is present, are prefixed to all messages and commands that are sent to the hard copy log. The system log—the only buffered hard copy device supported—must be specified at system generation, and can be modified at system initialization or during system operation. If hard copy is desired on a console, the hard copy device can be specified at system generation, at system initialization, or during system operation. Although the system log is supported with or without the MCS option, the hard copy log is only supported with the MCS option.

Note: If the multiple console support (MCS) option is selected when coding the SCHEDULR macro instruction, a SECONSLE macro instruction must be included for each console except the master console; that is, for the alternate to the master console and for each additional secondary console.

PARTITNS Macro Instruction

The PARTITNS macro instruction must be used. It establishes the maximum number of partitions for an installation. The maximum number of partitions that an installation *intends* to use should be established at system generation, because this number can be reduced during system initialization or during operation. Unnecessary system generations may be avoided because the number of partitions in use may *never* exceed the number established at system generation. Therefore, if the maximum number of partitions that might be used at some later date is generated, the system does not have to be regenerated to increase the number of partitions in the system. The attributes of each partition are established as follows:

Pn(C-class,S-size)

where the characters P, C-, and S- must appear as shown.

n specifies the partition number (0 to 51).
class specifies the function of the partition as follows:

S = system task partition
xxx = problem program partition specified as alphabetic characters from A through O that indicate the job classes that can use the partition.

size specifies the size of the partition from 0 to the maximum in 64K increments *without the K.*

Notes:

- Partitions may be specified in any order as long as every partition in the sequence is included.
- Partition sizes will be rounded up to 64K boundaries.
- No more than 37 system task partitions may be specified.
- At least 1, but no more than 15, problem program (user) partitions may be specified.

Page Boundary Loading

Aligning a control section or a named common area on a page boundary can be used to effect a lower paging rate, thus making more efficient use of real storage. To accomplish this alignment, the CSECT or common area is named on either the PAGE statement or the ORDER statement with the P operand. Either statement causes the linkage editor to locate the CSECT (or common area) on a page boundary within the load module. (When a note list is present, it precedes and is contiguous to the load module.) See *OS/VS Linkage Editor and Loader*, GC26-3813, for using page boundary alignment.

Using Resident Reenterable Routines

The resident reenterable load module feature allows problem programs to share reenterable code. It provides improved performance by placing frequently used modules in the resident reenterable routine area.

Resident reenterable routines may be made non-pageable or pageable at the user's option.

The feature uses the RAM parameter to make possible the pre-loading of both access method modules from the SVC library and any user-written reenterable modules from the link library. The feature uses four module lists created at system generation time. At system initialization, the user can either cancel the feature entirely or provide up to four different lists of access method and other reenterable modules of his own choosing to be loaded into the resident reenterable routine area.

In an operating environment where any problem program issues an ATTACH, LINK, LOAD, or XCTL macro instruction to request use of a reenterable module that is not resident in its partition, the supervisor will search the resident reenterable routine area for the module. No additional copies need be brought into virtual storage. Therefore, frequently used reenterable load modules should be loaded into the area. Any user-written reenterable load module and the loader modules from the link library can be loaded into the area by including it in one of the lists of modules specified; type 3 and 4 SVC routines, however, must be loaded only into the RSVC area.

Placing System Data Sets on Direct Access Devices

Several factors must be considered when putting system data sets (SVCLIB, MACLIB, SYSPOOL, LINKLIB, PROCLIB, PARMLIB, SWADS, PAGE, and SYSJOBQE) on direct access storage devices. If all nine data sets are on the same device, throughput is decreased because of excessive arm interference. To increase throughput, data sets should be balanced on devices; devices should be balanced on channels. The ideal condition would be to have each data set on a different direct access device, and each device on a separate channel. In installations with smaller systems, it would be best to have SYSJOBQE, SYSPOOL, and LINKLIB on the same direct access device on channel 1, and SVCLIB, PROCLIB, PARMLIB, SWADS, PAGE, and MACLIB on another device on channel 2.

Whenever more than one library is to be placed on a 2314, 2319, 3330, or 3333 disk storage device, arm movement can be substantially reduced by placing the volume table of contents (VTOC) approximately midway between the first and last cylinders being used. The data sets, starting with the most frequently referenced, can then be alternately placed on both sides of the VTOC with the least referenced data sets farthest from the VTOC.

Blocking the Procedure Library

Blocking the procedure library conserves DASD storage space. The procedure library may be blocked during system generation or afterwards by using utilities.

Blocking the procedure library during system generation is done by pre-allocating the procedure library with RECFM=FB and BLKSIZE=(a multiple of 80). During stage II of system generation, the IEHMOVE utility program blocks the procedure library on the new system.

A preliminary study used a typical procedure library containing 54 procedures to determine the most efficient blocking factor to conserve DASD storage space. Blocking factors from 1 to 40 (BLKSIZE=80 to 3200) were used. It was found that blocking factors in the range 8 to 12 were most efficient. For example, on a specific DASD, an unblocked procedure library required 30 tracks but a blocked procedure library (with blocking factors in the range 8 to 12) required only 21 tracks, a reduction of 30% in DASD storage space.

Sharing Direct Access Storage Devices (DASD) with Other Systems

If the shared DASD feature is selected at system generation, considerations should be given to volume assignment and classification (read only or read/write) of common data sets. Volume handling and device reservation procedures must be carefully defined, because the shared DASD feature cannot prevent or resolve interlocks between systems.

Choosing Number and Size of Partitions

The number of partitions needed at an installation depends primarily on the number of different job *categories* (that is, batch, graphics, telecommunications, and CPO) expected to run concurrently. At least one partition must be specified for each category. The number of partitions for each category, based on the number of jobs expected to be run in each, should then be established. In practice, the maximum number of partitions for which there is available virtual storage should be established. If fewer partitions are needed during operation, the number of partitions can be reduced by the operator either at system initialization or during operation. If necessary, partitions may be reestablished up to the limit specified at system generation.

Note: If the maximum number of partitions is established at system generation, the size of the system queue area (SQA) at system generation also should reflect this maximum number. Then, if the number is reduced at system initialization, the SQA can also be reduced, by replying to message IEA101A 'SPECIFY SYSTEM PARAMETERS'.

Within the limits of the system, the maximum number of partitions that can be specified depends on the amount of real and virtual storage available. If a job is known to exceed the size of its intended partition, an adjacent partition can be eliminated and its storage reassigned to the other partition. Reassignment of contiguous partitions can be accomplished without interfering with unaffected partitions.

Choosing Appropriate Job Classes

A system installation can be set up for maximum efficiency and throughput by properly using the partition job class concept. Particularly, the processing characteristics of jobs likely to execute concurrently must be examined. Failure to consider job mix can lead to degrading system performance. Multiprogrammed jobs can, under certain circumstances, run slower than they would if processed sequentially. The required balance can be achieved simply with proper use of the CLASS parameter by:

- Establishing the job characteristics to be controlled, based on the typical processing workload.
- Establishing a suitable partition structure compatible with the job characteristics to be monitored.
- Establishing the convention that jobs having certain characteristics are to be directed, through the CLASS parameter, to the appropriate partitions.

Typical job characteristics are:

- High compute, low input/output time.
- Balanced compute and input/output time.
- Low compute time, high input/output time.
- Use of specific types of input/output equipment, such as 2250 terminals, magnetic tape only, or telecommunications terminals.
- Large real storage requirements.
- Small real storage requirements.
- Large virtual storage requirements.
- Small virtual storage requirements.
- Setup or non-setup jobs.
- Use of preallocated data sets.
- Time-slicing considerations.

With this type of categorization, job mix can be balanced for improved throughput. For example, one partition can be established for high-input/output jobs and another for high compute-time jobs. Process-limited jobs (such as compilers) can then be assigned to the high compute-time partition, and jobs with high input/output requirements (such as sort programs, and reading and writing of data sets) to the input/output partition. Normal job scheduling should then produce a satisfactory job mix. Because jobs are queued by the CLASS parameter, and because each partition is scheduled for its next job immediately after the preceding one is complete, the system as a whole tends to execute complementary jobs concurrently.

The CLASS parameter may also be used to direct jobs that are to be time-sliced to partitions that were defined as time-slicing partitions. However, when using the time-slicing feature, caution should be taken when assigning a job class to a particular job. If a job is to be time-sliced, it must be assigned a job class that will be serviced by a time-slicing partition. Likewise, if the job is not to be time-sliced, it should not be assigned a job class that a time-slicing partition has been assigned to service.

A partition should be established to service each job class specified on a JOB card. If a job is assigned to an unserviced job class, it remains on the input queue for that class indefinitely, or until the operator discovers (by use of the DISPLAY N command) that the job has not been executed.

Default Job Class

If *no* CLASS parameter is specified on the JOB card, the system assigns job class A to the job. Therefore, a partition should not be assigned to service job class A unless all jobs run at the installation will fit into that partition. It is advisable to make job class A either a secondary or tertiary job class in one or more partitions, to ensure that any jobs that are assigned the default job class will be executed.

The default job class is given to a job only when *no* CLASS parameter is specified, not when an incorrect job class is given. For example, if P2 is specified as job classes M and L (Figure 11), P3 as C, J, and A, and P4 as N, C, and D, the following JOB card illustrates an invalid job class specification:

```
//MFT JOB ,'MYJOB',MSGLEVEL=0,CLASS=G
```

Because job class G is an invalid job class for this particular configuration, the job will not be assigned job class A. It is placed on the CLASS=G queue, and is never initiated. It remains there indefinitely until the operator discovers that the job has not been executed. Therefore, extreme caution should be used when choosing a job class for the job to ensure that a partition has been specified for that job class. To prevent delays in processing jobs with "invalid" job class designators, the operator should enter DISPLAY N periodically to obtain a listing of the jobs on the hold and input work queues. Jobs with CLASS= specified outside the range A-O (for example, CLASS=P) will be abnormally terminated.

Priority Scheduling within Job Classes

Jobs *within* job classes can be initiated according to a priority specified in the PRTY parameter on the JOB card. For example, several jobs may be designated job class B. Within this group of jobs, some are to be initiated before others. Therefore, higher priorities can be assigned to these jobs with the PRTY parameter. This affects only the way the job is initiated, *not* the way it is dispatched. If no PRTY parameter is specified, jobs are assigned the default priority established in the reader procedure and are initiated FIFO for each job class. Therefore, each group of jobs for a particular job class should be analyzed to determine if some are to be initiated before others, and to assign these preferred jobs higher priorities.

Assigning Job Names

Any valid job name is acceptable to the VS1 system. Therefore it is possible that jobs with identical job names could be in the system at the same time. These duplicate-name jobs could be on the same input queues, different input queues, or executing in different partitions. The existence of these duplicate-name jobs could cause confusion when using the DISPLAY, CANCEL, HOLD, RELEASE, and RESET commands. For example, if a CANCEL command is entered for a job in the hold queue or an input queue, the system will cancel the first job encountered if duplicate job names are in the queue.

To prevent this confusion, a procedure should be established which will ensure that all jobs have unique names. This could be done, for example, by varying a portion of the jobname, that is, JOBPAY1, JOBPAY2, etc., to reflect sequence of input. Other methods of unique identification for jobs could be derived from application, programmer's name, time-of-day, date, or any combination of these which would satisfy the needs of an installation.

In addition, job names of P0, P1, . . . P51 should not be assigned because these are the partition identifiers. For example, if a job is assigned a name of P4, and a CANCEL command is entered for this job, both the job named P4 and the job that is running in partition 4 will be canceled.

Formatting Problem Program Messages

In VS1 it is necessary to relate WTO and WTOR messages to the problem program which issued them. In order to do this, a partition identifier is added to each message issued by a problem program partition. The maximum length of WTO messages is 122 characters. The maximum length of WTOR messages is 119 characters.

The VS1 form of a WTO message is:

```
PHASE A ENTERED Pn
```

Similarly, WTOR messages in VS1 include the partition identifier as follows:

```
id REPLY 8 CHARACTER NAME Pn
```

Single Reader or Multiple Readers

In determining whether to have more than one reader, the number of problem program partitions necessary for the installation should be considered. It might be advisable to specify more than one reader. For example, the user could specify one reader for cards, one reader for magnetic tape, and a third for disk.

The primary consideration is to analyze the jobs and to determine which input device will have the majority of the input in terms of CPU time. If this device is the card reader, then one reader should probably be specified to read the input stream from the card reader continuously. If, on the other hand, there is a long input stream on magnetic tape and/or direct access storage, a reader should be specified for each of these devices. If there is only one long input stream, it would be advisable to specify one reader for that particular device.

Choosing System Output Writers

If possible, records to be written by a system output writer should be blocked. This improves throughput, because less input/output time is required, and disk arm interference is reduced. However, additional virtual storage must be provided within the problem program partition, where the records are initially blocked, and within the JES area, to which the logical records are read. (The additional space required, in each case, is equal to the logical record length times the blocking factor plus the input buffer space.)

Choosing Direct System Output Writers

Since the user has a choice between direct system output writers and system output writers, the following factors should be considered when deciding which type of writer to use:

- Each direct system output writer can process only one output class per partition and needs one I/O device assigned to it. In a system with several active partitions, there will probably not be enough I/O devices to run direct system output writers in all partitions for all output classes. In this case it is possible to have jobs running with more than one output class. A direct system output writer could handle one output class and system output writers could handle the others.
- Direct system output writers cannot handle output from system tasks, jobs canceled while on the input queue, and jobs failed by the reader/interpreter. It is necessary to start a system output writer to handle these types of output.
- System output writers and direct system output writers could be used together. If the output queue is filled, direct system output writers could be

started in the active partitions. This would allow the system output writers to clear the output queue, without stopping work in the problem program partitions.

Use of Multiple Writers

The use of multiple output writers has several advantages. In general, a unique output writer can be used for each requirement in the system. For example, the following output classes might be assigned:

- An output class for all system messages.
- An output class for all high-priority printed output, or for printed output requiring special forms.
- An output class for all punched output.
- An output class for all output to magnetic tape.

By specifying the appropriate output class in the DD statement, the programmer selects the particular device on which his output is to be recorded. Because writers can share output classes, a writer can have a primary and a secondary function. For example, if output class B is assigned to a high-priority printer, and output class C to a "background" printer, the high-priority printer processes only high-priority output (SYSOUT=B).

If no high-priority data is waiting on the output work queue, the output writer performs its secondary function by taking a job from the SYSOUT=C queue. The advantage in this use of multiple writers is not only that it makes writers available for certain types of unique work, but that it also permits them to perform other work when circumstances permit.

Note: Problem programs can access SYSOUT data sets only through BSAM and QSAM. They can no longer access them using EXCP.

Avoiding System Interlocks

A problem can occur when a task controls a resource, but is waiting for another resource which is under control of a second task. If the second task is waiting, and needs the resource now under control of the first task, a system interlock condition will occur. The first task cannot give up its resource until the second task relinquishes the resource it controls, and vice versa. The two tasks are therefore in a deadlock; processing cannot continue in either partition.

Two ways to avoid this problem are:

1. Request all resources initially; do not begin an irreversible course of action until all required resources have been obtained.
2. If the program is holding a formerly obtained resource which may prevent acquisition of another resource, release the resource *before* requesting the other resource(s). If the former resource is still required, request it together with the other resource(s).

Data Set Integrity

An interlock may also occur during job initiation, if a job requests one or more data sets which are reserved for use by another job which is currently executing in the system. In order to prevent this interlock, the operator is notified of the condition through a series of console messages. The operator must then make a decision, based on his knowledge of the jobs in the system, the system configuration, and the data sets in use. He may wait for the requested data set(s) to become available, he may place the job on the hold queue, or he may cancel the job being initiated.

Batch Processing

If the installation's work is primarily batch jobs, several factors must be considered when the system is initialized, and when it is operating. First, the number, size, and job class(es) for each partition must be determined. Then the decision must be made as to which partitions, if any, should contain direct system output writers. The proper output classes for the installation must also be determined.

Choosing Number and Size of Partitions

The number and size of partitions depends on the amount of real and virtual storage available in the system and on any virtual=real requirements that may exist for specific jobs. The best configuration for a particular installation usually depends on the type of job most frequently run.

Assigning Job Classes to Jobs

Every job should be assigned a job class, using the CLASS parameter. For batch processing of input/output-limited jobs, each job should be assigned a job class that corresponds to a high-priority partition. Process-limited jobs should be assigned to a lower-priority partition. In addition, jobs that can be executed without having any special input/output setups (that is, "non-setup" jobs such as a FORTRAN compiler), or that have preallocated data sets, can be directed to a high-priority partition for fast throughput.

Assigning Partitions to Job Classes

After the job classes have been assigned to jobs, appropriate partitions must be assigned to service those jobs. If the partitions do not have the appropriate job classes specified, the job classes can be changed (see *Partition Redefinition* in the *Concepts* section), or the CLASS parameter can be changed on the JOB card, before the job is entered in the input stream.

Choosing Output Classes

At an installation it may be advisable to set up certain output classes for specific duties. For example, output class A could be for system messages, and class B for problem program output. Or, class A could be for system messages, class B for problem program output for the accounting department, class C for problem program output for the purchasing department, etc.

Note: System messages are assigned an output class through the MSGCLASS parameter on a JOB card. Problem program output is assigned an output class through the SYSOUT parameter on a DD card.

Another approach would be one in which output class A represents printer system message output, class B represents punched card output, class C represents magnetic tape output, and class D represents printer problem program output. Up to 36 output classes may be specified. When using special forms on the printer, the operator should ensure that system messages are not written on the special forms. This possibility can be eliminated by establishing a different output class for output requiring the special forms.

Note: An identification problem may arise if system messages are assigned an output class different from problem program output. Therefore, it may be helpful for the programmer to print, as the first line of output, his name and department, if he chooses to use different classes for message and problem program output. This would also alleviate some operator problems. (See *Operating Considerations* in this section.)

When the system log option is present, system log data sets must be assigned an output class. The assignment can be made at system generation by using the WTLCLSS operand of the SCHEDULR macro instruction. Two log data sets are provided for recording the data sent to the log. To avoid the situation where the second data set becomes full before the first data set can be written, both the size of the data sets and the output writer class must be considered at system generation. To be sure that a full log data set is processed in a reasonable period of time, a unique output message class should be assigned for the log data sets and a writer should be assigned multiple output classes with the log class having the highest priority.

Telecommunications

VS1 enables telecommunications jobs to be run concurrently with other types of jobs such as batch, graphics, and concurrent peripheral operation (CPO). Several VS1 considerations are of interest to the telecommunications user.

These considerations include the number of telecommunications partitions required, their placement in the system, their size, and their job class(es).

Choosing Number and Size of Partitions

Telecommunications jobs arc considered unending in that they are scheduled only once, and are terminated only when a CANCEL command is entered, that is, for partition redefinition. (See *Partition Redefinition* in the *Concepts* section.) There must be at least one partition for each telecommunication job being run. The size of the partition depends upon the size of the telecommunications control program used by the installation.

To avoid delays in servicing lines, a telecommunications job should have unrestricted access to the resources of the central processor. For this reason, it is best to run telecommunications jobs in high-priroity partitions. Because the telecommunications job is not alone in the system, its activities should cause minimum interference with jobs in other partitions, and it should not be susceptible to interference from these other jobs.

Assigning Job Classes to Jobs

Each telecommunications job should have a unique job class assigned to it. The message control partition (P0) should have a different job class from the message processing partition. Caution must be taken to avoid assigning job classes to problem programs that correspond to the job class(es) of the telecommunications partitions. Caution must also be taken to assure that all telecommunications jobs are assigned class parameters corresponding to those defined for the high-priority teleprocessing partitions in the system.

Assigning Partitions to Job Classes

Each telecommunications partition should also have a unique job class so that the appropriate jobs may be directed to that partition. If the job classes are to be changed, a CANCEL command must first be entered to terminate the unending job, and then the system may be redefined. (See *Partition Redefinition* in the *Concepts* section.) Likewise, if the partition is not assigned to the telecommunications job class, the telecommunications job may never be initiated.

VS1 Telecommunications Compatibility

VS1 BTAM (Basic Telecommunications Access Method) and VS1 TCAM (Telecommunications Access Method) are the same externally as their equivalents in OS. Both types of programs will run, without modification, in the VS1 environment after reassembly. The reassembly allows the existing programs to benefit from the virtual storage feature of VS1. The user is cautioned that if any internal changes have been made to either OS BTAM or OS TCAM to tailor them for a particular need, similar changes must be made to the VS1 versions to retain their compatibility. For more information about BTAM, refer to *OS/VS Basic Telecommunications Access Method*, GC27-6980. For more information about TCAM, refer to *OS TCAM Concepts and Facilities*, GC30-2022, and *OS/VS TCAM Programmer's Guide*, GC30-2034.

Graphics

Graphic jobs in a VS1 environment are subject to several general considerations. A graphics job associated with an unbuffered IBM 2250 Display Unit may operate with reduced performance if high telecommunications activity interferes with its access to the central processor for regenerating the display. In this case the relative importance of the graphics and telecommunications jobs must be determined, and the decision made as to which to run in the higher-priority partition. Additional considerations for VS1 include assigning job classes to jobs, choosing the partition to service graphics jobs, using the time-slicing option, and assigning partitions to job classes.

Choosing Number and Size of Partitions

There must be at least one partition for each graphics job being run. The partition size depends upon the size of the graphics job. Generally, graphics jobs should be run in a high-priority partition to cause minimum interference with other jobs. If telecommunications and graphics are being run in the same system, the best performance would be gained by placing the telecommunications job in a high-priority problem program partition, and the graphics job in a relatively high-priority partition also.

Assigning Job Classes to Jobs

Graphics jobs should have a unique job class assigned to them, to ensure that they are executed in the selected partition.

Assigning Partitions to Job Classes

A graphics partition should be assigned a unique job class that corresponds to the job classes assigned to the graphics jobs. This ensures that jobs will be enqueued on the proper input queue, and executed in the appropriate partition. The partition could also be assigned secondary and tertiary job classes to reduce idle time. If the partition's job class is to be changed, and a graphics job is being run, a CANCEL command must be issued for the graphics job, and then the partition may be redefined.

Using the Time-Slicing Feature

The time-slicing feature of assigning uniform intervals of CPU time to a group of consecutive partitions is provided at system generation. (The number of time-slicing partitions and the time interval for each task are specified in the TMSLICE parameter of the CTRLPROG macro instruction.) The ability to get uniform response time is useful in a graphics environment, particularly for concurrent applications involving graphics terminals. To minimize contention for the CPU with other jobs, it is best to establish the higher-priority partitions as time-slicing partitions.

Graphics Support with VS1

The Graphic Subroutine Package (GSP) provides IBM 2250 graphics support for PL/I, FORTRAN, and COBOL F and is externally equivalent to OS MFT Release 21.0.

The Graphics Access Method (GAM) provides support for the IBM 2250 and IBM 2260 display units and is externally equivalent to OS MFT Release 21.0.

For more information about GSP, refer to the *OS/VS Graphic Subroutine Package (GSP) for FORTRAN IV, COBOL, and PL/I,* GC27-6973. For more information about GAM, refer to the *OS/VS Graphic Programming Services (GPS) for the IBM 2250 Display Unit,* GC27-6971, and to the *OS/VS Graphic Programming Services (GPS) for the IBM 2260 Display Station (Local Attachment),* GC27-6972.

Concurrent Peripheral Operations

Concurrent Peripheral Operation (CPO) is the capability of performing utility functions such as card-to-tape, tape-to-print, or tape-to-punch while other jobs in the system continue processing. Execution of CPO jobs in VS1 involves the same general considerations for assigning job classes to jobs and partitions as for telecommunications and graphics jobs. CPO jobs should be assigned a class that corresponds to that of the CPO partitions. The CPO partition should be assigned a job class different than that for any other partition.

Typical System Configurations

This topic describes partition configurations for systems with 128K, 192K and 384K bytes of real storage. These configurations are based on the considerations presented in the preceding four topics. Working configurations will depend on the individual requirements of each installation. The values for virtual=real areas shown in these configurations do not represent the actual size of the space available for V=R jobs. This is due to dynamic allocation of real storage by the system. See the *Storage Estimates* publication for estimates of the actual space requirements for any specific configuration.

Note: In Figures 22–25, do not assume that the portions of real storage above (to the right of) SQA space exist as shown in the figure; pages of JES, problem programs, etc., may be scattered throughout the available real storage. Also, this area above the SQA space includes the 36K reserved by the system (see the earlier topic *Virtual=Real Storage Availability*).

Systems with 128K Real Storage

A 128K system can support one configuration. The two parts of Figure 22 show possible storage configurations, real and virtual, for a minimum system, and illustrate the relationship of real and virtual storage.

Systems with 192K Real Storage

A 192K system makes possible a variety of configurations. Depending on the requirements of the installation, the most likely configurations will include two large (128K or larger) batch partitions, or three or four smaller (64K or larger) batch partitions. In either case, several system output writers could be provided to support the batch partitions. Figure 23 illustrates the first case, with two large batch partitions.

Systems with 384K Real Storage

The choice of configurations available with 384K bytes of real storage is so great that no "typical" system can be defined. Figure 24 shows a possible configuration with five batch partitions.

Figure 25 shows a possible configuration with a teleprocessing job running with four batch partitions of varying sizes.

Operating Considerations

The operator of a VS1 system must be aware of several considerations related primarily to program execution, partition definition, output class reassignment, restarting the system, and operator commands. If the system has the shared DASD option, the operator must also consider shared volume handling. These considerations are explained in the following paragraphs.

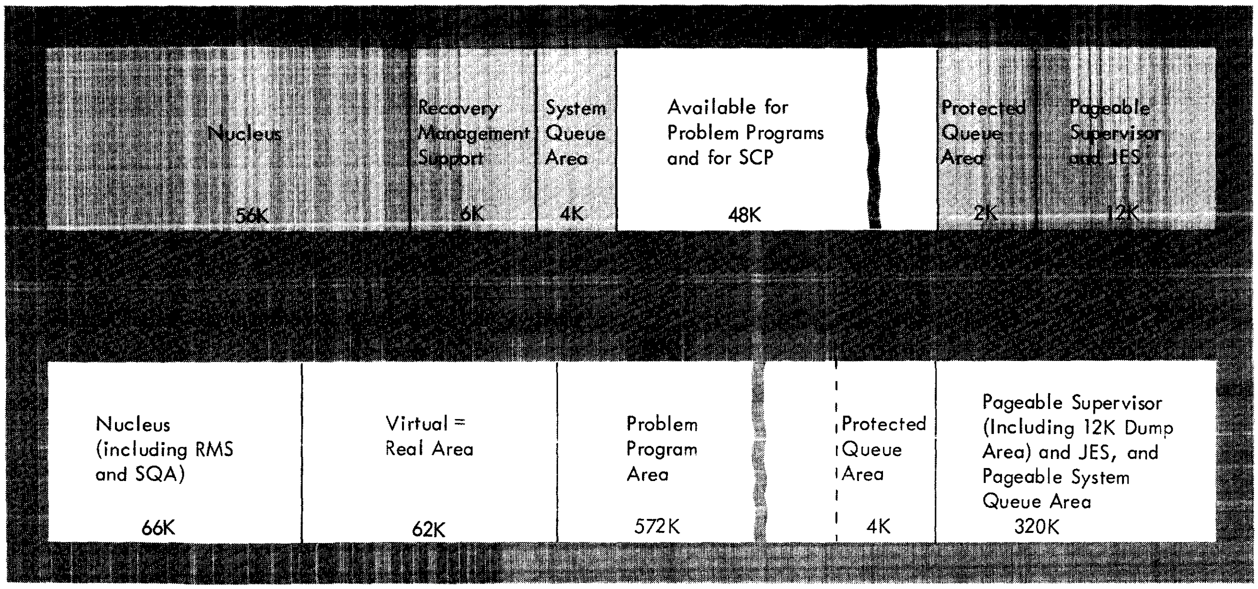

Nucleus
56K
Recovery Management Support
6K
System Queue Area
4K
Available for Problem Programs and for SCP
48K
Protected Queue Area
2K
Pageable Supervisor and JES
12K
Nucleus (including RMS and SQA)
66K
Virtual = Real Area
62K
Problem Program Area
572K
Protected Queue Area
4K
Pageable Supervisor (Including 12K Dump Area) and JES, and Pageable System Queue Area
320K

Figure 22. Storage Configurations for 128K System

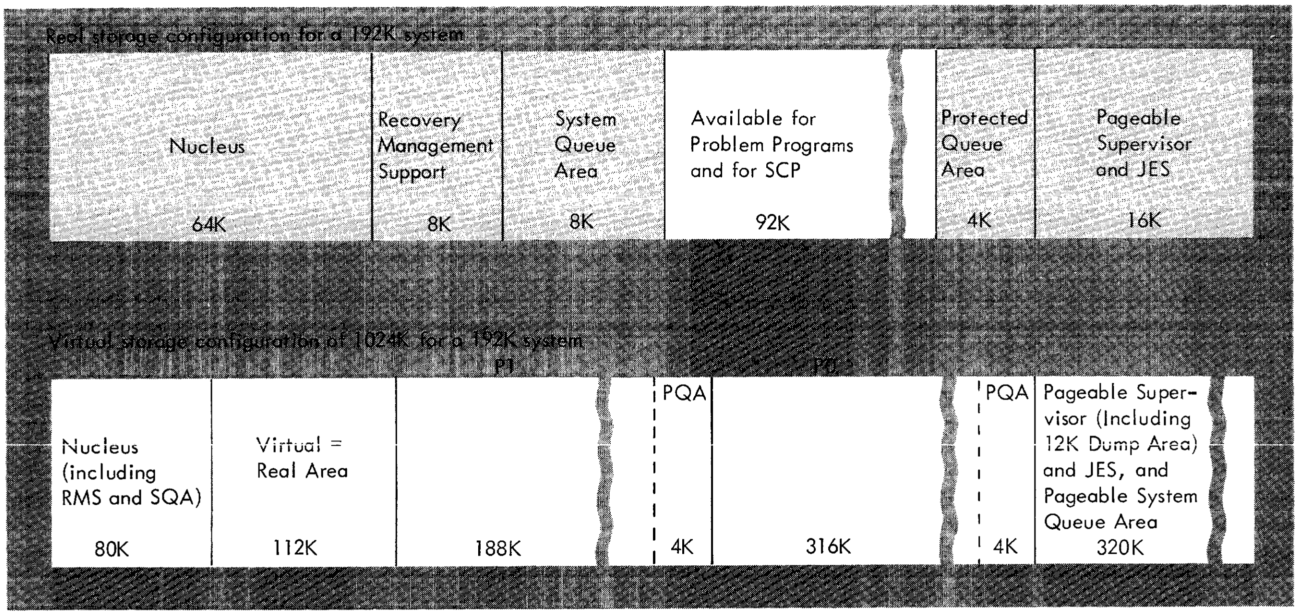

Real storage configuration for a 192K system
Nucleus
64K
Recovery Management Support
8K
System Queue Area
8K
Available for Problem Programs and for SCP
92K
Protected Queue Area
4K
Pageable Supervisor and JES
16K
Virtual storage configuration of 1024K for a 192K system
P1
P0
Nucleus (including RMS and SQA)
80K
Virtual = Real Area
112K
188K
PQA
4K
316K
PQA
4K
Pageable Supervisor (Including 12K Dump Area) and JES, and Pageable System Queue Area
320K

Figure 23. Storage Configurations for a 192K System

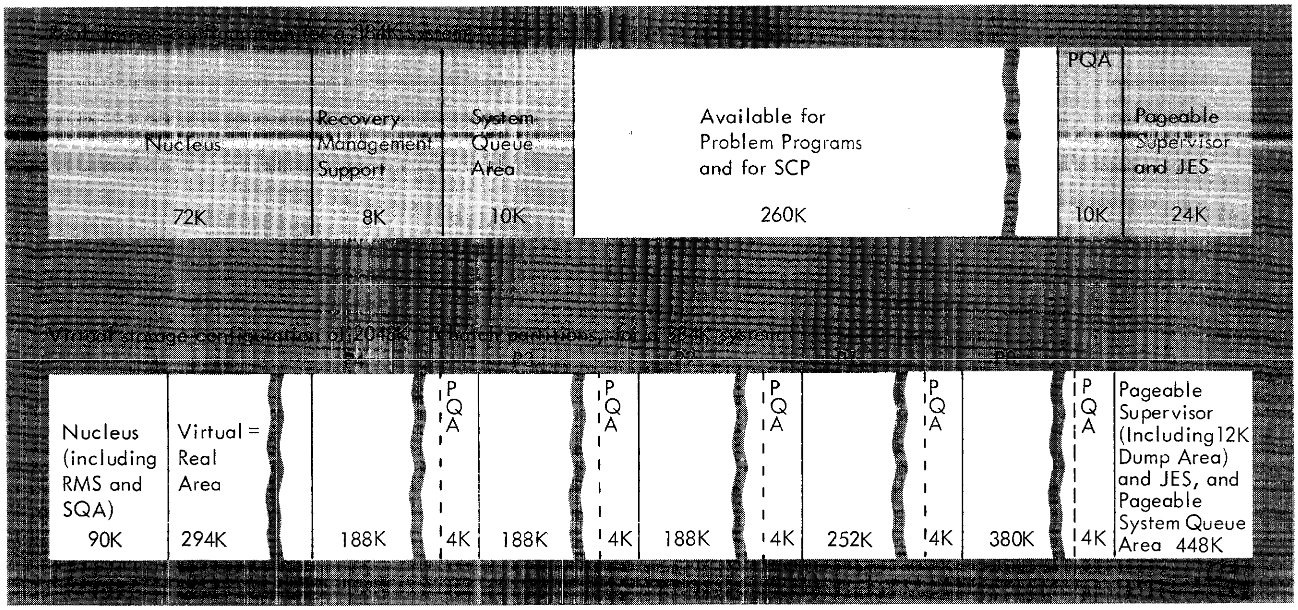

Real storage configuration for a 384K system
Nucleus
72K
Recovery Management Support
8K
System Queue Area
10K
Available for Problem Programs and for SCP
260K
PQA
10K
Pageable Supervisor and JES
24K
Virtual storage configuration of 2048K, 5 batch partitions, for a 384K system
P4
P3
P2
P1
P0
Nucleus (including RMS and SQA)
90K
Virtual = Real Area
294K
188K
PQA
4K
188K
PQA
4K
188K
PQA
4K
252K
PQA
4K
380K
PQA
4K
Pageable Supervisor (Including12K Dump Area) and JES, and Pageable System Queue Area 448K

Figure 24. Storage Configurations for a 384K System

Program Execution

Because 15 problem programs can be executed concurrently, the system places additional responsibility on the VS1 operator. At times he may become busy replying to system messages and problem program messages, placing special forms in the printer, etc. Therefore, whenever possible, he should perform preparatory work, such as obtaining and/or mounting required volumes, ahead of the required time. When responding to problem program messages, the operator should respond to the highest priority task first; that is, the message from the partition with the lowest number. The operator must also remember that problem program and system messages may be intermingled on the console device.

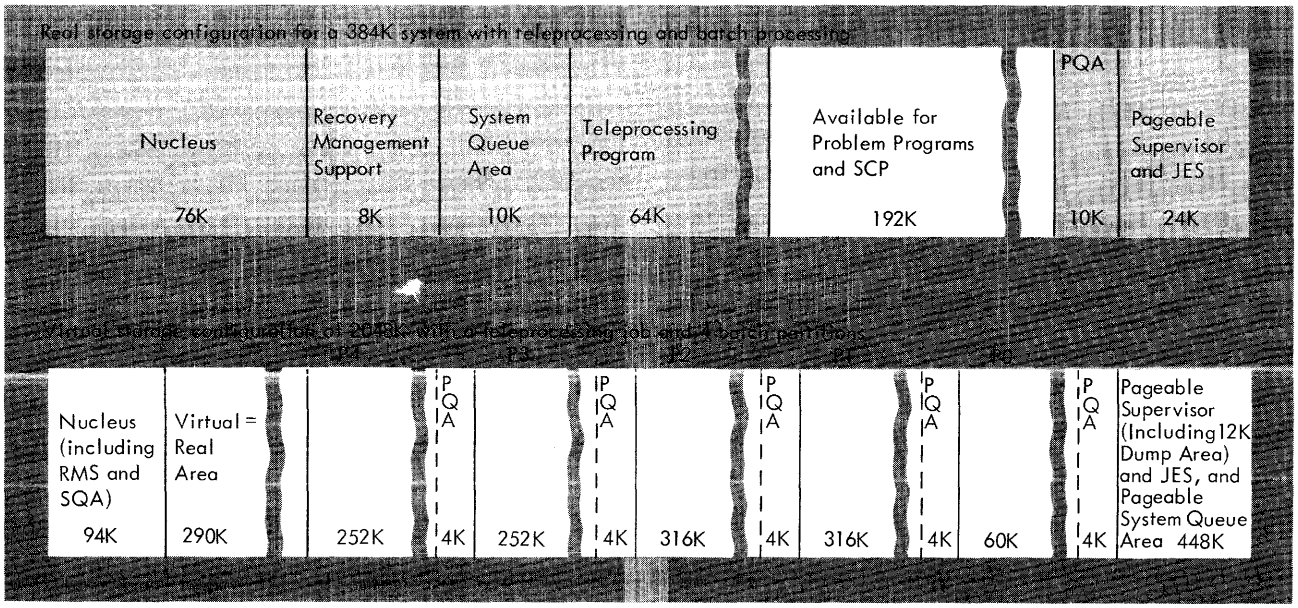

Real storage configuration for a 384K system with teleprocessing and batch processing
Nucleus
76K
Recovery Management Support
8K
System Queue Area
10K
Teleprocessing Program
64K
Available for Problem Programs and SCP
192K
PQA
10K
Pageable Supervisor and JES
24K
Virtual storage configuration of 2048K with a teleprocessing job and 4 batch partitions
P4
P3
P2
P1
P0
Nucleus (including RMS and SQA)
94K
Virtual = Real Area
290K
252K
PQA
4K
252K
PQA
4K
316K
PQA
4K
316K
PQA
4K
60K
PQA
4K
Pageable Supervisor (Including 12K Dump Area) and JES, and Pageable System Queue Area 448K
25

Figure 25. Storage Configurations for Teleprocessing and Batch Processing

In addition, because jobs may not be completed in the same order as they were entered in the system, the operator must ensure that the correct output is returned to each programmer.

The operator may also be required to start system input readers and output writers at certain times during operation. He may be given a specific time each day, or may have to use his judgment based on work load for the system.

Partition Definition

Even though the installation may not intend to use the maximum number of partitions at all times, the system must be regenerated if the number of partitions originally specified is increased. Therefore, the maximum number of partitions expected to be used should be specified at system generation. Partitions can then be redefined to decrease the number in use.

Caution must be observed when redefining partitions. Before redefining partitions, the operator should check the job class(es) of all pending jobs and ensure that the prospective partition definitions have job classes corresponding to the jobs that will be executed. This includes knowing the job classes of jobs which have already been placed on the input or hold queues, but have not been executed. (This can be accomplished by issuing the DISPLAY Q command.) If possible, he should also check pending jobs for their size requirement (by checking the job class versus the size of the partition assigned to service that job class) and compare this with the size of the job partitions. If they are originally assigned a CLASS parameter that corresponds only to a large partition, they should be reassigned to a large partition.

If the time-slicing feature is included in the system, the operator should not specify the same job class for both a time-sliced partition and a partition that is not time-sliced. For example, do not specify a partition with job classes B, C, D in a time-sliced group, and a partition with job classes D,E,F outside the group. Doing so would allow a job with a CLASS parameter of D to be executed either inside or outside the time-sliced group regardless of the programmer's intentions for that job. Also, a partition in a time-sliced group should not be assigned to service jobs with job classes of A, because A is the default job class, and the same problem could arise.

After all redefinitions have been completed, message IEF805I 'DEFINITION COMPLETED' is issued. The operator must enter either a START INIT command for each of the partitions that have been redefined, or a START INIT. ALL command.

Partition Deactivation/Reactivation

This function enables the operator to declare a partition eligible or ineligible for deactivation, or to reactivate any deactivated partition. The function is available at IPL time and, through the use of the DEFINE command, after IPL. The operator may display the current status of the partitions by using the LIST option of the DEFINE command, or by using the DISPLAY A command.

At nucleus initialization time, the operator can vary the function of *timed task reactivation* by responding to message IEA101A SPECIFY SYSTEM AND/OR SET PARAMETERS FOR RELEASE xx.yy.sss with the keyword REACT=*d*. *d* is a decimal digit from 0—9, the number of seconds for the timed interval. The value is used by the page supervisor at *system wait time* in an attempt to reactivate the highest priority partition currently deactivated. Task reactivation is executed when the specified time interval has elapsed, the paging rate has diminished to zero, and sufficient real storage has become available to reinstate the deactivated task. The fewer the number of seconds used for the interval, the more likely a partition may become reactivated. However, the availability of real storage space for pages is still the prime prerequisite for reactivation.

If REACT=*d* is not specified, deactivation/reaction works as follows:

- When a partition is deactivated, the time elapsed since the last reactivation is used to set a *delay interval*. When that interval elapses, the *zero page-rate interval* is set. This second interval is determined by the number of active tasks. When no paging for the zero page-rate interval has occurred, and enough real storage is available, the highest priority task is reactivated. The delay interval is reset to its maximum value, determined by the real storage size.

Though the operator can declare any partition ineligible for deactivation, he should exercise care in the selection. A problem program partition should be declared ineligible for deactivation only if activity within that partition is judged to be critical to the processing environment. For example, if a user-written teleprocessing application was deactivated (by the page supervisor), the user's entire teleprocessing network might fail.

Whenever the system operator reinstates a currently deactivated job, time should be the only criterion in the decision. For example, a job such as a payroll run might have to be "rushed" through the system. Whenever such a decision is made, other active tasks should be eligible for deactivation. If this is not done, the amount of real storage available for paging may be decreased to such a low level that the currently active partitions cause the system to run inefficiently. That is, each active task requires pages for itself, which in turn causes another task to begin paging, and so forth. This condition is called "thrashing". For all practical purposes, a system that is thrashing is running only the page supervisor task.

When the operator determines that a partition has been deactivated for a relatively long period of time, several actions are available:

- He can specify that the deactivated partition be reactivated for the duration of the job executing in that partition. At job completion, the partition is then elegible for deactivation should a shortage of pages develop again.
- He can put a hold on the job queue. As jobs in the active partitions end, the deactivated partitions can become active and complete their tasks. The queue can then be released and processing can continue.
- He can stop an active partition. Assuming that stopping the partition reduces the paging activity, the results would be comparable to putting a hold on the job queue.
- He can cancel the job in the deactivated partition, stop the partition, and re-enter the job in the job queue where it can be selected by another partition.

Regardless of what technique is used, it is imperative to remember that the partition was deactivated because its activity was detrimental to the system as a whole. With this in mind, it would be wise in most cases to stop a deactivated partition after applying one of the preceding methods.

For specifying partition eligibility for deactivation and reactivation, see message IEE802A and message IEE803A in the *OS/VS System Messages* publication.

Changing Output Classes

The output classes with which a writer is associated can be changed at any time, through proper use of the MODIFY command, or a combination of STOP and START commands. A program with a special forms requirement can obtain exclusive use of a printer by informing the operator to enter a MODIFY command. A STOP command followed by a START command for the same writer, but specifying a unique output class, could also be entered. The STOP command causes the writer to stop at the end of the job it is currently executing. The operator then inserts the required forms and issues the new START command. That command would limit use of the printer to the data set associated with the new output class until another STOP and START command sequence for the printer is issued. The MODIFY command can also be used to change the conditions under which the output writer pauses for servicing of its device.

For example, a writer named ONE, originally established to service output classes A, B, and C, could be changed to service only data sets for output class D by issuing the command:

```
MODIFY WTR.ONE,CLASS=D
```

Handling Shared Direct Access Volumes

If the shared DASD feature is selected at system generation, additional responsibilities are imposed on the operator. Volume mounting and dismounting instructions are normally issued by the operating system. In a shared DASD environment, volume handling must be initiated by the operator and must be conducted in parallel on both sharing systems. Thus thorough operator communication from system to system must be maintained.

Restarting the System

To restart the system after it has been shut down, the same steps taken in initially starting the system are followed, except when the SET command is entered. Either the "F" suffix from the "Q=(unitname, [F])" parameter is omitted, or the entire "Q= " parameter is omitted.

The following command illustrates this procedure:

```
SET DATE=yy.ddd,CLOCK=hh.mm.ss
```

By omitting the "Q= " parameter, job queue data set information is saved. When restarting the system to save the information, the operator must make certain that all auxiliary storage volumes which were in use remain available. This ensures that the job queue data set, spool data sets, and SWADS data sets accurately reflect the conditions which existed when a restart became necessary.

Operator Commands

The following commands, and their respective abbreviations, may be used in a VS1 system:

CANCEL	C	RELEASE	A
CONTROL	K	REPLY	R
DEFINE	N	RESET	E
DISPLAY	D	SET	T
DUMP		START	S
HALT	Z	STOP	P
HOLD	H	STOPMN	PM
LOG	L	SWAP	G
MODE		SWITCH	I
MODIFY	F	UNLOAD	U
MONITOR	MN	VARY	V
MOUNT	M	WRITER	WTR
MSGRT	MR	WRITELOG	W

Note: The commands are subject to the following restrictions:

- The DEFINE, HALT, MODE, SWAP, and WRITER commands are not allowed in the input stream; they must be entered through a console.
- The DUMP and MODE commands cannot be abbreviated.

Be sure to use the correct abbreviations for operator commands. For example, at system initialization, if you inadvertently key in S for SET, the system assumes you are giving a START command. It queues the command, and waits for a SET command.

The Use Guide

The *Use Guide* contains sections covering the following topics:

Handling Accounting Information
Automated System Initialization
OS/MFT-OS/VS1 Differences
VS1 Features and Options
JES Reconfigurability
Job Queue Format
Message Routing Exit Routines
The Must-Complete Function
The PRESRES Volume Characteristics List
System Reader, Initiator, and Writer Cataloged Procedures
Resident Routines Option
Output Separation
The Shared Direct Access Device Option
System Macro Instructions
Adding SVC Routines to the Control Program
How to Use the Tracing Routine
The Time Slicing Facility
Writing System Output Writer Routines
Appendix A: Theory of Operations
Glossary

Publication References

Reference is made in the *Use Guide* to other OS/VS publications. These references do not include the complete title and order number of the publication. To facilitate use of this publication, complete titles and order numbers of the referenced books are indicated here.

OS/VS Data Management Services Guide	GC26-3783
OS/VS Data Management Macro Instructions	GC26-3793
OS/VS JCL Reference	GC28-0618
OS/VS Message Library Publications	
VS1 System Messages	GC38-1001
System Codes	GC38-1003
Routing and Descriptor Codes	GC38-1004
Operator's Library Publications	
OS/VS1 Reference	GC38-0110
OS/VS Console Configurations	GC38-0120
OS/VS1 RES System Programmer's Guide	GC28-6878
OS/VS Service Aids	GC28-0633
OS/VS1 Storage Estimates	GC24-5094
OS/VS Supervisor Services and Macro Instructions	GC27-6979
OS/VS1 System Data Areas	SY28-0605
OS/VS System Generation Introduction	GC26-3790
OS/VS1 System Generation Reference	GC26-3791
OS/VS Utilities	GC35-0005

Conventions used in Illustrations of Coding

Certain conventions are used to illustrate the format of macro instructions included in this publication. These conventions are:

- Letters in capitals, numbers, and punctuation marks (except as noted below) must be coded exactly as shown.
- Brackets, [ ]; braces, { } ; ellipses, . . .; and subscripts are never coded.
- Lowercase letters represent variables for which you must substitute specific information or specific values.
- Items or groups of items within brackets are optional. They may be omitted at your discretion. Conversely, the lack of brackets indicates that an item or group of items must be coded.
- Stacked items enclosed in braces represent alternative items. Only one of the stacked items should be coded.
- If an alternative item is underlined, it is the default value. The system will automatically assume it is your choice if none of the items is coded.
- An ellipsis indicates that the preceding item or group can be coded two or more times in succession.

Handling Accounting Information

You may add accounting facilities to your vs1 operating system. This section describes the input available to an accounting routine; the characteristics and requirements of an IBM-supplied data set writer that may be used to log accounting information generated by an accounting routine; and how to insert an accounting routine into the control program. Conventions to be followed in preparing an accounting routine are also noted.

Section Outline

Accounting Routines

Your installation may prepare accounting routines for insertion in your VS1 operating system. These routines are inserted in the control program during, or after, system generation.

Prerequisite Actions

At system generation you must specify that an accounting routine is to be supplied. This is done through the SMF=parameter of the system generation SCHEDULR macro instruction.

This specification causes the linkage to your accounting routine to be installed in the scheduler component of the system being generated, and makes usable the accounting data set writer routine. If you are not going to install your accounting routine until after the system is generated, a dummy accounting routine (named IEFACTRT) is also placed in the system at this time. Insertion of accounting routines in the control program is discussed later in this section.

Accounting Routine Conventions

Format

Your accounting routine may consist of one or more control sections.

Attribute

An accounting routine written for insertion in your VS1 operating system must be serially reusable.

CSECT Name and Entry Point

The control section containing the entry point of your accounting routine, and the entry point, must be named IEFACTRT.

Register Saving and Restoring

The content of registers 0 through 14 must be saved upon entry to your accounting routine and restored prior to exiting.

Entrances

Control is given to your accounting routine at the following times:

- Step initiation
- Step termination
- Job termination

Exit

You can use the RETURN macro instruction to restore the contents of the general registers and return control to the operating system.

Input Available to Accounting Routines

Register 0 contains an entrance code, indicating at what time the accounting routine is being given control.

Register 0 = 8: Step initiation
= 12: Step termination
= 16: Job termination

Register 1 contains the starting address of a list of pointers to items of accounting information. Each pointer is on a fullword boundary. The sequence of pointers in the list and the items of information provided are described in Figure ACC 1.

User accounting routines should only use pointers that are in the list addressed by register 1. Other pointers are subject to change in subsequent releases.

Output from Accounting Routines

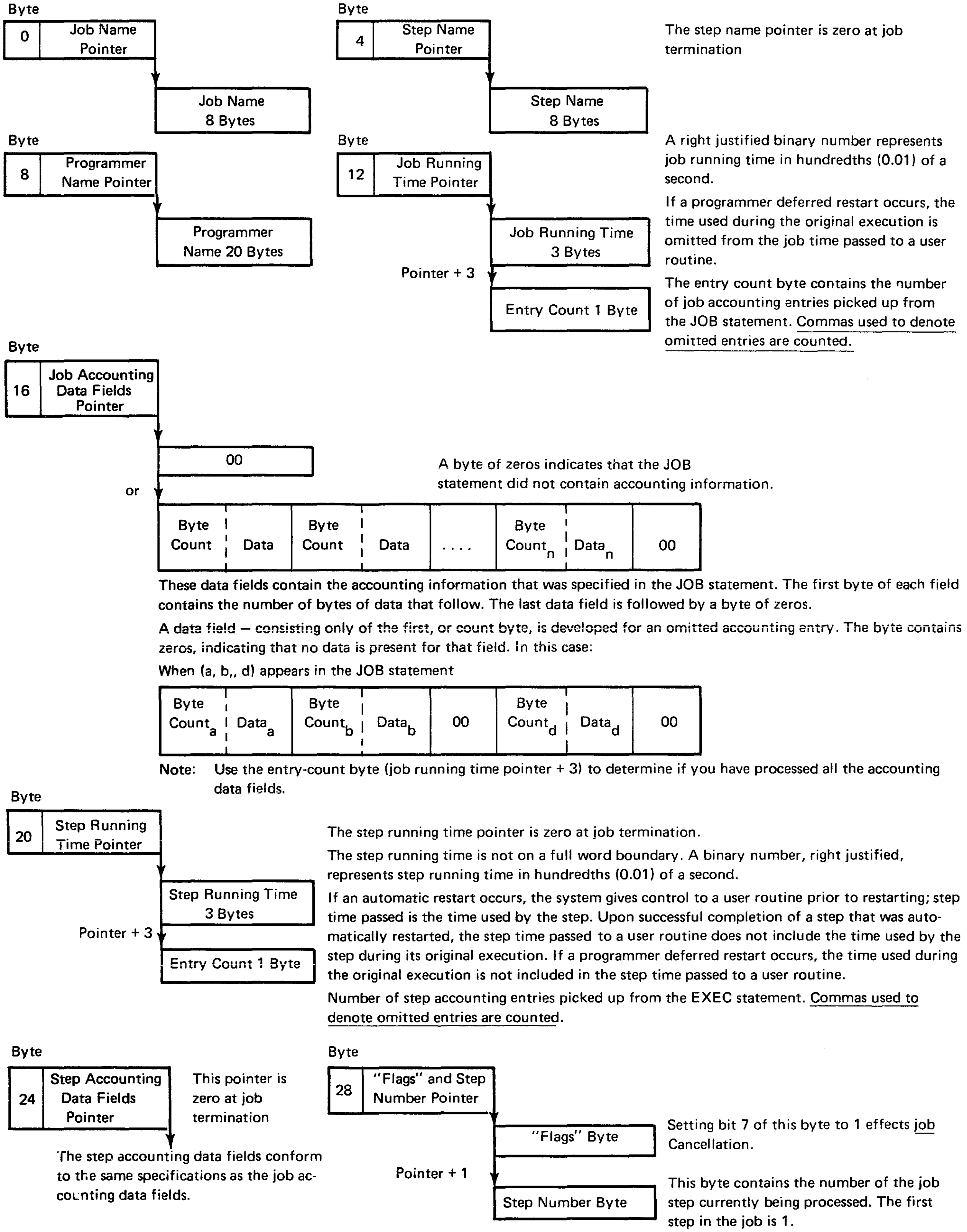
You can write output in three ways: by issuing console messages; by using the standard system output; by using an IBM-supplied accounting data set writer.

1. *Console messages*—You can use write to operator (WTO) or write to operator with reply (WTOR) macro instructions.
2. *System output*—Assemble the following calling sequence into your routine. The contents of register 12 must be the same as when your accounting routine was entered, and register 13 must contain the address of an area of 32 fullwords.

   When writing an accounting routine for inclusion in the job scheduler, you must be aware that register saving conventions within the control program are different from those for problem programs. In the job scheduler, registers are saved in the sequence 14-12 in a 15-word save area. There is no place provided to save register 13. You must provide some other means of saving register 13; you may either save it in another register or provide an additional save area that is not known to the control program. This can be done by adding a word to the end of the save area that is provided and is addressed as SAVE + 60.

Name	Operation	Operand	
	MVC	36 (4, 12), MSGADDR	MOVE MESSAGE ADDRESS
	MVC	42 (2, 12), MSGLEN	AND LENGTH TO SYSTEM
	L	REG15, VCONYS	TABLE BRANCH AND LINK
	BALR	REG14, REG15	TO MESSAGE ROUTINE
	.		
	.		
MSGADDR	DC	A (MSG)	
MSG	DC	C'text of message'	
MSGLEN	DC	H 'two character length of message'	
VCONYS	DC	V (IEFYS)	

3. *Accounting data set writer*—This writer places accounting records you have constructed in your accounting routine in a data set named SYS1.ACCT. The data set must reside on a permanently resident direct access device. You must provide, in your accounting routine, linkage to the writer, and pass the beginning address of the record to be written, to it. Use of the data set writer is covered later in this section.



The step name pointer is zero at job termination

A right justified binary number represents job running time in hundredths (0.01) of a second.

If a programmer deferred restart occurs, the time used during the original execution is omitted from the job time passed to a user routine.

The entry count byte contains the number of job accounting entries picked up from the JOB statement. Commas used to denote omitted entries are counted.

A byte of zeros indicates that the JOB statement did not contain accounting information.

Byte Count	Data	Byte Count	Data	....	Byte $Count_n$	$Data_n$	00

These data fields contain the accounting information that was specified in the JOB statement. The first byte of each field contains the number of bytes of data that follow. The last data field is followed by a byte of zeros.

A data field – consisting only of the first, or count byte, is developed for an omitted accounting entry. The byte contains zeros, indicating that no data is present for that field. In this case:

When (a, b,, d) appears in the JOB statement

Byte $Count_a$	$Data_a$	Byte $Count_b$	$Data_b$	00	Byte $Count_d$	$Data_d$	00

Note: Use the entry-count byte (job running time pointer + 3) to determine if you have processed all the accounting data fields.

The step running time pointer is zero at job termination.

The step running time is not on a full word boundary. A binary number, right justified, represents step running time in hundredths (0.01) of a second.

If an automatic restart occurs, the system gives control to a user routine prior to restarting; step time passed is the time used by the step. Upon successful completion of a step that was automatically restarted, the step time passed to a user routine does not include the time used by the step during its original execution. If a programmer deferred restart occurs, the time used during the original execution is not included in the step time passed to a user routine.

Number of step accounting entries picked up from the EXEC statement. Commas used to denote omitted entries are counted.

This pointer is zero at job termination

The step accounting data fields conform to the same specifications as the job accounting data fields.

Setting bit 7 of this byte to 1 effects job Cancellation.

This byte contains the number of the job step currently being processed. The first step in the job is 1.

Note: You can use the flag byte to cancel the execution of a job whose accounting information does not conform to your installation's standards. You can equate step initiation for the first step in a job to job initiation, i.e., the step number byte contains 1.

Figure ACC 1. Accounting Information Available to User

Sample Accounting Routine

A sample accounting routine, showing use of the data set writer, output to system output, and issuance of console messages, is stored under the member name SAMACTRT in the SYS1.SAMPLIB data set furnished with the starter operating system.

Inserting an Accounting Routine into the Control Program

Your accounting routine can be inserted in the control program in two ways; by placing the routine on the SYS1.AOS00 data set used in system generation or by placing the routine in the appropriate load module of the control program after system generation. The effect of either action is to replace a dummy accounting routine with your accounting routine.

Insertion at System Generation

To insert your accounting routine into the control program during system generation, you must, prior to the start of the system generation process, place your routine in the SYS1.AOS00 data set, using the linkage editor. The SYS1.AOS00 data set (furnished with the starter operating system) contains load modules which are combined during the system generation process to form the load modules composing the control program. In response to the specification made in the system generation SCHEDULR macro instruction, your accounting routine is incorporated in the appropriate load modules for the system being generated.

You must place your accounting routine in the SYS1.AOS00 *data set under the name* IEFACTRT. You will be replacing the dummy accounting routine—also named IEFACTRT.

Insertion after System Generation

To insert your accounting routine into the control program after system generation you place the routine in load modules of the scheduler component of the generated control program, using the linkage editor. *The scheduler load modules are in the linkage library* (SYS1.LINKLIB *data set*) *of the generated system.* The affected load modules are as follows:

load module IEFW21SD—step initiation
load module IEFSD161—step/job termination

An example of the input for a linkage editor run to insert your accounting routine into the job scheduler follows:

```
//jobname      JOB      (parameters)
//stepname     EXEC     PGM=IEWL, (parameters)
//SYSPRINT     DD       SYSOUT=A
//SYSUT1       DD       UNIT=SYSDA,SPACE=(parameters)
//SYSLMOD      DD       DSNAME=SYS1.LINKLIB,DISP=OLD
//SYSLIN       DD       *
                .
                .
                .                                     This sequence must be
          (object code)                               repeated for each
                .                                     scheduler load module
                .                                     into which you wish
                .                                     to insert accounting
             INCLUDE    SYSLMOD(load module name)    routines.
                ALIAS   alias names
                ENTRY   entry point name
                NAME    load module name(R)
```

In this example "load module name" represents the appropriate scheduler load module as identified in the preceding text. *To ensure accuracy in identifying the correct alias names and entry point names for the load modules, obtain these names from the system generation listing produced during generation of the system you are working with.* These names are specified in the system generation Stage II linkage editor output for the linkage editor execution that produced the load module.

Accounting Data Set Writer

The accounting data set writer (module IEFWAD) is inserted in the appropriate scheduler load modules during system generation when accounting routine inclusion is specified in the SCHEDULR macro instruction. These are the same modules in which your accounting routine is inserted. Scheduler storage requirements are increased by the amount of storage needed by your accounting routine plus 2600 bytes. The writer places accounting records developed by your routine in a data set named SYS1.ACCT.

Linkage

Your accounting routine links to the writer via the following mechanism:

```
             L          R15,VCON
             BALR        14,15
              .
              .
              .
VCON         DC         V(IEFWAD)
```

Input

Your accounting routine passes in register 1 the address of the accounting record to be written.

The record format is:

DS 3H – space used by the data set writer

DC H'__' – contains the number of bytes of data being passed. This number cannot exceed the capacity of 1 track on the direct access volume being written on.

DC __ – the data to be written in SYS1.ACCT.

Registers 13, 14, and 15 are used as specified by operating system conventions (14 and 15 are used for linkage, as previously shown; 13 must point to an 18-word save area).

Specifying the SYS1.ACCT Data Set

The SYS1.ACCT data set must be pre-allocated on a direct access volume that will be permanently resident. The data set must be named SYS1.ACCT, have no secondary extents, and be allocated contiguous space. *Do not catalog the data set.*

If your installation has two permanently resident volumes available for accounting routine use, you may create two SYS1.ACCT data sets and utilize the console messages and replies to notify the system of the data set to be addressed.

Output

If the IEFWAD routine successfully writes your record in the SYS1.ACCT data set, the routine returns control to your accounting routine immediately. If the routine fails to write your record, it uses message IEF507D to bring the error condition to the attention of the operator. (See the appropriate *OS/VS Messages Library* publication for the text of, and answers to the message.) Depending upon his answer, the routine may try again to write your record in the SYS1.ACCT data set.

In any case, a code is returned to your routine indicating either that the record was written successfully, or, if it was not written successfully, the cause of the failure. The return codes are described in the following:

Contents	Type	Meaning
Register 15		
0	D	The record was written to the data set.
4	D	The record was not written to the data set because the record exceeds the length of one track.
8	D	The record was not written to the data set because there is no more space in the data set.
12	D	The record was not written to the data set because no space had been allocated to the data set.
16	D	The record was not written to the data set because a permanent I/O error was encountered while trying to write it.
20	D	The record was not written to the data set because the previously last record could not be found.
24	D	Operator gave invalid device address.
Register 0		
n	B	Number of tracks still available in the data set. (Valid only if register 15 is zero.)
Type — Type of number: D — Decimal, B — Binary		

Use of ENQ/DEQ

IEFWAD enqueues on the major queue name SYSIEFSD and the minor queue name WD.

Automated System Initialization

This section describes the use of the automated system initialization feature. This feature makes the system initialization process quick and flexible through the use of SYS1.PARMLIB data set members to hold system initialization parameters. Use of this feature significantly reduces the operator's role in the initialization process.

Note: The term initialization here refers to the period beginning when the IPL program is loaded and ending when the system is ready to perform meaningful work for the user. It involves all responses and commands issued by the operator until the system is initialized.

Section Outline

Advantages of Automated System Initialization

Automated system initialization is a standard feature available for use at your convenience. It requires no changes in system generation options and provides the following advantages over the other procedures, which involve the operator's making lengthy entries on the system console:

- The responsibility of altering the system is placed directly on the system programmer.
- The flexibility of keeping system initialization parameters in SYS1.PARMLIB members permits each system initialization to be a tailoring process that enables the system to better meet the needs of the anticipated job mixture.
- The time needed for initialization is reduced.
- The operator's role is reduced, thus freeing him to do other tasks.
- The operator may see only one entry across systems, thus eliminating confusion over which system is being initialized.
- Informational messages not critical to the operator can be eliminated until the initialization is complete.
- Redefinition of partition sizes, job classes, and time slicing can be done with a minimum of operator involvement.

The Automated System Initialization Process

Use of automated system initialization involves first the creation of members in SYS1.PARMLIB and then the processing of these members by the nucleus initialization program (NIP) and the master scheduler initialization (MSI) program:

Process	Comments
Before Initialization	
System programmer creates initialization members in SYS1.PARMLIB by using the IEBUPDTE utility.	Naming conventions must be followed.
At Initialization	
NIP request operator to "SPECIFY SYSTEM AND/OR SET PARAMETERS".	The keyword identifies either a member of SYS1.PARMLIB or a card reader. The member or card deck so identified lists the members of SYS1.PARMLIB that hold the initialization parameters to be used.
NIP uses the entries in the list to modify the standard default list of SYS1.PARMLIB members to be used.	
NIP references the modified list to get the name of the system parameters member. NIP processes this member and then continues with the other parameters entered by the operator.	The operator is advised if any abnormal conditions or invalid parameters are detected.
NIP passes the modified list to MSI and gives control to MSI, which processes all other members indicated on the modified list.	The operator is advised if any abnormal conditions or invalid parameters are detected.
Initialization is complete.	

Implementation of Automatic Commands

Although system generation options used in os/vs1 Release 1 have not been changed for automatic commands, a new implementation is now used. Any automatic commands must be put in an automatic commands member of SYS1.PARMLIB and automated system initialization must be invoked. The automatic START commands formerly generated via the STARTI, STARTR, and STARTW parameters of the SCHEDULR system generation macro must be placed in an automatic commands member. A system generation deck containing these parameters can be used to generate the new-release system, but the parameters are ignored.

Creating SYS1.PARMLIB Members

You can use the IEBUPDTE utility program to place parameter lists in members of the system parameter library, SYS1.PARMLIB. (This utility is also used to maintain the members of SYS1.PARMLIB.) The parameter lists are composed of 80-byte records, formatted much like the entries that the operator would be required to make in a manual initialization. The format, contents, and processing of these parameter lists are described in the following paragraphs.

Naming Conventions for SYS1.PARMLIB Members

The names used for the SYS1.PARMLIB members that hold parameters for use in automated system initialization signify the types of parameters they hold. Each name consists of from three to eight characters, the first three of which signify the member's contents:

First Three Characters	*Contents of the Member*
NIP	System Parameters
JES	JES Reconfiguration Parameters
DFN	DEFINE Parameters
SET	SET Parameters
PRE	Permanently Resident Volume List Parameters
CMD	Automatic Commands
SMF	SMF Parameters
RES	RTAM Parameters

The name used for a member of SYS1.PARMLIB that lists the other members to be used in an initialization is not bound by this "first three characters" convention.

Formats of SYS1.PARMLIB Automated Initialization Members

In general, the formats of the automated system initialization members are the same as the formats that the operator would use if he were entering the parameters on the system console. The format of each type of automated initialization member is described in the following paragraphs.

Member (or Card Deck) that Lists Members to be Used

The following format restrictions apply to the listing member referenced by the AUTO keyword and the card deck referenced by the RDR keyword. This member (or card deck) consists of a list of member names to be used to alter the default list of member names in NIP.

- Each record (or card) holds only one member name.
- A member name may start anywhere in the record (or card) as long as it is completed by column 71.
- A member name must be delimited by a blank.
- Comments may be placed after the delimiter.
- A record (or card) having its first 71 columns blank is ignored.
- The member names must be consistent with the naming conventions previously described. (The naming conventions permit you to list the member names in any order.)
- Adding "NULL" to the three required characters to form a member name (for example, SMFNULL) causes the corresponding entry in the default list to be made null (blanks).

The System Parameters (NIPxxxxx) Member

This member consists of entries made in the same format as if they were entered by an operator at the console. The following format restrictions apply to this type of member:

- The AUTO and RDR keywords are invalid entries.
- An entry may start anywhere in the record as long as it is completed by column 71.
- Continuation onto several cards is signalled by either
  the CONT keyword, or
  a comma followed by a blank.

The DEFINE Parameters (DFNxxxxx) Member

This member consists of the parameters of the DEFINE command in the same format as if they were entered by an operator at a console. The following format restrictions apply to this member:

- An entry may start anywhere in the record, but it must be completed by column 71.
- All records are read and processed in the order they are read in as long as they are syntactically correct.

To determine which parameters you want to place in the member, see the parameter descriptions in the response to message IEE802A in *OS/VS Message Library: VS1 System Messages.*

The Automatic Commands (CMDxxxxx) Member

This member consists of any system commands you want executed during system initialization; for example, START commands, LOGON commands, and VARY commands. The commands must be in the same format as if they were entered by an operator at the console. All commands contained in this member are displayed on the console unless NOLIST was specified in the AUTO or RDR keywords. The following format restrictions apply to this member:

- Only one command per record is allowed. No continuation is permitted.
- An entry may start anywhere in the record, but it must be completed by column 71.

The Permanently Resident Volume List Parameters (PRExxxxx) Member

For a description of this member, see the PRESRES description in the section *The PRESRES Volume Characteristics List.*

The SMF Parameters (SMFxxxxx) Member

The SMFxxxxx member controls SMF operations. This member consists of a series of parameters contained in 80-character card-image records. To determine which parameters you want to place in the member, see the description of SMFDEFLT parameters in *OS/VS System Management Facilities* (*SMF*), GC35-0004.

The JES Reconfiguration Parameters (JESxxxxx) Member

For a description of this member, see the JESPARMS description in the section *JES Reconfigurability.*

Unless it is necessary to override the values specified at sysgen, do not specify JESxxxxx (that is, let the null entry be generated). This allows the system to bypass all the I/O related to overriding the sysgen-specified values.

The SET Parameters (SETxxxxx) Member

This member consists of the parameters of the SET command in the same format as if they were entered by an operator at a console, with these exceptions:

- The unit address (unitaddr) in the Q= and PROC= keywords has been replaced with a volume identification (volid).
- A new keyword, QPARM=(jobqueue parameters), has been added. For a description of the jobqueue parameters, see message IEF423A in *OS/VS Message Library: VS1 System Messages,* GC38-1001.
- The DATE, CLOCK, and GMT operands cannot be used in this member. If DATE and CLOCK are specified, they must be in reply to message IEA101A or in a SET command issued after initialization. If GMT is specified, it must be in reply to message IEA101A.

The RTAM Parameters (RESxxxxx) Member

For a description of this member, see the discussion of the SYS1.PARMLIB member in the *RES System Programmer's Guide.*

Performing Automated System Initialization

During initialization, NIP generates the message

IEA101A SPECIFY SYSTEM AND/OR SET PARAMETERS FOR RELEASE xx.yy.sssss

where:

xx is the release number
yy is the release level
sssss is the system type

The operator's response to this message consists of a selection of keyword parameters. The formats and purposes of all parameters are described in *OS/VS Message Library: OS/VS1 System Messages,* GC38-1001.

Because of automated system initialization, the following parameters are included in the list of eligible keywords. Their purposes are briefly as follows:

Keyword *Purpose*

AUTO To specify a SYS1.PARMLIB member that lists other members to be used in automated system initialization. The NOLIST option of this keyword prevents the sending to any consoles non-critical informational messages issued by NIP or MSI.

RDR To specify a readied card reader holding a deck of cards that lists members of SYS1.PARMLIB to be used in automated system initialization. The NOLIST option of this keyword prevents sending non-critical informational messages, issued by NIP or MSI, to any consoles.

DATE To set the date in the TOD (time-of-day) clock.

CLOCK To set the time of day in the TOD clock.

GMT To express the time of day in Greenwich Mean Time.

Q To specify the DASD that holds the system job queue (SYS1.SYSJOBQE).

PROC To specify the DASD that holds the system procedure library (SYS1.PROCLIB).

SPOOL To override spool parameters.

DEVSTAT To specify device type(s) for NIP device status checking. Devices of the type(s) indicated are tested for a not-ready condition. The UCB for any checked device in a not-ready condition is marked as offline. Otherwise, each UCB is left marked as online.

Automated system initialization results if either AUTO or RDR is specified in the reply to message IEA101A. A normal (manual) initialization results otherwise.

In an automated initialization, the SET parameters DATE, CLOCK, and GMT should be specified in the reply to message IEA101A only if they must be changed. The other SET parameters (Q, PROC, and SPOOL) should be specified in the reply to message IEA101A if a SET parameters member is not used or if like parameters in the SET parameters member to be used must be overridden. Another opportunity for specifying SET parameters is not given unless an error occurs.

If NIP detects a parameter in the wrong format or an invalid parameter, NIP issues a message indicating the error. The operator must then respecify the parameter and any system or SET parameter that NIP has not yet processed. (NIP processes the parameters in the same sequence as they were entered by the operator and does not process any parameters after an error is encountered.)

In a manual initialization, the operator may specify the SET parameters Q, PROC, and SPOOL in either the reply to message IEA101A or the reply to message IEE114A, which occurs after NIP has completed its processing. If they are specified in the reply to message IEA101A, message IEE114A is not generated.

NIP processes each keyword in the reply to message IEA101A in the same sequence that the operator specifies them. Therefore, to override any keyword(s) specified in the system parameters member for a particular initialization, the keyword(s) must be specified after the AUTO or RDR keyword in the reply to message IEA101A.

The NOLIST subparameter of AUTO and RDR takes effect after the time is entered, so it cannot affect any entries made prior to the AUTO or RDR keyword entry. NOLIST prevents the generation of the IEE101A READY message and other non-critical messages.

Any SET parameters (Q, PROC, and SPOOL) passed from NIP to MSI override the corresponding keywords specified in the SET parameters member.

Note: The term "SET parameters" is used differently here than in other systems where such parameters were specified by an operator's use of the SET command. In VS1, the SET command is not related to Q, PROC, and SPOOL parameters.

Processing Notes

The List of SYS1.PARMLIB Members to be Used

The list of SYS1.PARMLIB member names referenced by the AUTO or RDR keyword contains the names of the members in the following table. If neither AUTO nor RDR is specified by the operator, the default list shown in the following table is used during initialization.

Members	Default List
System Parameters	Null Entry*
JES Reconfiguration Parameters	JESPARMS
DEFINE Parameters	Null Entry*
SET Parameters	Null Entry*
Permanently Resident Volume List Parameters	PRESRES
Automatic Commands	Null Entry*
SMF Parameters	SMFDEFLT
RTAM Parameters (if RES is included at SYSGEN)	RESPARMS
*A Null Entry consists of all blanks	

Use of the default list results in the manual initialization procedure, with the exception that the SET parameters can be specified in the reply to message IEA101A along with system parameters, or in the reply to message IEE114A, but not at both times. Processing of the list ends when:

- An end-of-file condition occurs on the card reader *or*
- The number of entries processed equals the number of entries in the default list.

If the list provided contains fewer member names than the default list, the remaining names are taken from the default list. The default list is thus altered by the names provided in the member. Once the default list has been altered, NIP processes its member (the system parameter member) for system parameters before continuing to process the operator's other replies to the message IEA101A.

If the system parameters member is null, no member is processed. Any change to the SYSGEN specifications must have been entered in response to the IEA101A message.

If a syntax error occurs, processing up to that point will have been completed, and the default list will have been altered up to that point. The record containing the syntax error is written to the console and control is returned to the operator.

The System Parameters (NIPxxxxx) Member

Processing of this member is the same as if the parameters had been entered by the operator at the console. An error in this member not only terminates the processing of this member, but also terminates processing of any keywords that followed the AUTO or RDR keyword entered by the operator.

The Define Parameters (DFNxxxxx) Member

Processing of this member is the same as if the parameters had been entered by the operator at the console. Error detection and recovery are also the same as if the entry was a reply by the operator.

The SET Parameters (SETxxxxx) Member

Processing of this member is the same as if the parameters had been entered by the operator at the console.

The Automatic Commands (CMDxxxxx) Member

Processing of this member is the same as if the parameters had been entered by the operator at the console. All commands are displayed on the console unless NOLIST was specified in the AUTO or RDR keyword.

OS/MFT—OS/VS1 Differences

This section contains a resume of some of the more significant differences between OS/MFT and OS/VS1. It is intended as a quick reference aid for the system programmer involved in a conversion from MFT to VS1. The differences appear in random order, and the sequence of their presentation in no way implies a level of importance or significance. Differences listed include enhancements, restrictions, and changes necessitated by the implementation of a virtual storage system.

OS/MFT – OS/VS1 Differences

1. JES readers and writers do not require a partition, but a partition must be available to start and stop them. Additionally:

   - The start reader and start writer commands do not require partition identifiers.

   - A "hot reader" capability is provided for unit record readers. (Unit record readers remain ready to process input, after being started, until they are stopped. It is not necessary to issue a reader START command after reloading the reader following an empty hopper condition.)

   - A WRITER command enhancement is provided to control writer output activity, such as the number of copies, forward space functions, and backspace functions.

   - The sequence of the writer output has been changed. The output now follows the sequence:

     JCL
     Messages } Messages and JCL are interspersed for multi-step jobs.
     Program output

2. The number of job classes that may be specified per partition has been increased from 3 to 15.

3. The interpreter function is no longer a subtask of the reader. Instead, the function occurs at job initiation time, as a subroutine to the initiator.

4. The contents of the job queue has been changed in VS1. A SWADS data set is now provided for each initiator, and this reduces the contention that exists for the job queue in MFT.

5. With the implementation of JES, a spool data set is provided. In addition to SYSIN and SYSOUT, JCL, messages, WTP messages, and system log data sets are included in the spool data set.

6. Programs compiled using PL/I F and using the teleprocessing facilities of this language translator cannot be run under VS1 because PL/I F uses QTAM as its teleprocessing access method. These programs can be recompiled using the PL/I checkout compiler or the PL/I optimizing compiler, both of which use TCAM as their teleprocessing access method.

7. Programs which previously required storage approaching 64K, 128K, 192K, etc. may require an additional 64K of virtual partition area to accommodate the system requirements (such as PQA area) of VS1.

8. MFT supports tape and disk for SMF output. VS1 supports disk only for SMF output.

9. Programs that modify active CCW strings require changes in order to execute in a virtual storage system.

10. Commands in the input stream between jobs are processed at reader time; those within the job, at interpreter time.

11. Users with I/O appendages must code a page-fix appendage in their programs to interface with IOS and fix the I/O appendages associated with the program.

12. Changes are required in all programs that declare or reference certain PSW fields directly, such as the system mask, interrupt code, condition code, program mask, ILC, and bit 12 of the PSW.

13. Programs using the SSK and ISK instructions will be affected because the operand 1 register now contains the "change and reference" bits as well as the storage key and fetch protect bit.

14. The SSM instruction will have degraded performance due to interrupt processing.

15. Programs that execute the LPSW instruction must be carefully checked because of the changes in the PSW format.

16. No EXCP support is provided for SYSIN/SYSOUT data sets.

17. DSCBS and user labels are not supported with SYSIN/SYSOUT data sets.

18. SMF data set compatibility is not supported because the format and content of SMF records has changed under VS1.

19. VS1 does not support:

    Main storage hierarchies (obviated by virtual storage concept)
    QTAM (superseded by TCAM)
    RJE (superseded by RES)
    IEBUPDAT (superseded by IEBUPDTE)
    TESTRAN (low usage component of MFT)
    GJP (low usage component of MFT)
    SGJP (low usage component of MFT)
    IBCRCVRP (superseded by IEHATLAS)
    HASP
    IMAPTFLE (replaced by HMAPTFLE)
    IMDMDMAP (replaced by HMBLIST)
    IHGUAP Utility (low usage component of MFT)

20. VS1 does not support these devices:

    IBM 1017
    1018
    1285
    2301
    2303
    2305-1
    2311
    2321
    2841
    7772

21. MFT provides three reader procedures with the system. VS1 provides two, RDR and RDRT. RDR400 and RDR3200 are not included in VS1. VS1 also includes two writer procedures, WTR and WTRT.

22. Partitions in VS1 have a minimum size of 64K and must be multiples of 64K in size.

23. The resident SVC, RAM, and BLDL default lists have been changed in VS1.

24. Resident options may be made non-pageable or pageable in VS1.

25. All problem program output is spooled to DASD by JES under VS1 and unit record UCBs are not created. Any user programs that handle conditions such as print overflow (channel 12) by checking bits in the UCB will not operate properly.

26. VS1 TCAM line groups can contain a maximum of 32 lines, except for the IBM 2260 (local attachment), for which there is no maximum.

27. RBs, etc. are not part of the problem program area in VS1. They have been moved to the PQA.

28. A PQA (one per partition) is provided in VS1. This does not exist in OS/MFT.

29. Programs that reference storage not obtained with GETMAIN or referenced after a FREEMAIN may not execute.

30. In MFT systems, SYSIN blocking factors are selected prior to job initiation, and BSAM users have to either write code accommodating the block size selected or explicitly provide JCL overrides. In VS1, the blocking done by the system is transparent to the user. For the BSAM interface, records are dynamically reblocked to user requirements during execution.

31. No DADSM facilities (OBTAIN, SCRATCH, RENAME) are supported for the SYSIN/SYSOUT data set in VS1.

32. SYSOUT data sets can only be written sequentially without repositioning. No support is provided for NOTE/POINT, for update, or for reading SYSOUT. Once written, SYSOUT records cannot be erased or overlaid.

33. In VS1, NOTE/POINT support for SYSIN is restricted as follows:
    a. The user must save four bytes of data (TTRL) instead of three as in MFT.
    b. POINT to following record (TTR01) is not supported.
    c. Track numbers (TT) do not begin with 0 and are not in ascending sequence.

34. No SYSIN support is provided for update or for overlaying or adding to the data.

35. SYSIN support for variable length records is not compatible. For MFT, block and record descriptor fields must be punched (in binary) in the input card image. In VS1, the entire 80-byte image is treated as data, and block and record descriptor fields are prefixed to it. Both blocked and unblocked formats are supported for SYSIN, but spanned records are not. (VS and VBS are supported for SYSOUT along with the other variable length formats.)

36. All SYSIN records processed by QSAM contain one logical record per original 80-byte card image. It is not possible to divide cards into multiple logical records or combine multiple cards into a single logical record except by using BSAM with user deblocking routines.

37. BSAM SVCS BSP and FEOV are not supported in VS1 and will result in an error return if issued.

38. SYNADF may only be issued from within a SYNAD exit in VS1.

39. SAM chained scheduling is supported for V=R jobs only. If a user IS not running V=R and specifies chained scheduling, regular scheduling will be substituted.

40. TCAM message control programs and TCAM message processing programs using the ICOPY, TCOPY, QCOPY, and TCHNG macro instructions must be reassembled and linkage edited. TCAM message processing programs not using any of these macro instructions only need to be re-linkage edited.

41. Additional JCL parameters are provided in the JOB and EXEC statements to facilitate running V=R jobs.

42. MFT reader procedures will not run in VS1 after release 1. Because the PARM field of the EXEC statement is changed, user reader procedures must be updated.

43. MFT stand-alone DASDI will not run in VS1 after release 1 because the IPL text has been changed.

44. In MFT (and in release 1 of VS1), reentrant user programs could modify themselves (for example, store registers into the user area). Such attempts will now cause a protection check.

VS1 Features and Options

This section contains a brief description of some of the features and options available in vs1. Although comprehensive coverage of all the features and options available is not provided, this section will serve as an aid to the planner responsible for determining if a specific optional feature or option should be included in (or excluded from) the system at system generation time. Features and options are arranged in the section in alphabetic sequence.

Section Outline

Continued —

Alternate Path Retry (APR)

Status: Optional. (The function is standard with the system but is effective only when the OPTCHAN= operand of the IODEVICE macro is coded.)

The alternate path retry (APR) option allows an I/O operation that has developed an error on one channel path to a device to be retried on another channel path to the same device. This can be done only if another channel path has been assigned to the device performing the I/O operation. APR also provides the capability to vary a path to a device online or offline by use of the VARY command.

APR can handle:

- Up to four paths to one device.
- The ninth drive on a 2314.

While it is not module-dependent, APR performs its function usefully only in a system that has the channel check handler (CCH) and alternate paths to at least some of the I/O devices. CCH checks for channel errors, analyzes the error, and produces an interface that aids in setting up an alternate path retry.

The operation of the selective retry function of APR, in conjunction with the I/O supervisor, does not depend on anything you do. The operator can initiate the VARY path function by entering the VARY PATH command in the input stream or at the console.

Attach Function

Status: Standard.

The ATTACH function, with subtasking, creates subtasks so that the issuing program and the program requested in the ATTACH macro instruction compete for system resources. The function allows more than one task to be operative within a partition.

Attach Function Made Resident

Status: Standard.

The routines that make up the ATTACH function are resident in storage in VS1. This function is optionally resident in OS MFT. Having the routines resident eliminates the necessity of bringing them into the supervisor transient area each time an ATTACH macro instruction is issued.

Automatic Partition Redefinition

Status: Standard.

The addition of the keyword PARM=membername to the DEFINE command allows the operator to redefine partitions more easily. Using this keyword, the operator can specify a SYS1.PARMLIB member containing partition redefinitions. This causes the system to go to that member to obtain the redefinitions. The operator is thus relieved from entering the redefinitions from the console.

The member consists of entries in the same format as if they were entered from the console. If END is not in the parameters, the redefinition can continue through operator replies. No restriction is made on where entries must start, but the last entry must be completed by column 71. Blank records are ignored. The parameters contained in the member are the same as those given in response to message IEE802A (see *OS/VS Message Library: VS1 System Messages,* GC38-1001).

Automatic Volume Recognition (AVR)

Staus: Optional. Included when VLMOUNT=AVR is specified in the SCHEDULR macro instruction.

This feature issues volume mounting instructions to the operator to minimize the time lost in performing job setups. The operator can premount labeled volumes on any available tape or disk device. The identification of the volume and the device used is automatically recorded in a table.

When a particular volume is needed for job setup, the table containing the volume information is searched. If the required volume is already mounted, the usual procedure of issuing a volume mounting message is bypassed. This feature is advantageous in installations where work schedules are normally set in advance and follow a repeated pattern.

Basic Direct Access Method (BDAM)

Status: Standard.

In the Basic Direct Access Method (BDAM), records within a data set are organized on direct access volumes in any manner chosen by the programmer. Storage and retrieval of a record is by actual or relative address within the data set. This address can be that of the desired record or a starting point within the data set where a search for the record, based on a key furnished by the programmer, begins. Addresses are also used by BDAM as a starting point for searching for available space for new records.

Basic Indexed Sequential Access Method (BISAM)

Status: Optional. Included when ACSMETH=ISAM is specified in the DATAMGT macro instruction.

Sequential and direct processing are provided by the Indexed Sequential Access Methods (ISAM). Records are maintained in control field sequence by key. The system maintains a multilevel index structure that allows retrieval of any record by its key. Additions can be made to an existing ISAM data set without rewriting the data set.

The Basic Indexed Sequential Access Method (BISAM) stores and retrieves records randomly from an indexed sequential data set. Selective reading is performed using the READ macro instruction, specifying the key of the logical record to be retrieved. Individual records can be replaced or new records can be added randomly.

Basic Partitioned Access Method (BPAM)

Status: Standard.

The Basic Partitioned Access Method (BPAM) is designed for efficient storage and retrieval of discrete sequences of data (members) belonging to the same data set on a direct access device. Each member of the data set has a simple name. The data set includes a directory that relates the member name with the address where the sequence begins. Members may be added to a partitioned data set as long as space is available in the directory and the data set.

Basic Sequential Access Method (BSAM)

Status: Standard.

In the Basic Sequential Access Method (BSAM), data is sequentially organized and physical blocks of data are stored or retrieved. The READ/WRITE macro instruction causes the initiation if an input/output operation. The completion of these operations is tested by using synchronization macro instructions. Automatic translation between EBCDIC and ASCII codes is provided for magnetic tape labels and record formats.

BLDL Table Made Non-Pageable

Status: Optional. Invoked by the specification of OPTIONS=BLDL in the CTRLPROG macro instruction.

The BLDL table is always resident in VS1. However, the user has options as to the entries he wishes to include in the table and as to whether he wants the table to be pageable or non-pageable. Having the table non-pageable eliminates the necessity of paging the table into real storage whenever the function is required. See the *Resident Routines Options* section of this publication for a comprehensive discussion of the function.

Channel Check Handler (CCH)

Status: Standard.

The Channel Check Handler (CCH) intercepts channel-check conditions, performs an analysis of the environment, and facilitates recovery from channel-check conditions by allowing for the scheduling of device-dependent error recovery procedures by the input/output supervisor, which will determine whether the failing channel operation can be retried.

Checkpoint Restart Facility

Status: Optional. The facility is standard in the system, but the user must code the RESIDNT=(ACSMETH) parameter in the CTRLPROG macro instruction to use it. You can also indicate the ABEND codes that you want to be eligible or ineligible for automatic restart, in the NOTELIG and ELIGBLE parameters of the CKPTREST macro instruction.

The checkpoint/restart facility expands the use of the restart capabilities that are provided by the RD parameter of the JOB and EXEC statements. The RD parameter permits execution of jobs to be restarted automatically at a job step after abnormal termination occurs.

Checkpoint/restart enables you to write checkpoint macro instructions (CHKPT) at various points in your program to record job status information. Then when an ABEND occurs, your program can be restarted automatically at the last of these points, or restart can be deferred until a later time, when the job can be resubmitted and the RESTART parameter in the JOB statement used. The RD parameter can also be used to suppress partially or totally the checkpoint/restart facility.

The following restrictions apply to the establishment of a checkpoint by the CHKPT macro instruction.

- When the checkpoint is established, the job step must comprise a single task. The job step task must be the only task when the job step is restarted.

- A checkpoint cannot be established by an exit routine that returns control to the control program.

- If a STIMER or WTOR macro instruction has been issued, a checkpoint cannot be established before the time interval is completed or the operator's reply is received.

Under certain conditions, such as the following, a checkpointed job may not be restarted at will:

- If a job took a checkpoint while running V=R, that job cannot be restarted while another partition is using part or all of the required V=R space. The job to be restarted must wait until the V=R space is available before restart is possible.
- If a job took a checkpoint while running V=R, and the system queue area (SQA) has expanded into the V=R space required to restart the job, restart is not possible until the system is re-IPLed and the V=R address space is again made available.

Consoles — Alternate and Composite Console Options

Status: Optional. Alternate consoles are optionally specified in the ALTCONS= parameter of the SCHEDULR macro instruction.

One primary console must always be specified for any vs1 operating system. One alternate console can be specified. A composite console, such as a card reader and a printer, can be specified as a primary or an alternate console. The composite console is considered one console even though it may consist of two different physical devices.

The following guidelines apply when Multiple Console Support (MCS) is not selected:

- A primary console must be specified in the SCHEDULR macro instruction.
- A composite console can be used as a primary or an alternate console.
- When a graphic device is going to be active as a console, a device that produces printed output must be specified.

Consoles — Multiple Consoles Support (MCS)

Status: Optional. Included by specification of the OPTIONS=MCS parameter of the SCHEDULR macro instruction.

You must specify the multiple console support (MCS) option to have two or more consoles active during execution. One console must be specified in the SCHEDULR macro instruction; it is called the "master" console. An alternate console for the master console must be specified in the ALTCONS parameter of the SCHEDULR macro instruction. A SECONSLE macro instruction must be coded defining the alternate as a secondary console. Additional secondary consoles can be defined with SECONSLE macro instructions—up to a maximum of 31 secondary consoles. For all consoles for which no alternate console is specified, the master console is automatically assigned as the alternate.

A hard copy log can be specified either at system generation or by the operator during system initialization or execution. A hard copy log is required when there is more than one active console during initialization or execution, or when there is an active graphic console. The hard copy log can be the system log that is contained on SYS1.SYSVLOGX and SYS1.SYSVLOGY, or it can be a console with output capability. If the log is required, the system records the operator commands, the system commands and responses, and the messages with routing codes of 1, 2, 3, 4, 7, 8, and 10 on the hard copy log. Additional messages can be recorded if desired.

Routing codes and descriptor codes are required for all messages handled by a system using MCS. Messages that already exist can be assigned routing codes at system generation time or, by default, they will be sent to the master console.

Routing codes are assigned to all new operator messages (WTO and WTOR). They designate what function the message is connected with and determine where a message will be sent. A system generation parameter provides the ability to supply routing codes to all operator messages that already exist and do not have routing codes.

Each console is assigned one or more routing codes. The routing codes assigned to a console are matched to the routing codes assigned to WTO and WTOR messages. If there is a match, the message is sent to the console. Some messages, such as a message that is broadcast to all active consoles, are not routed by the routing code.

Descriptor codes must be specified for all new operator messages. They are specified in the WTO or WTOR macro instructions. They designate how a message is to be printed or displayed.

All commands have been arranged by function into four command code groups: informational, system control, I/O control, and console control.

An exit, just before the routing codes of a message are checked, to enable you to supply your own routine to add, delete, or change routing and descriptor codes is provided. (See the *Message Routing Exit Routines* section of this publication for a description of the exit routine.)

The following guidelines must be used:

- If HARDCPY=SYSLOG is specified in the SCHEDULR macro instruction during system generation, then at IPL time the operator must change the HARDCPY parameter to refer to the address of an operator console that has output capability. The device should not be the master console. The HARDCPY specification can be changed back after the message IEE141I has been received.

- Master console must be specified in the CONSOLE keyword parameter of the SCHEDULR macro instruction.

- An alternate console to the master console must be specified in the ALTCONS keyword parameter of the SCHEDULR macro instruction.

- The alternate for the master console must be defined in the CONSOLE parameter of a SECONSLE macro instruction to make it a secondary console.

- A console with at least printing output capability must be specified as the hard copy log. Although the system log is not a console, and does not directly produce printed output, it can be used.

- A record of the operator commands, system commands and responses, and routing codes 1, 2, 3, 4, 7, 8, and 10 should be maintained.
- Up to 31 secondary consoles can be specified with SECONSLE macro instructions. They can all have alternate consoles specified. If no alternate is defined, then the master console automatically becomes the alternate.
- A 2250 display unit can be specified as a master, secondary, or alternate console.
- Any number of the consoles can be composite consoles.
- Routing and descriptor codes are assigned to all new operator messages that are written.

Conversational Remote Job Entry (CRJE) Facility

Status: Optional. Included by specification of the OPTIONS=CRJE parameter in the SCHEDULR macro instruction.

The conversational remote job entry (CRJE) facility provides remote access to the operating system from printer-keyboard terminals. Authorized terminal users can conversationally prepare and update programs and data, submit them for OS background processing, and receive the output either at the central installation or at the remote terminal.

CRJE is specified at system generation time in order to have the necessary modules included in the system. After generation, you must create the specific CRJE system required for your installation. There are three macro instructions available for this job—CRJELINE, CRJETABL, and CRJEUSER. You set up a job that includes the CRJE macro instructions necessary to specify your system; you may include your own routines. The assembler translates these macro instructions and creates the required modules. The linkage editor incorporates the modules into the operating system.

SYS1.MACLIB must be in the operating system so that the assembler can expand the macro instructions. SYS1.TELCMLIB must be in the system to hold some of the CRJE load modules as well as the telecommunication subroutines. Enough system queue space must be specified in the CTRLPROG macro instruction during system generation to handle the necessary CRJE space requirements.

DEB Validity Checking

Status: Standard. Reduced DEB validity checking is obtained by specifying OPTIONS=NODEBCHK in the CTRLPROG macro instruction.

DEB validity checking is designed to prevent a user's data set (associated with a given DEB) from being read or modified, either accidentally or intentionally, by another user program. In full DEB validity checking, system OPEN routines provide protected DEB entry creation and validation, IOS provides additional (and important) validation each time I/O is performed, and system CLOSE routines provide DEB entry validation and deletion.

Although some degree of data set security is achieved by the OPEN and CLOSE functions, it is substantially reduced without the IOS portion of DEB validity checking. Specification of OPTIONS=NODEBCHK removes IOS linkage to the DEB validity check module, thus limiting the overall effectiveness. In installations where data set security is a primary concern, full DEB validity checking should be allowed (that is, OPTIONS=NODEBCHK should not be specified).

Device Independent Display Operator Console Support (DIDOCS)

Status: Optional.

The device independent display operator console support (DIDOCS) provides uniform operator console support for a range of display devices. DIDOCS is included in your system when a display console and multiple console support (MCS) are specified.

DIDOCS provides the capability to:

- Print out messages from the vs1 control program and problem programs to the display console device.
- Enter commands from the display console to the control program by the alphanumeric keyboard and/or the light pen, when available.
- Print two out-of-line status displays as requested by the status display support.

Status display support provides for the presentation of information to a system operator clearly and understandably. It also provides the ability for messages to a display device to be displayed out of line in a special area of the screen. This allows related messages to be grouped together and easily read by the operator.

Direct Access Volume Serial Number Verification

Status: Optionally Excludable. The facility will be included in the system unless OPTIONS=NODAV is specified in the CTRLPROG macro instruction.

The direct access volume serial number verification facility checks or verifies the volume serial number of a volume after an unsolicited device-end interrupt condition has been corrected and the volume has been put back online.

When an unsolicited device-end interrupt 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).

Dynamic Device Reconfiguration (DDR)

Status: Optionally Excludable. The facility will be included in the system unless OPTIONS=NODDR or NODDRSYS is specified in the CTRLPROG macro instruction.

The dynamic device reconfiguration option allows a demountable volume to be moved from one device to another and repositioned if necessary without abnormally terminating the job or redoing IPL. A request to move a volume may be initiated by either the system or the operator.

The system transfers control to the DDR routines when a permanent I/O error occurs. These routines then determine whether another device of the same type to which the volume can be moved is available. When another device is available, the system requests a volume swap by issuing a message to the operator. The operator must answer this message by entering a SWAP command.

Notes:

- You should not code specific unit addresses in programs that will be processed on a system that has DDR.
- The direct access serial number verification routines must be in the system that has the DDR routines.

For FETCH: When I/O errors occur while the FETCH routines are addressing the SVCLIB, the DDR system residence routines receive control, and, if possible, request a swap. For this to occur, OPTIONS=DDRSYS must be specified in the CTRLPROG macro instruction and the conditions listed above must exist.

For DDR System Residence Routines: When these routines are specified in the OPTIONS keyword parameter of the CTRLPROG macro instruction, another keyword parameter, ALTSYS, must also be specified.

If high availability is important to the installation, a duplicate system residence volume is advisable. However, to use such a volume, writing on any part of the system residence volume other than SYS1.LOGREC would have to be prohibited.

The alternate residence device specified during system generation can be changed at IPL time by the operator.

For Nonstandard Labels: If you want DDR and have nonstandard magnetic tape labels, OPTIONS=DDRNSL must be specified. A nonstandard label routine with the name NSLREPOS must be supplied. This routine can either be added during system generation using the SVCLIB macro instruction, or be link-edited into SVCLIB after the system generation process is completed.

For DR when EXCP is Used: When the EXCP macro instruction is used to address magnetic tape drives in a program that will run under a system with DDR, REPOS=Y or N must be coded in the DCB macro instruction to indicate whether an accurate block count is being maintained.

Dynamic Dispatching

Status: Optional. Included in the system by specification of the DYNPART= and DYNINTR= parameters of the CTRLPROG macro instruction.

Dynamic dispatching is intended to prevent CPU-dominant tasks from monopolizing the CPU while I/O resources are idle and I/O dominant tasks are dispatchable. It is unlikely to aid environments in which most tasks are either CPU-dominant or I/O-dominant.

Dynamic dispatching provides for the alteration of dispatching priorities of selected tasks as they are being executed. It calculates the dispatching priorities so that tasks can, in some cases, use the system resources more efficiently. Dynamic dispatching not only alters the handling of each task as the task's characteristics change, but it also evaluates itself and alters itself based on its effectiveness in handling the tasks under its control.

Dynamic dispatching distinguishes between I/O-bound tasks and CPU-bound tasks. I/O-bound tasks receive the higher priority. Initially, all tasks are designated as I/O-bound. As each task is dispatched, its activity is monitored for a predetermined time interval. At the end of this time interval, each task is designated as either I/O-bound or CPU-bound.

Specifications at system generation time may be changed by the operator via response to message IEA101A, issued by the nucleus initialization program (NIP). For the sysgen parameters, see the *System Generation Reference* publication. For an explanation of the responses to message IEA101A, see the *VS1 System Messages* publication. The following shows a comparison of the sysgen parameters and the message responses with a brief explanation.

Sysgen Parameter	*Explanation*	*IEA101A Message Response*
DYNPART=(Pn–Pm)	Specifies contiguous partition(s) for which dynamic dispatching will be used.	DDG=(pn–pm) DDG=, may be used to cancel dynamic dispatching for the duration of this IPL.
DYNINTR=(a, b, c, d)		
	a delta value to be added to or subtracted from time-slice value at end of each statistics interval	DDDEL=nn
	b lower bound of time-slice that may be given to a task	DDMIN=nnn
	c ratio of CPU to I/O bound tasks	DDRATIO=nnn
	d length of statistics interval	DDSTAT=nnnnn

Note: Any partition that is part of the dynamic dispatching group must not be time sliced.

Dynamic Support System (DSS)

Status: Announced but not available. This information is for planning purposes only.

The dynamic support system (DSS) is an optional interactive debugging program that can be used by an IBM programming system representative or other authorized maintenance person to help identify and correct causes of programming failures. DSS requires the program event recording (PER) hardware feature of the System/370.

DSS has its own I/O capability and has access to both real and virtual storage. When DSS is executing, it is stand-alone and has control of the system. It can gain control from and return control to OS/VS through its monitoring functions and the integral operator's console (system restart is not necessary).

Because 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, such as a teleprocessing or time-slice task, is running. When DSS is in control, no other processing takes place unless DSS is being used only for monitoring. In that case, normal multiprogramming continues, reduced only by the processing of DSS.

The various features of the DSS language include:

- Displaying and altering real storage, virtual storage, and registers.
- Providing control for the program event recording (PER) hardware of the System/370.
- Stopping operation of the system at any instruction or PER interrupt, performing maintenance functions, and resuming operations.
- Saving information within DSS for later use or writing out the information on high-speed printers or on tape.
- Using tape or card readers for secondary command input buffers.
- Writing procedures that are often used for reiterative command sequences.

Though changes can be made by DSS, they are not permanent. Any modifications made to the system are not carried over to the next IPL. Also, DSS cannot modify itself, IPL, or NIP.

Extract Function Made Resident

Status: Standard.

This function is optionally resident in OS/MFT but is included as a standard function in OS/VS1. Having the function resident eliminates the necessity of bringing the routines into the supervisor transient area every time an EXTRACT macro instruction is issued.

The EXTRACT macro instruction provides your program with information contained in specified fields of the task control block (TCB) of either the task that issued the macro instruction or, in a multiprogramming environment, one of its subtasks.

Fetch Protect

Status: Optional. Included by specifying SECURITY=FPROT in the CTRLPROG macro instruction.

Fetch protect provides security for user data by preventing any user from examining the contents of another user's area of storage. This protection includes the entire dynamic storage area (virtual storage partitions assigned to job steps and system tasks) and all non-key 0 subpools. Partitions are initially fetch protected by the DEFINE command.

A combination of hardware and software support guards the non-key 0 contents of a partition from disclosure to any non-key 0 task operating in another partition. The PQA and SQA subpools and the nucleus are not fetch protected so that non-key 0 tasks can still reference these areas. When storage from the PQA is deallocated and returned to the partition, virtual storage management executes the SSK instruction, setting the fetch protect bit.

With a 2K block fetch protected, no task can access it unless the task's current PSW has key 0 or the 4-bit storage key for that block. Attempts to do so result in fetch protection program checks.

Graphic Programming Services (GSP, GAM)

Status: Optional. Included by specifying the appropriate parameters in the GRAPHICS macro instruction.

The graphic programming services control graphic input and output and a set of problem-oriented routines that are used as building blocks in the construction of graphic processing programs. The graphic subroutine package (GSP) allows the FORTRAN IV, COBOL F, or PL/I F programmer to use the graphic programming services.

The problem-oriented routines generate graphic instructions for displaying various images and alphameric information on the 2250 display unit. These routines function as part of the problem program and are reached by a CALL or LINK macro instruction.

Greenwich Mean Time (GMT)

Status: Standard. Utilized through use of the GMT parameter in the REPLY command.

The Greenwich Mean Time feature allows the user to maintain a time clock that is independent of local time. This is especially advantageous for teleprocessing operations that extend across time zones.

The TOD clock can be changed only at IPL time and requires that the GMT parameter be placed in the reply command. This parameter is optional, and if coded must give the date and time in Greenwich Mean Time. If the GMT parameter is not used, the system assumes that the date and time are local and does not alter the TOD clock.

If the GMT parameter is used, the local time and date are maintained by establishing an offset from Greenwich Mean Time. This offset is established at system generation time by using the TZ parameter of the CTRLPROG macro. The offset (local clock) can be changed during IPL in response to messages IEA101A and IEE055A (see *VS1 System Messages*) or after IPL by the SET command.

All system-issued time stamps are given in local time. To obtain GMT time stamps, a store clock (STCK) instruction must be used. For the format of the STCK instruction, see *System/370 Principles of Operation.*

Identify Function Made Resident

Status: Standard.

This function is optional under OS/MFT but is standard in vs1. Having the *identify* routines resident eliminates the necessity of bringing them into the supervisor transient area every time the IDENTIFY macro instruction is issued.

The IDENTIFY macro instruction is used to inform the supervisor of an embedded entry point within a load module.

After the IDENTIFY macro instruction has been executed, the entry point can be referred to by an ATTACH, LINK, XCTL, or LOAD macro instruction.

Indexed Sequential Access Method (ISAM)

The Indexed Sequential Access Method (ISAM) is comprised of the Basic Indexed Sequential Access Method (BISAM) and the Queued Indexed Sequential Access Method (QISAM). See the write-ups on these two access methods in this section.

I/O Load Balancing

Status: Optionally excludable. This facility will be included in the system unless OPTIONS=NOLOADBAL is specified in the SCHEDULR macro instruction.

I/O load balancing allocates data sets to devices in such a way as to attempt to equalize the amount of I/O contention on each device. This facility can be used only for non-specific requests (that is, where no volume serial number or device address is specified).

I/O load balancing attempts to select the best device for data set allocation by considering many variables. It accumulates information about the speed of the device, counts the number of I/O events to each device, and compares the characteristics of different devices in determining the best device to be allocated. I/O load balancing selects this best candidate for device allocation. If space is not available on that volume, the next best choice is used.

Job Step Timing

Status: Standard.

Job step timing is an optional feature under OS/MFT but is a standard feature in VS1.

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 option: 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.

Machine Check Handler (MCH)

Status: Standard.

This program processes machine-check interruptions. Depending upon the severity of the malfunction, the machine check handler:

- Restores the system to normal operation
- Terminates tasks associated with the malfunction so the system can resume processing, or
- Places the system in a wait state.

In all cases, the machine-check handler program writes diagnostic messages and error records.

Missing Interruption Checker (MIC)

Status: Standard. Program must be started by the operator (start mic.pn).

The MIC (missing interruption checker) is a program that polls active I/O operations to determine if a channel end and/or device end interruption has been pending for more than three minutes (default time). When this occurs, or when the system has issued a mount request and the request has not been satisfied within the time period, message IGF991E is issued. (See the *OS/VS System Messages* publication for an explanation of the message and for operator action.)

After message IGF991E is issued, the operator is given the specified time interval in which to respond. When that interval expires, the message is issued again. The action is repeated until the operator makes the necessary action.

The time interval can be changed through the use of a separate 8-byte CSECT (IGFINTVL) residing on SYS1.LINKLIB with IGFTMCHK. To change the interval, IGFINTVL must be replaced in the user's LINKLIB as follows:

```
//REPLACE          JOB        MSGLEVEL=1
//ASMLK            EXEC       ASMFCL,                                  X
//                          PARM.LKED='XREF,LET,LIST,NCAL,RENT'
//ASM.SYSIN        DD         *
IGFINTVL           CSECT
                   DC         CL8'00tt0000'
                   END
//LKED.SYSLMOD     DD        DSN=SYS1.LINK, DISP=OLD
//LKED.SYSIN       DD         *
                   INCLUDE SYSLMOD(IGFTMCHK)
                   ENTRY IGFTMCHK
                   NAME IGFTMCHK(R)
/*
```

Where: tt (two decimal characters) is the user-defined time interval. It expresses the interval in minutes to be used in checking for UCB conditions. Zero or non-numeric characters cause a default of three minutes.

Multiple Wait Option

Status: Standard.

The number of events that can be specified in a WAIT macro instruction can be extended from 1 to a maximum of 255. The WAIT macro instruction specifies that the task issuing the macro instruction should continue in control only after a particular event has occurred. An event could be the completion of an input or output operation or, in a multiprogramming system, the completion of another task.

On-Line Test Executive Program (OLTEP)

Status: Standard.

The On-Line Test Executive Program (OLTEP) is a function designed to direct the selection, loading, and execution of the On-Line Test sections (OLTS). OLTEP, with the OLTS, allows the testing of input/output devices used with the system concurrent with the running of customer jobs.

Concurrent debug with OLTEP is not supported in the first release of OS/VS1 for systems with 144K or less of real storage.

The OLTEP/OLT system is designed to:

- Diagnose I/O errors
- Verify I/O device repairs and engineering changes
- Exercise a device requiring dynamic adjustments
- Check the operation of I/O devices
- Verify the integrity of customer data

OLTEP operates as a job under OS/VS1 and is called by standard job control statements. It operates under control of the operating system at all times and uses the system facilities to accomplish the tests. It competes with other jobs in the system for the use of system facilities when running in a multiprogramming environment.

Definition of test runs can be entered by console or non-console devices. Prompting is available on consoles to assist in defining tests to be run.

IBM Field Engineering will supply the OLTS to the customer on magnetic tape or cards. The OLTS must be reformatted and link edited into a partitioned data set in order to be used under the operating system.

OLTEP must normally be executed in the non-pageable (virtual=real) area of real storage. It requires a minimum virtual partition of 64K and 36K bytes of real storage. The *logout analysis* program, which runs under OLTEP similar to an OLT, does not require virtual=real storage.

The initial release of OLTEP must run in the virtual=real storage area and is not supported on systems with less than 144K of real storage.

Program Controlled Interrupt (PCI)

Status: Optional. Included by specification of FETCH=PCI in the CTRLPROG macro instruction.

The program controlled interrupt (PCI) facility permits the program to cause an I/O interruption during execution of an I/O operation. PCI provides a means of alerting the program of the progress of chaining during an I/O operation.

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.

Queued Indexed Sequential Access Method (QISAM)

Status: Optional. Included when ACSMETH=ISAM is specified in the DATAMGT macro instruction.

Sequential and direct processing are provided by the Indexed Sequential Access Method (ISAM). Records are maintained in control field sequence by key. The system maintains a multilevel index structure that allows retrieval of any record by its key. Additions can be made to an existing ISAM data set without rewriting the data set.

The Queued Indexed Sequential Access Method (QISAM) is used to create an indexed sequential data set or to retrieve and update records sequentially from such a data set. Synchronization of the program with the completion of input/output transfer, and record blocking/deblocking are automatic. QISAM is also used to reorganize an existing data set.

Queued Sequential Access Method (QSAM)

Status: Standard.

In the Queued Sequential Access Method (QSAM), logical records are retrieved or stored as requested. The access method anticipates the need for records based on their sequential order, and normally will have the desired record in storage, ready for use, before the request for retrieval. When writing data, the program normally continues as if the record had been written immediately, although the access method routines may block it with other logical records and defer the actual writing until the output buffer has been filled. As with BSAM, automatic translation between EBCDIC and ASCII codes is provided for magnetic tape labels and record formats.

Reenterable Load Modules Made Resident
Resident Access Method Routines

Status: Standard.

Reenterable load modules from SYS1.LINKLIB and SYS1.SVCLIB and reenterable access method modules from SYS1.LINKLIB are resident under VS1. Having these modules resident eliminates the necessity of bringing them into storage whenever they are required.

Standard lists are used during IPL to indicate the load modules that are to be made resident. See the *Resident Routines Options* section for a complete discussion of this function.

Shared DASD

Status: Optional. Included by specification of FEATURE=SHARED in the IODEVICE macro instruction.

Up to four central processing units can access the same direct access device concurrently, depending upon the device configuration.

See the *Shared Direct Access Device Option* section for a complete discussion of this function.

SPIE Routines Made Resident

Status: Standard.

The routines that make up the Set Program Interruption Element (SPIE) function are resident in storage in VS1. This function is optionally resident in OS/MFT. Having the function resident eliminates the necessity of bringing it into storage whenever the SPIE macro instruction is issued.

The SPIE macro instruction specifies the address of a routine to be used when specified program interruptions occur in the task that issued the macro instruction.

Storage Protection Option

Status: Standard.

This is an optional feature under OS/MFT but is standard in VS1.

Storage protection keys are assigned to 2K areas of storage that are designated for use by either the system (storage protection key of 0) or problem programs (storage protection keys of 1-15). This feature prohibits the modification, by a problem program, of areas of storage other than those identified with the problem program's storage protection key. The system has access to all allocated storage protection keys and may, on occasion, use non-key 0 areas.

System Management Facilities (SMF)

Status: Optional. Included (or excluded) by specification of the appropriate parameter in the SMF= operand of the SCHEDULR macro instruction.

The System Management Facilities (SMF) collect and optionally record system, job management, and data management information. They also provide control program exits to installation-supplied routines that can periodically monitor the operation of a job or a job step.

SMF collect such information as:

System configuration
Job and job step identification
CPU wait time
CPU time used by each job and job step
Virtual or real storage requested by each job step
Virtual or real storage used by each job step
Paging statistics on a job step and system basis
I/O device use by each job step
Temporary and non-temporary data set use by each job and job step
Temporary and non-temporary data set status
Status of removable direct access volumes
Input count by each job and job step
Output count by each job
Output writer records by each job
Allocation recovery records by each job
VARY ONLINE and OFFLINE records

It is possible to suppress the writing of all, or of selected, SMF records at IPL time.

The SMF exits to installation-written routines allow certain parameters to be passed to them to identify the job and job step being processed and to provide accounting and operating information. These exit routines can cancel jobs, write records to the SMF data set, open and use their own data sets, and suppress the writing of certain SMF records.

Telecommunications Option

Status: Optional. Included by specification of BTAM and/or TCAM in the DATAMGT macro instruction.

The telecommunications option is comprised of two access methods, the Basic Telecommunications Access Method (BTAM) and the Telecommunications Access Method (TCAM).

BTAM provides the basic facilities required to process a telecommunications program. These include facilities for creating terminal lists and for performing the following operations:

Initiating and answering calls to and from terminals on switched networks
Polling and addressing terminals on non-switched multi-point lines
Changing the status of terminal lists
Transmitting and receiving messages
Code translation
Retransmitting messages which are received with detected errors
Providing on-line terminal test facilities
Keeping error statistics

BTAM supports binary synchronous communications on a variety of low, medium, and high speed start/stop devices.

BTAM supports binary synchronous communications over non-switched (leased or private direct connection) and switched (dial) networks in a System/370 to terminal communication.

Optional communication serviceability facilities are available in BTAM. They include error recovery procedures, diagnostic error information, error counts, and on-line terminal tests. It is recommended that these facilities be included, since they increase system availability.

OS/VS1 BTAM supports the same functions as OS BTAM and requires no additional programmer training. The user is *cautioned* concerning internal changes he may have made in OS BTAM. Similar changes will be required in VS1 BTAM.

The Telecommunications Access Method (TCAM) is a general purpose teleprocessing support program. It provides:

- A regionalized, general-purpose teleprocessing access method with facilities that permit exchange of data between a central System/370 and remote terminals.
- A control program designed to optimize the allocation and scheduling of a computer's resources in a real-time teleprocessing environment.
- A high-level language composed of macro instructions designed specifically to facilitate the construction of a teleprocessing network control program.

TCAM provides unified management of terminal devices, local and remote, including binary synchronous communication devices, through a single message control program. The TCAM application program interface has been defined to provide maximum compatibility with BSAM (READ/WRITE level) and QSAM (GET/PUT level), yet provide the ability to identify or specify source and destination of terminal I/O. Network control functions may be provided in an application program able to issue TCAM operator control commands.

Teleprocessing applications using TCAM are constructed by providing a message control program and one or more TCAM application programs.

The TCAM message control program serves as an interface between remote terminals, user-written application programs, and secondary storage devices on which messages are queued until their destinations are available to receive them. The message control program controls the flow of messages to and from the terminals, application programs, and queuing devices in a manner that optimizes allocation and scheduling of the computer's resources.

TCAM permits the user to code one or more application programs and interface these with the message control program. Application programmers are insulated from the teleprocessing environment. They issue ordinary GETS and PUTS or READS and WRITES to move data between the message control program and application program work areas.

TCAM application programs can be SAM compatible, and may be debugged in a non-teleprocessing environment using BSAM or QSAM as the access method, with a tape, card reader, disk, card punch, printer, etc. as I/O devices. Once debugged, many application programs can be plugged into TCAM without reassembly by changing a single job-control statement. The user can specify that either messages or user-defined records be transferred when he issues his GET/READ or PUT/WRITE macros.

TCAM offers an extensive set of service facilities including:

- A set of operator commands that allow the user to determine the status of his teleprocessing system and alter, activate, or deactivate portions of that system by entering appropriate commands from the system console, remote terminals, or application programs.
- A checkpoint/restart facility that allows the user to specify that his message control program environment be restored following system failure or closedown.
- A facility for selectively logging incoming or outgoing messages or message segments.
- Comprehensive debugging aids, including error-recovery and event-recording facilities, and utilities that permit debugging information to be dumped to tape or disk and then printed out.
- An on-line test facility that allows the user to test transmission control units and remote terminals without closing down the message control program or deallocating the device being tested.

OS TCAM message control programs must be reassembled to run in the OS/VS1 environment. This reassembly allows the message control programs to benefit from the virtual storage capability of VS1. Under VS1, TCAM runs as a subsystem in a virtual partition. Certain TCAM elements, such as the buffer pool, I/O appendages, control blocks, and tables are fixed in real storage for the duration of the TCAM task.

Time-Slicing Facility

Status: Optional. Invoked by specification of the appropriate parameters in the TMSLICE= operand of the CTRLPROG macro instruction.

When the time-slicing facility is included in the system, you can establish a group of partitions or tasks (called a time-slice group) that are to share the use of the CPU, each for the same fixed interval of time. This is done for jobs scheduled into a group of consecutive partitions that have been defined as the partitions to be used for time slicing.

The priority of a job can be changed by the CHAP macro instruction so that its priority will fall within the range of the priorities for the partitions defined for time slicing. This job will then be handled in the same manner as the other jobs in the time slice group.

When a member of the time-slice group has been active for the fixed interval 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 length of time. In this way, all member tasks are given an equal slice of CPU time, and no task or partition within the group can monopolize the CPU.

Only partitions that are assigned to the time-slice group will be time-sliced, and they are time-sliced only when the first partition in the group is the highest priority ready task. Dispatching of the partitions continues within the group until all the partitions are in a waiting state, or until a partition with a higher priority is in a ready state.

The group of tasks to be time-sliced (selected by priority or partition range) and the length of the time slice are specified at system generation time in the CTRLPROG macro instruction. This can be modified in vs1 through the DEFINE command. Any task or partition in the system that is not defined within the time-slice group is dispatched under the current priority structure; that is, the task or partition is dispatched only when it is the highest priority ready task or partition on the TCB queue. The maximum number of milliseconds, a number specified from a range of 20 to 9999, is the amount of time that each ready task is to have control of the CPU during one pass through the group.

Trace Option

Status: Optional. Included when the TRACE= parameter is specified in the CTRLPROG macro instruction.

A tracing routine aids in debugging and maintenance of the system.

The tracing routine stores information pertaining to start I/O (SIO) instruction execution, supervisor (SVC) interruptions, external interruptions, program check interruptions, and I/O interruptions in the trace table. When the table has been completely filled, the succeeding entries overlay the existing ones.

During system generation, only the size of the table is specified. However, when this system generation parameter is specified, the trace program routines are also included as part of the control program.

Transient SVC Table

Status: Optional. Included when OPTIONS=TRSVCTBL is specified in the CTRLPROG macro instruction.

The relative track address (TTR) of all transient supervisor (SVC) routines are included as part of the resident table of control program SVC routines. (See the description in *Types 3 and 4 SVC Routines Made Resident* in this section.)

Types 3 and 4 SVC Routines Made Resident

Status: Optional. Included when RESIDENT=TRSVC is specified in the CTRLPROG macro instruction.

Modules of types 3 and 4 supervisor (SVC) routines can be made permanently resident in storage.

Types 3 and 4 SVC modules are loaded and made resident at IPL time. When this option is specified, the transient SVC table option is assumed. The SVC table is a table containing the relative track addresses of all transient SVCs. This table is also stored in the resident portion of the control program.

The names and sizes of the types 3 and 4 SVC routine modules are given in the *OS/VS1 Storage Estimates* publication. (See also the preceding description *Transient SVC Table* and the *Resident Routines Options* section of this publication.)

User Modify Logical Cylinder Facility

Status: Optional. Included by specification of the ALCUNIT parameter in the JESPARMS member of SYS1.PARMLIB.

The user modify logical cylinder facility allows you to define the unit of allocation for spooling. A default value for logical cylinder definitions set at system generation time allows for approximately 28K of DASD work space per allocation. These values are adequate for an installation whose spool data sets (JCL, SYSIN, SYSOUT, etc) vary in size. If your installation consistently has jobs with small spool data sets and uses less than 28K, DASD work space is wasted. This facility allows you to specify a smaller unit of allocation, increasing spool availability. If your installation consistently has jobs with large spool data sets (using more than 28K), the default logical cylinder definitions could cause extra allocation processing. Defining a larger unit decreases the number of spool allocation calls.

To modify the logical cylinder definitions, specify the unit of allocation in bytes via the ALCUNIT parameter of the JESPARMS member of SYS1.PARMLIB. The system converts the byte value to a value in tracks for each spool device type. The default value, 28,672 bytes, allows for the following logical cylinder definitions:

- For 2314 or 2319, a logical cylinder is 5 tracks.
- For 3330 or 2305-2, a logical cylinder is 3 tracks.

The formula used by the system to convert the byte-cylinder definition to tracks is:

$$\frac{\text{ALCUNIT}}{\text{BUFSIZE} * \textit{buffers per track}}$$

where

ALCUNIT is the specification of the spool allocation unit in bytes via the ALCUNIT parameter in the JESPARMS member of SYS1.PARMLIB.

BUFSIZE is the spool buffer size as specified in the BUFSIZE parameter in the JES SYSGEN macro or the JESPARMS member of SYS1.PARMLIB.

buffers per track is the buffer-per-track count for the volume.

ALCUNIT must be large enough so that the logical cylinder will contain at least one logical cylinder map (see the Master Cylinder Map size formula in the section on SYS1.SYSPOOL in the *System Generation Reference* manual). It must also be small enough that, when it is converted to tracks, it is less than 256 tracks.

The following is an example of the JCL and control cards that might be used to change ALCUNIT in the JESPARMS member of SYS1.PARMLIB to 5,000 bytes:

```
//PARMSC JOB MSGLEVEL=1
//SG1 EXEC PGM=IEBUPDTE,PARM=NEW,COND=(4,LT)
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD DISP=OLD,DSN=SYS1.PARMLIB,UNIT=2314,VOL=SER=AOS102
//SYSIN DD DATA
./  ADD NAME=JESPARMS,LEVEL=01,LIST=ALL,SEQFLD=738
./  NUMBER NEW1=1,INCR=5
         BUFSIZE=436,NUMBUFS=7,STEPWTP=15,                      00000001
         SWDSLMT=15,SPOLCAP=80,WTLRCDS200,                      00000006
         RDR=(R=1,Y=5,B=0),JOUTLIM=5000,                        00000011
         WTR=(W=1,U=5,Z=10,B=132),                              00000016
         ALCUNIT=5000,                                          00000021
         SPOLVOL=(SYSRSM)                                       00000026
         ↑                                                      ↑
       col 10                                                 col 73
```

Note: When ALCUNIT has been varied, system restart cannot be performed.

User-Added SVC Routines

Status: Optional. Included by specification of the appropriate parameters in the SVCLIB macro instruction.

User-written supervisor (SVC) routines can be added to the control program.

All SVC routines, whether they are to be transient or resident, must be listed in the operand of the SVCTABLE system generation macro instruction.

Any resident SVC routines that are to be added must be specified in the system generation RESMODS macro instruction. The fixed storage requirement is increased by the total of the sizes of the routines that are going to be added plus the size of the control information.

Any transient svc routines that are to be added must be specified in the svclib system generation macro instruction in the operand. In this case, only the size of the control information is added to the fixed storage requirements.

Nonstandard error routines can be one of the types of routines that are added. User-written routines must have a value from 220 to 229. This value is the suffix of the name IGE00 by which the error routine is named in sys1.svclib.

See the section *Adding SVC Routines to the Control Program* in this publication for a complete discussion of this feature.

Validity Check Option

Status: Standard.

This is an optional feature under os/mft, but is standard in vs1. Extra validity checking is included in the system to determine whether addresses are located within proper boundaries. The validity checking is provided for the WAIT, POST, and GETMAIN/FREEMAIN modules. The checking for WAIT also checks for the number of events.

Virtual Storage Access Method (VSAM)

Status: Announced, but not available.

The Virtual Storage Access Method, vsam, is announced but not available. vsam is an access method for use with direct access storage devices on IBM System/370 with vs. vsam creates and maintains two types of data sets. One 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 is sequenced by the time of arrival of each record into the data set and is called an *event-sequenced data set.* Data records are located by using the records displacement from the beginning of the data set. The displacement is called the relative byte address (RBA). The RBA is 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. vsam stores data records in a new format designed for long term data stability and for data base applications. Data in both types of data sets can be accessed either sequentially or directly.

vsam enhances many ISAM capabilities including device independence, concurrent processing, data portability, and kinds of accessing supported. It provides additional password security protection. vsam creates and maintains separate catalogs that contain specilized information about each vsam data set and are used to link a data set to its index. vsam includes a multifunction utility program that defines, deletes, prints, copies, and provides backup and portability of vsam data sets and maintains the separate catalogs. An interface routine to allow most ISAM programs access to vsam data sets is also provided. For a more detailed explanation of vsam, see the *OS/VS Virtual Storage Access Method (VSAM) Planning Guide,* GC26-3799.

Volume Statistics Facility

Status: Optional. Included when SMF=FULL is specified in the SCHEDULR macro instruction with appropriate operands for the ESV= and EVA= parameters.

The volume statistics facility is used only for magnetic tape volumes with or without labels. It provides two functions, either or both of which can be specified at system generation time in the SCHEDULR macro instruction.

One function is error statistics by volume (ESV). It is intended primarily to be used with labeled volumes, but will handle an unlabeled volume if the serial number is given to the operating system. Statistics about the number of read or write errors and the system and unit on which the volume is located are recorded.

The other function is error volume analysis (EVA). It is intended primarily to be used for unlabeled or nonstandard labeled volumes. It monitors the number of read or write errors based on the limits you provide at system generation time.

The error statistics by volume (ESV) collects a set of statistics for each labeled tape volume whenever the volume is open. An unlabeled tape volume can be handled if the serial number has been supplied to the operating system.

If ESV=SMF is specified at system generation time, the statistics are accumulated on the system management facility (SMF) data sets SYS1.MANX and SYS1.MANY.

If ESV=CON is specified or if ESV is not coded, an abridged version of the statistics is printed on the console. This occurs at end-of-volume or when the tape volume is closed.

You can provide your own recording routine. ESV=CON must be specified, or the keyword parameter can be omitted because the default is CON. The UCBS, in the proper format, are constructed at system generation time. You can provide your own access method, using SVC 91, specify your own record format, and select your own recording data set. If you use the record 21 format instead of your own, you can use the IFHSTATR utility to print out the statistics.

The error volume analysis (EVA) acts as a monitor about the number of read and write errors for unlabeled or nonstandard labeled tape volumes. You provide the maximum limits for read errors and/or write errors and, if the maximum is reached or exceeded, a message, IEA620I, is printed on the console.

JES Reconfigurability

This section explains how to temporarily modify, or reconfigure, the Job Entry Subsystem (JES) parameters which were specified for your system at sysgen time. The temporary modification occurs at IPL time and will be effective until the next IPL.

Section Outline

JES Reconfigurability

JES reconfigurability permits you to make temporary modifications, at IPL time, to the JES parameters which were specified at sysgen time. To make permanent updates to the system JES values, you must do a new sysgen.

JESPARMS Member in SYS1.PARMLIB

The JESPARMS member in SYS1.PARMLIB contains the JES options or default values that were specified at sysgen. This member also provides an example for you to use in re-specifying any or all of the JES parameters. The parameters in JESPARMS are checked at IPL time by a master scheduler initialization routine. If any errors are detected during the check, a message is written on the console, indicating the error and the JES values specified at sysgen are used by the system. If no errors are detected, the JES values specified in JESPARMS will temporarily override those specified at sysgen. If the JESPARMS member is deleted, the JES values specified at sysgen are used.

The sysgen process uses the IEBUPDTE utility program to place JESPARMS in SYS1.PARMLIB. This utility is also used to maintain and update JESPARMS.

JESPARMS Entries

Each JESPARMS entry must be an 80-byte record using positions 1 through 71 to contain the JES parameters, and the optional information field. JES parameters can start in any position in the record, but they must be completed, with the exception of the SPOLVOL parameter, on the same record. All parameters must be separated by a comma. If an optional information field is present in the record, it must be separated from the parameter(s) by at least one blank position following the comma.

During the check at IPL time, the parameters are scanned until:

1. An error condition is found
2. A parameter not followed by a comma is found
3. The end of the data set is reached.

If either of the first two conditions occur, the remaining parameters are not checked.

Two examples of routines to modify the JES values on a temporary basis follow. For a complete description of the JES parameters, see the *VS1 SYSGEN* publication.

```
//STEP1       EXEC   PGM=IEBUPDTE,PARM=MOD
//SYSPRINT    DD     SYSOUT=A
//SYSUT1      DD     DSNAME=SYS1.PARMLIB,VOL=SER=VOLID1,UNIT=2314,DISP=OLD
//SYSUT2      DD     DSNAME=SYS1.PARMLIB,VOL=SER=VOLID1,UNIT=2314,DISP=OLD
//SYSIN       DD     DATA
./ REPL       REPL   NAME=JESPARMS,LEVEL=01,LIST=ALL,SEQFLD=738
./ NUMBER            NEW1=1,INCR=5
              BUFSIZE=600,     **JES PARAMETER OPTIONS**
              NUMBUFS=15,
              STEPWTP=20,
              SWDSLMT=30,
              WTLRCDS=200,
              JOUTLIM=5000,
              RDR=(R=2,Y=5,B=9999),
              WTR=(W=1,U=0,Z=6,B=9999),
              SPOLCAP=20,
              SPOLVOL=(000000,111111,222222,333333,444444,555555,666666,777777,
              888888,999999)
/*
```

Note that the SPOLVOL parameter is the only parameter that can span records. All other parameters must be started and completed on the same record.

```
//STEP1       EXEC   PGM=IEBUPDTE,PARM=MOD
//SYSPRINT    DD     SYSOUT=A
//SYSUT1      DD     DSNAME=SYS1.PARMLIB,VOL=SER=VOLID1,UNIT=2314,DISP=OLD
//SYSUT2      DD     DSNAME=SYS1.PARMLIB,VOL=SER=VOLID1,UNIT=2314,DISP=OLD
//SYSIN       DD     DATA
./ REPL       REPL   NAME=JESPARMS,LEVEL=01,LIST=ALL,SEQFLD=738
./ NUMBER            NEW1=1,INCR=5
              BUFSIZE=600,NUMBUFS=15,WTLRCDS=200,
              JOUTLIM=5000,STEPWTP=20,SPOLCAP=20,
              SWDSLMT=30,SPOLVOL=(000000,111111,222222,444444,555555),
                     **CONTINUATION OF SPOLVOL PACK**
              RDR=(R=2,Y=5,B=9999),
              WTR=(W=1,U=0,Z=6,B=9999)
/*
```

Note that some parameters are combined on one record and that optional information can be placed in the positions following the parameter and at least one blank.

Job Queue Format

The job queue format is specified when the system is generated and may be altered during subsequent system start procedures. Formatting consists of specifying the number of queue records in a job queue logical track, specifying the number of queue records to be reserved for each initiator, and reserving queue records needed to start at least one initiator and one writer.

This section provides guidelines for estimating:

- The number of queue records in a job queue logical track.
- The number of queue records to be reserved for use by an initiator.
- The number of queue records to be reserved to start at least one initiator and one writer.

Section Outline

VS1 Job Queue Formatting

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 some of the tables and control blocks developed by the reader, interpreter, and initiator control program routines.

Lack of queue records is not critical for a reader routine. Processing of a reader 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. Because one or more readers 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 operation. The main function of job queue formatting is to reserve queue records for initiator use.

To format the job queue, you must:

1. Designate the address of the device on which SYS1.SYSJOBQE resides, and indicate that it is to be formatted.
2. Designate:
   a. 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. Queue records are assigned in terms of logical tracks.
   b. The number of queue records to be reserved for use by an initiator. Each initiator is allocated this number of records.
   c. The number of queue records to be reserved to start a writer and an initiator if queue space becomes critical.

The balance of the queue (total queue records less the reservations in items 2b and 2c) is the maximum available for use by the reader(s).

Specify initial values for items 2a, 2b, and 2c in the SCHEDULR macro instruction parameters JOBQFMT, JOBQLMT, and JOBQTMT, respectively. The *System Generation* publication describes the procedure.

The service aids program IMCJOBQD 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. For a complete description of IMCJOBQD see the appropriate *Service Aids* publication.

No comprehensive, foolproof formulas exist for calculating values of JOBQFMT, JOBQLMT, and JOBQTMT. The values to be estimated depend on the requirements and structure of jobs to be presented to the system. The rest of this section provides some basic guidelines for your use in determining these values.

Logical Track Size—JOBQFMT

This specification affects the efficient use of queue records. Queue space is allocated in terms of logical tracks. Unused records in a logical track are not available for use for other jobs. Therefore, an overly large logical track size results in wasted queue records and the reduction of job queue capacity. Conversely, a small logical track size can result in decreased performance if most jobs require frequent assigning of a new logical track.

The distinction needed here is between problem program and system tasks. In the context of a discussion of the queue space used, all generalized start jobs are considered system tasks. The job queue space required for a problem program is quite small, whereas the space needed for a system task is considerably larger. Thus, the factor to consider in picking a value for this parameter is the number of system tasks run, as compared to the number of problem programs.

This trade-off is best summarized as follows:

- The larger the logical track size, the less frequently new tracks are allocated in handling system tasks, but space will be wasted in handling problem programs. The smaller the logical track size, the more efficiently job queue space will be used, but the overhead in processing system tasks will be increased.
- The range of possible values is from 5 to 255. The default value is 5. This minimum size is sufficient to contain the entire input queue entry for a problem program. This size also allows up to five data sets per SYSOUT class other than the message class and four data sets in the message class without requiring the allocation of more than the minimum number of tracks needed for any job. This minimum is x+1 tracks, where x equals the number of SYSOUT classes used by the job. The default of five, therefore, results in maximum space usage of the queue. A larger size would only be needed if the generalized start function is used frequenty, and the installation wants to decrease the number of allocations needed for the system tasks on the queue.

You may, as a starting point, wish to use the default value for JOBQFMT (five queue records).

Reserving Initiator Queue Records—JOBQLMT

The value specified for this parameter must take into account the possibility of tables being written to the job queue by components other than the interpreter. During normal job execution, this value should be equal to SWDSLMT (in the JES sysgen macro) plus the product of the logical track size (JOBQFMT) times the average number of SYSOUT classes used by a job. However, if the user intends to use automatic restart in the system, this number must be substantially increased. This is because of the conditions under which restart functions. The job to be restarted must be reinterpreted. Therefore, a set of interpreter records is needed. Also, the restart reader needs ten records for its own use. Finally, if automatic checkpoint/restart is being used, some restart housekeeping records are needed. These requirements can be summarized by the following formula:

JOBQLMT = L + S + 10A + 12A

where:

L = value of JOBQLMT if automatic restart is not used.
S = size of SWADS, in records.
A = number of times a job may be automatically restarted.
10 = step restart housekeeping records (needed also for checkpoint).
12 = Checkpoint/restart housekeeping records (needed also for checkpoint in addition to the ten needed for step restart).

If jobs with automatic restart may be held for operator restart, the JOBQLMT requirement is increased even further, because the system must maintain both the queue records and the housekeeping records for the held jobs until they have completed processing and are written. The formula then becomes:

JOBQLMT = (H + 1) (L + S + 10A + 12A)
where:
H = number of jobs that may be held at any one time.
Other terms are as previously described.

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 initiator is added to the total number of records reserved for active initiators. For example, if four initiators have been started and the number of records reserved for each initiator is 25, the number of records reserved for starting a writer and initiator if queue space becomes critical is 80, then the total number of records reserved is 205. This total includes 25 records reserved for each initiator, 80 records reserved for starting a writer and an initiator if queue space becomes critical, and 25 records reserved as a basic threshold.

Reserving Records to Start a Writer and an Initiator—JOBQTMT

When the reserves of queue space are becoming critical, space must be available to get system tasks started (especially the writer since it returns queue records to the system). Thus, to ensure availability of this space, the value of JOBQTMT should be the number of records needed to start the writer and the initiator. The amount needed to start each of these tasks may be estimated by following the formula for SWADS size (see the *OS/VS1 Storage Estimates* publication) taking into consideration that the JCL from the START command and the cataloged procedure must be interpreted. The queue space needed for input and output queue entries is automatically added by the queue manager.

Message Routing Exit Routines

This section provides detailed information on how to write user exit routines that modify the routing and descriptor codes of WTO or WTOR messages for any OS/VS operating system that has the Multiple Console Support Option (MCS). 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.

Documentation of the internal logic of the supervisor and its relationship to the remainder of the control program can be obtained through your IBM Branch Office.

Section Outline

Characteristics of MCS

Multiple Console Support (MCS) is an option of VS1 that 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 16 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. (For descriptions and definitions of the routing codes see the *Routing and Descriptor Codes* publication.)

If the standard routing codes provided on application and system messages do not cover special situations at your installation, the routing codes used on the message can be modified by coding a user exit routine. The exit routine receives control prior to the routing of messages so that you can examine the message text and modify the message's routing and descriptor codes. The system uses your modified routing codes to route the message. Descriptor codes provide a mechanism for message presentation and deletion and are explained later in this section.

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 option is provided to record messages, operator and system commands, and operator and system responses to commands. The hard copy log may be a console device or it may be the system log (SYSLOG). The number and type of messages recorded on the log is also optional. Your 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.

Information about RES support for WTO and WTOR appears in the *System Macro Instructions* (*SMI*) section of this publication.

Writing a WTO/WTOR Exit Routine

You write a WTO/WTOR exit routine to modify the standard routing codes and descriptor codes. 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, your routine receives control prior to the routing of the message. When your 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 4-byte routing code field and a 4-byte descriptor code field. Your exit routine may examine but not modify the message text.

A message is 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 is sent to the master console by the operating system.

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 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	

Figure MSG 1. Programming Conventions for WTO/WTOR Exit Routines

Programming Conventions for WTO/WTOR Exit Routines

The programming conventions for the WTO/WTOR exit routine are summarized in Figure MSG 1. Details about many of the conventions are in the reference notes that follow that figure. The notes are referred to by the numbers in the last column of the figure.

Reference Code *Reference Notes*

1 If your exit routine is to be reenterable, you cannot use macro instructions whose expansions store information into an inline parameter list.

2 You should write your exit routine so that program interruptions cannot occur. If a program interruption occurs during execution of the exit routine, 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 Text (128 Characters – padded with blanks)
UCMROUTC	Routing codes (4 bytes)
UCMDESCD	Descriptor codes (4 bytes)

DSECT IEECUCM is contained in SYS1.AMODGEN

System messages have a message code as the first seven characters of the message text. This code may be examined to aid in identifying system messages, but it must not be modified.

Reference Code *Reference Notes*

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
Bit 1	Routing code 2	Master console informational
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	Emulator program (under OS)
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	MLWTO message in response to a monitor or display command (exit routine not entered for MLWTO)
	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 operating system to terminate abnormally (code of F03).

Messages that Don't Use Routing Codes

The exit routine bypasses some messages. These are messages that have the MSGTYP operand in the WTO or WTOR macro instruction coded with the JOBNAMES, STATUS, or Y parameter, and messages that are being returned to the requesting console; that is, 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.

Adding a WTO/WTOR Exit Routine to the Control Program

A system generation option is available to enable you to include a resident, user-written exit routine in the communications task.

The OPTIONS=EXIT operand of the SCHEDULR system generation macro instruction controls the inclusion of the exit routine. A description of SCHEDULR is found in the *OS/VS1 System Generation Reference* publication.

Task supervision must be performed for the exit routine when the routine is requested at system generation. This supervision is performed every time a message is routed by its routing codes, even if the exit routine is not present. To maintain optimum throughput, the exit routine should not be specified at system generation unless it will be used.

Inserting the WTO/WTOR Exit Routine

To enter your exit routine into the control program before system generation. use the linkage editor to replace the dummy WTO/WTOR exit routine IEECVCTE in SYS1.AOSB with your WTO/WTOR exit routine, IEECVXIT.

To enter your exit routine into the control program after system generation, use the linkage editor to replace the dummy WTO/WTOR exit routine IEECVCTE in the SYS1.NUCLEUS with your WTO/WTOR exit routine.

WTO/WTOR (Write to Operator/Write to Operator with Reply) Macro Instructions

The WTO and WTOR macro instructions have two special operands, the MSGTYP and MCSFLAG operands. These operands should be used only by the system programmer who is thoroughly familiar with the multiple console support (MCS) communications task, since improper use of these operands can 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.

Note: Multiple-line WTO messages are not passed to the user-written WTO exit routine. The WTOR macro instruction cannot be used to pass multiple-line messages.

The MSGTYP and MCSFLAG operands may be specified on either the standard or list form of the WTO and WTOR macro instructions. The standard form of the WTO macro instruction is shown here.

[symbol]	WTO	{ 'message' } { ('text'[,line type]),... } [,ROUTCDE=(number[,number],...)] [,DESC=number] [,MSGTYP={ N | Y | JOBNAMES | STATUS | ACTIVE }] [,MCSFLAG=(name[,name],...)]

'message'

specifies that the message text is to be placed between the first and second apostrophes.

(*'text'* [*,line type*])

is used to write a multiple-line line message to the operator (no message is passed to the user-written WTO exit routine for multiple-line message).

ROUTCDE=

specifies the routing codes to be assigned to the message.

DESC=

specifies the descriptor codes to be assigned to the message.

MSGTYP=JOBNAMES or MSGTYP=STATUS

specifies that the message is to be routed to the console which issued the DISPLAY JOBNAMES or DISPLAY STATUS 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.

[ ] indicates optional name or operand; select one from vertical stack within { }

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. 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).

When MSGTYP=Y, 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), he must also set byte 0 of the MCSFLAG field in the macro expansion to a value of X'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. See Figure MSG 2 for bit definitions for MSGTYP=Y.

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.

Bit	Meaning
0	DISPLAY JOBNAMES
1	DISPLAY STATUS
2	Reserved for future system use. Must be zeros.
3	QID field exists.
4-15	Reserved for future system use. Must be zeros.

Figure MSG 2. Bit Definitions for MSGTYP=Y

Name	Bit	Meaning
-----	0	**Routing and descriptor fields exist.**
REGO	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	**MSGTYP field exists.**
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.
QREGO	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.
-----	9	MLWTO indicator
-----	10-12	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	**Reserved for Graphics.**
Note: Invalid specifications are ignored and produce an appropriate error message.		

Figure MSG 3. MCSFLAG Parameters

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 MSG 3.

ROUTCDE, DESC, and MSGTYP parameter combinations are shown in Figure MSG 4. Coding of any one of the four keyword parameters (ROUTCDE, DESC, MSGTYP, MCSFLAG) causes a new format WTO or WTOR to be generated.

Parameter Coded					Expansion Generates			
No.	ROUTCDE	DESC	MSGTYP	MCSFLAG	ROUTCDE	DESC	MSGTYP	MCSFLAG
1	Specified	Specified	Y	Optional	Codes Specified	Codes Specified	Zeros	As Specified #
2	Specified	Specified	N	Optional	Codes Specified	Codes Specified	Field Omitted	As Specified #
3	Specified	Specified	JOBNAMES	Optional	Codes Specified	Codes Specified	X'8000'	As Specified #
4	Specified	Specified	STATUS	Optional	Codes Specified	Codes Specified	X'4000'	As Specified #
5	Specified	Specified	Omitted	Optional	Codes Specified	Codes Specified	Field Omitted	As Specified #
6	Specified	Omitted	Y	Optional	Codes Specified	Zeros	Zeros	As Specified #
7	Specified	Omitted	N	Optional	Codes Specified	Zeros	Field Omitted	As Specified #
8	Specified	Omitted	JOBNAMES	Optional	Codes Specified	Zeros	X'8000'	As Specified #
9	Specified	Omitted	STATUS	Optional	Codes Specified	Zeros	X'4000'	As Specified #
10	Specified	Omitted	Omitted	Optional	Codes Specified	Zeros	Field Omitted	As Specified #
11	Omitted	Specified	Y	Omitted*	Routing Code 2	Codes Specified	Zeros	X'8000'
12	Omitted	Specified	N	Omitted*	Routing Code 2	Codes Specified	Field Omitted	X'8000'
13	Omitted	Specified	JOBNAMES	Omitted*	Routing Code 2	Codes Specified	X'8000'	X'8000'
14	Omitted	Specified	STATUS	Omitted*	Routing Code 2	Codes Specified	X'4000'	X'8000'
15	Omitted	Specified	Omitted	Omitted*	Routing Code 2	Codes Specified	Field Omitted	X'8000'
16	Omitted	Specified	Y	REG0/QREG0	Zeros	Codes Specified	Zeros	As Specified #
17	Omitted	Specified	N	REG0/QREG0	Zeros	Codes Specified	Field Omitted	As Specified #
18	Omitted	Specified	JOBNAMES	REG0/QREG0	Zeros	Codes Specified	X'8000'	As Specified #
19	Omitted	Specified	STATUS	REG0/QREG0	Zeros	Codes Specified	X'4000'	As Specified #
20	Omitted	Specified	Omitted	REG0/QREG0	Zeros	Codes Specified	Field Omitted	As Specified #
21	Omitted	Omitted	Y	Omitted*	Routing Code 2	Zeros	Zeros	X'8000'
22	Omitted	Omitted	N	Omitted*	Routing Code 2	Zeros	Field Omitted	X'8000'
23	Omitted	Omitted	JOBNAMES	Omitted*	Routing Code 2	Zeros	X'8000'	X'8000'
24	Omitted	Omitted	STATUS	Omitted*	Routing Code 2	Zeros	X'4000'	X'8000'
25	Omitted	Omitted	Omitted	Omitted*	Field Omitted	Field Omitted	Field Omitted	Zeros
26	Omitted	Omitted	Y	REG0/QREG0	Zeros	Zeros	Zeros	As Specified #
27	Omitted	Omitted	N	REG0/QREG0	Zeros	Zeros	Field Omitted	As Specified #
28	Omitted	Omitted	JOBNAMES	REG0/QREG0	Zeros	Zeros	X'8000'	As Specified #
29	Omitted	Omitted	STATUS	REG0/QREG0	Zeros	Zeros	X'4000'	As Specified #
30	Omitted	Omitted	Omitted	REG0/QREG0	Zeros	Zeros	Field Omitted	As Specified #

* If an MCSFLAG other than REG0 or QREG0 is specified, the expansion generates the same fields except that the MCSFLAG field contains the MCSFLAG specified and the high-order bit set to 1.

# High order bit set to 1 to indicate a new format macro expansion (routing code and descriptor code fields exist).

Figure MSG 4. ROUTCDE, DESC, and MSGTYP Combinations

The Must-Complete Function

This section provides information concerning system routine use of the must-complete function. This function is available to system routines operating in VS1 as an extension of the ENQ/DEQ facilities.

Section Outline

The Must-Complete Function

System routines (routines operating under a storage protection key of zero) often engage in updating and/or manipulation of system resources (system data sets, control blocks, queues, etc.) that contain information critical to continued operation of the system. These routines must complete their operations on the resource. Otherwise, the resource may be left in an incomplete state or contain erroneous information — either condition leads to unpredictable results.

The must-complete function is provided in the ENQ service routine to ensure that a routine queued on a critical resource(s) can complete processing of the resource(s) without interruptions leading to termination. The effect of the must-complete function is to place other routines (tasks) in a wait state until the requesting task — the task (routine) 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 through issuance of a DEQ macro instruction with the reset-must-complete (RMC) operand.

Realize that, for the time it is in effect, the must-complete function serializes operations to some extent in your 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, you should 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 serializes execution of tasks.

Scope

The must-complete function can be applied at two levels:

THE SYSTEM LEVEL

Only the requesting task, and system tasks included during system generation, are allowed to execute. All other tasks in the system are placed in a wait state.

THE STEP LEVEL

In a partition, only the requesting task is allowed to execute. All other tasks in the partition, 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 alternatives have been exhausted. Except for extremely unusual conditions the system level of must-complete should never be used.

Requesting the Must-Complete Function

You request the must-complete function by coding the set-must-complete (SMC) operand in an ENQ macro instruction. The format is:

name	ENQ	. . . , SMC = {SYSTEM STEP}

You may specify SYSTEM or STEP. The parameters SYSTEM and STEP indicate the level to which the must-complete function is to apply. The other operands of ENQ are described in the *Supervisor Services and Macro Instructions* publication.

Because of the properties of the TEST and USE parameters of the RET operand of the ENQ macro instruction, the SMC operand should be used only if the RET operand is to use the parameters HAVE or NONE (in the E-form of ENQ), or if the RET operand is not used at all.

You may request the must-complete function only in routines operating under a protection key of zero. If the protect key is not zero, the task using the routine requesting "must complete" is abnormally ended.

Operating Characteristics

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 tasks 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.

At the system or step level, the requesting task can cause its own abnormal termination. If the requesting task does come to an abnormal termination before a reset condition has been effected, the operating system is stopped at the point of error to permit investigation of the trouble. It is then necessary to restart the system with the initial-program-load (IPL) procedure.

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 should avoid calling another routine which also operates in the must-complete mode. However, one level of nesting is permitted, when necessary, with the following cautions:
   a. A task may set the must complete mode for both the system and the step. If multiple settings are made for either the system or the step, only the first setting of each is effective — the others are treated as no operation.
   b. The same is true for reset-must-complete. The first RMC for the system will reset the status of the system, the first RMC for the step will reset the status of the step, and all others will be treated as no operation.

3. Interlock conditions that can arise with the use of the ENQ function are discussed in the *Supervisor Services and Macro Instructions* publication.

   Additionally, an interlock may occur if your routine issues an ENQ macro instruction while in the must-complete mode. The resource you want to queue on may already be queued on by a task placed in the wait state due to the must-complete request you have made. Since the resource cannot be released, all tasks wait.
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 your 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.

Terminating the Must-Complete Function

Terminate the must-complete function and release the resource queued upon by coding the reset-must-complete (RMC) operand in a DEQ macro instruction. The format is:

name	DEQ	. . . , RMC = $\left\{\begin{matrix}\text{SYSTEM}\\ \underline{\text{STEP}}\end{matrix}\right\}$

The parameter (SYSTEM) or (STEP) *must agree* with the parameter specified in the SMC operand of the corresponding ENQ macro instruction.

Tasks placed in the wait state by the corresponding ENQ macro instruction are made dispatchable and asynchronous exits from the requesting task are enabled.

The PRESRES Volume Characteristics List

This section 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.

Section Outline

The PRESRES Volume Characteristics List

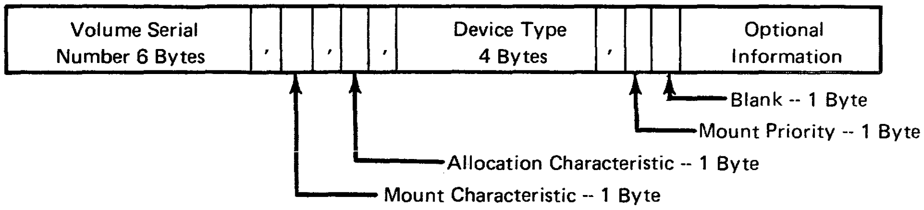
You may use the PRESRES volume characteristics list to define the mount and allocation characteristics of direct access device volumes used by your installation. Use of the list enables you to predefine the mount characteristics (permanently resident, reserved) and allocation characteristics (storage, public, private) for any, or all, direct access device volumes used by your installation. The *JCL Reference* publication provides a full discussion of the volume characteristics and the operating system's response to the various designations. The information presented here describes the creation of the characteristics list, the format and content of entries in the list, and how the operating system uses the list.

Creating the List

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

PRESRES Entry Format

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 characteristic 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 the optional information field is *always* preceded by a blank. All character representation is EBCDIC. This format is shown below.


Volume Serial Number 6 Bytes
,
,
,
Device Type 4 Bytes
,
Optional Information
Blank -- 1 Byte
Mount Priority -- 1 Byte
Allocation Characteristic -- 1 Byte
Mount Characteristic -- 1 Byte

The volume serial number consists of up to six characters, left justified. Mount characteristics are defined by:

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 characteristics are defined by:

0 to denote storage
1 to denote public
2 to denote private

The default characteristic is "public" and is assigned if any character other than 0, 1, or 2 is present in the field.

The device type is defined by:

A 4-digit number designating the type of direct access device on which the volume resides; for example, the IBM 2314 Direct Access Storage Facility is indicated by the notation 2314. Note that this field only indicates the basic device type for the associated volume. *You must advise the operator if the device requires special features (such as track overflow) to process the data on the designated volume.*

The mount priority field is used to suppress mount messages at IPL time for a volume; the alphabetic character N should be inserted in this field to suppress the mount message. 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 user does not wish to have the mount message suppressed, he may omit the mount priority field and the preceding comma.

The optional information field contains:

Any descriptive information about the volume that you may wish to enter. This information is not used by the system, but is available to you 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.

Operational Characteristics

Upon receiving control from the nucleus initialization program (NIP), the scheduler 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; the volume containing the data sets SYS1.LINKLIB, SYS1.PROCLIB, SYS1.SYSJOBQE; or a physically nondemountable volume, 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 selects the unit on which the volume is to be mounted.

The applicable *OS/VS Messages Library* publication describes the operator messages and responses associated with the use of the PRESRES volume characteristics 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.

Volume characteristics assigned by a PRESRES list entry are inviolate. They cannot be altered by subsequent references to the volume in the input stream.

Note:

1. A PRESRES entry identifying a physically nondemountable volume will appear in the mount list issued to the operator if the volume (device) is OFFLINE or is not present in the system.
2. Use of the PRESRES list can only be suppressed by deleting the member from the parameter library (SYS1.PARMLIB).
3. Only the first 102 volumes on the PRESRES list can be placed on the mount list.

Programming Considerations

The only way to assign an allocation characteristic other than "public" to volumes whose mount characteristic is "permanently resident" is through 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 determine the types of data sets that can be placed on the volume and its usage, you can exercise effective control over the volume through a PRESRES list entry.

System Reader, Initiator, and Writer Cataloged Procedures

In vs1, readers, initiators, and output writers are controlled by cataloged procedures. This section describes the reader, initiator, and writer cataloged procedures that are supplied by IBM, and tells how to write your own.

Section Outline

Continued —

Reader, Initiator, and Output Writer Cataloged Procedures

In vs1, system readers, initiators, and output writers are controlled by cataloged procedures. These procedures reside in the procedure library (SYS1.PROCLIB) and provide the parameters required for operation of the readers, initiators, and writers.

IBM supplies cataloged procedures for readers, initiators, and for output writers. You can:

- Use the IBM-supplied procedures.
- Use the IBM-supplied procedures, and override given parameters.
- Write and use your own cataloged procedures.
- Write and use your own cataloged procedures, and override given parameters.

The START command starts a reader, an initiator, or an output writer, and designates the procedure to be used. If you use the START command to start a problem program, there will be no SMF recording, or checkpoint/restart done for that job. You can override given parameters in the cataloged procedure by specifying the desired parameters in the START command. For a complete description of the START command, see the appropriate *OS/VS Operator's Library* publication.

Some of your installation's parameters may differ consistently from those in the IBM-supplied procedure. If so, you may wish to use your own cataloged procedure, rather than respecifying the parameters in every START command. You can use your own cataloged procedure by:

1. Writing the procedure in the required format.
2. Adding the procedure to the procedure library.
3. Specifying the procedure name in the START command.

To test your procedure by reference in another job but before adding it to the procedure library, format it as an in-stream procedure. See the *OS/VS Job Control Language Reference* publication for a description of in-stream procedures. (In-stream procedures can be used with any reader.)

If the parameter values in a cataloged reader, initiator, or writer procedure change frequently, use symbolic parameters in place of ordinary parameters. You may then assign values to the symbolic parameters in the START operator command. For a description of the START operator command, see the appropriate *OS/VS Operator's Library* publication. An illustration of the use of symbolic parameters is given in this section under *Example of the Use of Symbolic Parameters.*

Note: Symbolic parameters cannot be used for the program name on the EXEC statement in the reader, initiator, or writer procedure.

Data Set Integrity for System Tasks

Access during a job to a named data set depends on the disposition assigned it in the DD statement. If a data set is named (DSNAME=anyname) and its status is either OLD or NEW (DISP=status), the operating system gives exclusive control of that data set *name* to that job for the life of the job.

If you start several concurrent system tasks (such as several readers or several writers) using the same cataloged procedure, this data set integrity feature would nevertheless permit only one reader, or one writer, to execute at a time. To avoid this undesirable serialization of access (and hence, of the tasks) for readers, the SYS1.PROCLIB data set is assigned a status of SHR (in place of OLD). To avoid this for writers, the SYSOUT data set name is exempted from the protection of the data set integrity feature (since SHR cannot be assigned in place of NEW).

IEEVMPCR is the cataloged procedure called when you issue mount commands. This procedure resides in SYS1.PROCLIB. When not using an IBM-supplied cataloged procedure library, you should add IEEVMPCR to your own procedure library so that the mount commands can be properly executed. You can do this by using the IEBCOPY utility program.

Reader Procedures

A cataloged procedure for a reader requires two job control statements: an EXEC statement and a DD statement. A second DD statement may be necessary if the procedure library required is not SYS1.PROCLIB. See *DD Statement for the Procedure Library* in this section for specific details on when this statement is required. A third DD statement can be provided for reader ABEND. See *DD Statement for Storage Dump* in this section. The names and purposes of these statements are listed here:

- An EXEC statement with the step name IEFPROC specifies the reader program.
- A DD statement named IEFRDER provides the reader with a description of the input stream.
- A DD statement named IEFPDSI describes the procedure library (optional).
- A DD statement named SYSABEND/SYSUDUMP requests a storage dump (optional)

IBM supplies two reader procedures for VS1 named RDR and RDRT. RDR is used with local unit record input devices and RDRT is used with local non-unit record input devices. The two procedures follow. Your installation must modify the reader procedure for use with remote devices (see *Procedure Requirements*).

Procedure: RDR

```
// IEFPROC    EXEC    PGM = IEFVMA                                          X
//                    PARM = 'bppttttsscccrlaaaaefh'
// IEFRDER    DD      UNIT = 2540, LABEL = (,NL), VOLUME = SER = SYSIN,     X
//                    DCB = (LRECL = 80, RECFM = F)
```

Procedure: RDRT

```
// IEFPROC    EXEC    PGM = IEFVMA,                                         X
//                    PARM = 'bppttttsscccrlaaaaefh'
// IEFRDER    DD      UNIT = 2400, LABEL = (,NL), VOLUME = SER = SYSIN,     X
//                    DCB = (LRECL = 80, RECFM = F, BLKSIZE = 80)
```

A description of the PARM values is included under the heading *The PARM Field in the EXEC Statement of the Reader* later in this section. The IBM-supplied value of the PARM field in the EXEC statement of both the RDR and RDRT procedures is PARM='006003000005010E00011A'.

Procedure Requirements

When creating your own reader procedure, you must conform to the procedure format and the statement requirements. Use the IBM-supplied procedures as examples. The statement requirements are explained individually in the following paragraphs.

When creating a reader procedure for use with remote unit record input devices, you must provide an EXEC statement as explained here, and a DD statement as explained in *DD Statement for the Input Stream from a Remote Device* in this section.

The EXEC Statement

The EXEC statement specifies the reader program and also passes a set of parameters to it. The format of the EXEC statement is:

```
// IEFPROC     EXEC     PGM = IEFVMA,                                 X
//                      PARM = 'bppttttsscccrlaaaaefh'
```

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

PGM=IEFVMA

specifies the reader program. The name of the program must be IEFVMA, as shown.

The PARM Field in the EXEC Statement of the Reader

PARM=*'bppttttssccccrlaaaaefh'*

is a set of parameters for the reader and interpreter. This parameter field must consist of 21 characters, but the last seven have default values and need not be specified. Their meanings are explained in the following text.

b — character from 0 through 7, which indicates what the ADDRSPC= default will be, whether an account number is required or not, and whether a programmer name is required. The following chart shows the meaning of each possible character.

PARM field value b			
Character	ADDRSPC = default	Acct. info. req'd	Pgmr name req'd
0	VIRT	no	no
1	VIRT	no	yes
2	VIRT	yes	no
3	VIRT	yes	yes
4	REAL	no	no
5	REAL	no	yes
6	REAL	yes	no
7	REAL	yes	yes

pp – two numeric characters from 00 to 13 that indicate the default priority for jobs read from this input stream. When no priority is specified in the JOB statement, this default priority is assigned to the job.

ttttss — six numeric characters that indicate the default for the maximum time (in minutes and seconds) that each job step may run.

ccc – three numeric characters from 001 to 999 that indicate the default real size assigned to job steps read from this input stream when running virtual=real jobs.

r – a numeric character from 0 to 3 which specifies the disposition of commands from this input stream. The *r* parameter is used by the reader or interpreter whether or not the command is authorized to be entered into the input stream (see *aaaa* parameter). The reader and interpreter, if *r* is:

0 — passes the command to the command scheduling routine to be executed.

1 — displays the command (by a WTO macro instruction), and passes it to the command scheduling routine to be executed.

2 — displays the command (by a WTO macro instruction), asks the operator whether the command should be executed (by a WTOR macro instruction), and passes the command to the command scheduling routine if the operator replies in the affirmative.

3 — ignores the command (treated as a no-operation).

The WTO and WTOR macro instructions issued by the reader and interpreter are sent to the primary console in systems without the multiple console support (MCS) option and to the MCS master console in systems with the MCS option.

l — a numeric character 0 or 1 that specifies the bypass label processing options. 0 signifies that the BLP parameter in the label field of a DD statement is to be ignored. The label parameter is processed as NL. 1 signifies that BLP is not to be ignored. The label parameter is processed as it appears.

aaaa — four hexadecimal numbers. The first number must be an even number between 0 and E. The next three numbers must be 0. These numbers indicate which operator command groups are to be executed if read from this input stream. (WRITER commands are not allowed in the input stream.) This parameter is valid only for systems with the multiple console support option. In systems without the multiple console support option, this parameter is set to X'E000', permitting all commands except DEFINE and HALT to be entered into the input stream. In systems with the multiple console support option, default is to X'E000' when the parameter is omitted. This processing applies to those commands that are passed to the interpreter; that is, those within some job stream, and those commands processed in the reader outside of any job stream.

Figure PRO 1 shows the operator commands that are affected by the *aaaa* parameter in an MCS environment. 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.

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

Command Group	Function	Commands		
0	Information	BRDCST CONTROL DISPLAY LOG	MONITOR MSG MSGRT REPLY	SHOW STOPMN
1	System Control	CANCEL CENOUT DEFINE DUMP HALT HOLD	MODE MODIFY RELEASE RESET SET START	STOP SWAP SWITCH USERID WRITELOG WRITER
2	I/O Control	MOUNT SWAP	UNLOAD	VARY*
3	Console Control	VARY*		
1,2,3	Master Console	All commands are valid, plus CONTROL M VARY MSTCONS VARY HARDCOPY VARY CONSOLE with AUTH=		

*Note: VARY (Group 2) is accepted only to vary non-console device online or offline. VARY (Group 3) provides only for console switching and console reconfiguration on secondary consoles.

Figure PRO 1. Operator Command Groups

Bit settings for the *aaaa* parameter are:

Bytes	*Bits*	*Bit Settings*	*Meaning*
0 (aa)	0	1	Group 1 commands executed
	1	1	Group 2 commands executed
	2	1	Group 3 commands executed
	3–7	00000	Reserved
1 (aa)	0–7	00000000	Reserved

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

efh — MSGLEVEL value in absence of a value in the JOB statement. Unless there is a MSGLEVEL= entry 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 *efh* parameter. The values and their effects are:

e — kind of job control statement recorded.
- 0 — JOB statement only
- 1 — Input statements, cataloged procedure statements, and symbolic parameter substitution values.
- 2 — Input statements only.

A blank defaults to a value of 0.

f — kinds of allocation/termination messages recorded.
- 0 — None, except in the case of an ABEND condition. (In that event, all messages are recorded.)
- 1 — All

A blank defaults to a value of 1.

h — default message class.

DD Statement for the Input Stream from a Local Device

Your 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),                          X
//               VOLUME = SER = SYSIN,                                    X
//               DCB = (list of attributes) [, DSNAME = name, DISP = OLD]
```

This statement must be named 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 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.

Note: Label types AL and AUL (American National Standard Label types) should not be used.

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.

Note: The volume serial numbers should not identify a volume that contains a data set written in ASCII.

DCB=(list of attributes)

specifies the characteristics of the input stream and the number of buffers acquired by JAM (Job Entry Subsystem-oriented Access Method). The RECFM and LRECL subparameters must be present and only fixed 80-byte input is valid. For unit record devices, the BLKSIZE subparameter may be modified to take advantage of command chaining available through JAM. If you omit the BLKSIZE subparameter, JAM acquires three buffers of the size indicated at sysgen time. (See the *System Generation Reference* manual for more details.) If the BLKSIZE subparameter is coded, the value specified on the DD statement overrides the amount of buffer space required by the reader as specified at sysgen time. BLKSIZE must be a multiple of LRECL and a BLKSIZE=800 would allow JAM to read ten cards from the card reader with one CCW chain. Care should be taken to ensure that BLKSIZE (if specified) is not greater than that which was specified at sysgen. Otherwise, too little space may be available to allow readers and writers to operate at the level specified at sysgen time.

The BUFNO parameter in the IEFRDER DD card is used to define the number of buffers that JAM obtains. If this parameter is omitted, JAM obtains three buffers of the specified or defaulted BLKSIZE. If coded, the only valid entry is 1, which would cause JAM to obtain one buffer. One buffer can only be specified when BLKSIZE has been coded to indicate that command chaining is not being used; that is, BLKSIZE=80. Any other value is invalid and would result in three buffers being obtained. The specification of one buffer and no command chaining results in optimal use of storage since the JAM input buffer will become part of the reader workarea.

The DCB subparameters must be specified as required by QSAM when reading from non-unit record input devices. When starting a reader to a non-unit record device using the IBM-supplied RDR procedure, BLKSIZE must be specified with the START command; that is,

```
S   RDR.a,280,,DCB=BLKSIZE=80
```

When you use the RDRT procedure, it is not necessary to specify BLKSIZE.

DSNAME=name, DISP=disposition

specifies the name and disposition of the input stream data set to be read. This keyword should be used only with direct access input stream.

DISP=OLD

specifies that the input stream is an existing data set.

DD Statement for the Input Stream from a Remote Device

Your reader procedure for use with remote devices must include a DD statement that describes the input stream. The format for this statement is the same as the DD statement in the IBM-supplied reader procedure used with local unit record devices, except for a difference in the UNIT specification and the addition of another parameter. These parameter requirements are:

UNIT=RD*n*

specifies the remote unit record device from which the input stream is read. The device is specified in the form RD*n*, where *n* is the number representing the desired device of that type.

TERM=RT

indicates that the input stream is read from a remote device.

The UNIT specification can be overridden in the START command, but TERM=RT cannot be changed by the START command.

DD Statement for the Procedure Library

Your procedure for the reader can include a DD statement that defines the procedure library. For maximum performance, the user should restrict himself to either using the IEFPDSI DD statement, as indicated here, or omitting it. When used as illustrated or omitted, called procedures are not written to the SYS1.SYSPOOL data set by the reader. They are instead read from the cataloged system procedure library by the interpreter. You can make use of private procedure libraries by replacing the standard IEFPDSI DD statement with one describing the private procedure library or concatenating the private library(s) to the standard library by using the usual JCL conventions. However, if the reader procedure refers to a data set other than SYS1.PROCLIB, or supplies volume information, or concatenates procedure libraries, the reader will read the called procedures from their respective libraries and write them to the SYS1.SYSPOOL data set.

```
/ / 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, you must follow the IEFPDSI DD statement with 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 Storage Dump

Your procedure for the reader can include a DD statement that allows you to receive a SYSABEND or SYSUDUMP storage dump if the reader reaches ABEND. If the SYS1.DUMP data set is properly defined and available to accept the dump, you will receive an SVC dump. Otherwise, you will receive a SYSABEND or a SYSUDUMP, depending on which DD statement you include. If both SYSABEND and SYSUDUMP DD statements are included, the last one found is used.

```
//SYSABEND DD SYSOUT=A

              or

//SYSUDUMP DD SYSOUT=A
```

This statement can be named SYSABEND or SYSUDUMP, as shown.

SYSOUT=A
  specifies the output class for the dump.

Reader Procedure Used by Restart

The procedure, named IEFREINT, used to process job control statements for a job being restarted, is a skeleton of the normal reader procedure. 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:

```
                          Procedure: IEFREINT
/ / IEFPROC    EXEC   PGM = IEFVRRC,         RESTART READER PROGRAM    X
/ /                   REGION = 50K,          RESTART READER REGION     X
/ /                   PARM = RESTART
/ / IEFRDER    DD     DUMMY
/ / IEFPDSI    DD     DSNAME = SYS1.PROCLIB, DISP = SHR     PROCEDURE
                                                            LIBRARY
/ / IEFDATA    DD     DUMMY
```

Procedure Requirements

When creating your own restart reader procedure, you must conform to the procedure format and the statement requirements. Use the IBM-supplied procedures as examples. The statement requirements are explained individually in the following paragraphs.

The EXEC Statement

The EXEC statement specifies the restart reader program and passes a parameter to it. 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 as follows:

PGM=IEFVRRC
: specifies the restart reader program. The name of the program must be IEFVRRC, as shown.

REGION=nnnnnK (included for compatibility with VS2 and MVT. Not used by VS1.)

PARM=RESTART
: must be coded as shown.

DD Statement for the Input Stream

Your 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 as follows:

DUMMY
: must be coded as shown. System input is taken from the SYS1.SYSJOBQE data set which is open already.

DD Statement for the Procedure Library

Your 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 as follows:

DSNAME=SYS1.PROCLIB
: identifies the procedure library. To concatenate other data sets with the system library, you may follow the IEFPDSI DD statement with other unnamed DD statements thus expanding the system procedure library.

DISP=SHR
: specifies that the procedure library is an existing data set.

DD Statement for the CPP Data Set

Your procedure for the restart reader must include a DD statement that defines the CPP (concurrent peripheral processing) 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 as follows:

DUMMY
  must be coded as shown.

Initiator Procedures

A cataloged procedure for an initiator requires two job control statements, an EXEC statement and a DD statement for the Scheduler Work Area Data Set (SWADS). Additional DD statements may be optionally added so that specific control volumes will be allocated to the initiator task.

- An EXEC statement with the step name IEFPROC specifies the initiator program.
- A DD statement with the step name IEFRDER provides the specifications for the SWADS data set.
- Optional DD statements specify control volumes to be allocated to the initiator task.

IBM-Supplied Procedure

The initiator procedure supplied by IBM for vsl is named INIT. The INIT procedure is:

```
                              Procedure: INIT

/ / IEFPROC     EXEC     PGM = IEFIIC, PARM = 'A, LIMIT = 13'
/ / IEFRDER     DD       & & SWADS, UNIT = 2314, SPACE = (176, (250) , ,         X
/ /                      CONTIG), DCB = (BLKSIZE = 176, LRECL = 176,            X
/ /                      RECFM = F), DISP = (NEW, DELETE)
```

Procedure Requirements

When creating your own initiator procedures, you must conform to the procedure format and the statement requirements. The statement requirements are explained individually in the following paragraphs.

The EXEC Statement

The EXEC statement specifies the initiator program and passes a set of parameters to it. The format of the EXEC statement is:

```
 / / IEFPROC   EXEC    PGM = IEFIIC, PARM = 'x [(n)] [, x2 [(n2)]. . .          X
 / /                   [, LIMIT = K] ]
```

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

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 15 job classes may be named.) (This parameter is ignored in VS1.)

n — (0–15), a *force value* priority at which all jobs from the preceding class will be run. (This parameter is ignored in VS1.)

K — (0–15), the priority above which no job will be run by this initiator. (This parameter is ignored in VS1.)

The preceding PARM values are allowed for compatibility with MVT and with VS2 but are not processed in VS1.

DD Statement for the Scheduler Work Area Data Set (SWADS)

The Scheduler Work Area Data Set (SWADS) facility is activated when the initiator is started. During the system's "START INIT" processing, the data set is allocated and initialized. Since SWADS is a temporary data set, care should be taken to avoid its destruction when temporary data sets are scratched during the use of IEHPROGM. The cataloged procedure for initiators contains a DD statement for the SWADS. If an installation writes its own procedure for initiators, it must include this DD statement as specified here:

```
//IEFRDER    DD      DSNAME=&&SWADS,DISP=(NEW,DELETE),
                     UNIT=        ,SPACE=(        , (        ) , , CONTIG)
```

All parameters are required exactly as shown, except that DISP could be omitted since NEW,DELETE is the default and UNIT and SPACE contain values determined by the installation. These fields should be specified according to the following:

UNIT: Must be for one of the valid DASD types (2314, 2319, 2305-2, 3330, or 3333). It can be expressed as a unit type (2314) or as a group designation (SYSDA), as long as the group does not include any non-supported device types, or as an actual channel and unit address. (When expressed as unit type, a 2319 is specified as 2314, and a 3333 is specified as 3330.)

Because all SWADS need not be on the same type device, an incompatibility is possible if the job uses the automatic step or checkpoint/restart features of VS1. If the SWADS of the initiator trying to restart the job has a smaller track capacity than that of the SWADS used when the job was first run, restart is not possible. In this case, the operator is informed and given the option of holding the job for restart by another initiator or of canceling the job.

SPACE: The amount of space to request can be calculated according to a formula provided for the purpose. The formula is found in the *VS1 Storage Estimates* publication. This value, of course, depends on the size of the jobs that will be run in the installation. The subparameter CONTIG is always needed. The actual request can be in terms of CYL, or TRK, or 176-byte blocks.

The IBM-supplied initiator procedure contains values for UNIT and SPACE. These values should, if necessary, be changed by the installation to reflect the needs of the specific system more accurately. You also have the ability to override these DD parameters in your START command for the initiator. This is useful if a particular job to be run on a given day is known to be unusually large. It will be advantageous for the installation to override the SPACE parameter when starting the initiator, rather than risk having the job canceled for lack of SWADS space. Generally, the overrides are useful when temporary changes are required. For permanent changes, the procedure should be modified so that the format of the START command is simplified.

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 you use dedicated data sets for temporary data sets, the checkpoint/restart facility is internally suppressed. To dedicate any data set quickly to successive jobs or job steps, you add a DD statement to the initiator procedure. An initiator procedure (INITD) for use of dedicated data sets with processor programs has been added to the system. To save repeated catalog searches, you may also dedicate system library data sets.

The dedicated data sets feature has been implemented by adding code to the allocation routine that, before allocating space for a temporary data set, attempts to relate a request for a temporary data set with a dedicated data set. If the space required for the temporary data set fits within the dedicated data set, the dedicated data set space is used. If not, normal allocation takes place. The same criterion will be used with presently coded requests for temporary data sets, that is, if the space requested is within the range of the dedicated data set, it will be used.

How to Dedicate a Data Set

You dedicate a data set by adding a DD statement (for each data set to be dedicated) to the initiator procedure. The unit must be a DASD; the space may be for a sequential or partitioned data set. Each DD statement must be of the form:

```
/ / ddname    DD    UNIT = unitparms, VOL = volparms,
                    SPACE = (kind, (amount, increment, dirblks)), DISP = (NEW,
                    DELETE)
```

ddname

A user supplied ddname must be given 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.

unitparms

Parameters that describe the unit to be used for the dedicated data set. The unit must be a DASD. 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.

volparms

Volume parameters.

A volume may be specified for each unit specified in the preceding unit parameter entry. The volume parameters specified here override those of the job step DD statement for which the dedicated data set is used.

(kind,(amount,increment,dirblks))

Type and size of space (in terms of CYL, TRK, *avgbl,* or ABSTR) to be allocated to the data set. If *,dirblks* is obmitted, 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. Increments, once allocated, remain allocated until the initiator stops.

NEW,DELETE

These disposition parameters may either be coded explicitly or may take effect by default, that is by omitting the DISP= entry.

The effect of NEW is that the data set is freshly 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.

DSNAME

The allocation procedure for an initiator pre-allocated data set is the same as for any temporary data set. This procedure is simplest with no dsname= entry in the DD statement. That results in a system assigned data set name of the form:

SYSnumber.Rnumber.procname.RVnumber.

You may also code DSNAME=&name, DSNAME=&&name, or DSNAME=name. These names will override those used in the job step DD statement for which the dedicated data set is used.

DCB parameters:

DCB parameters specified here have no effect.

How to Get to Use a Dedicated Data Set

If you want a dedicated data set to be used for a data set needed temporarily in a job step, define the temporary data set in a DD statement of the form:

```
/ / ddname    DD      DSNAME = &ddname,
/ /                   SPACE = (avgbl, (amount, increment, dirblks)),
/ /                   UNIT = unitparms, DISP = (NEW, DELETE), DCB = dcbparms
```

&ddname

name of the DD statement for the dedicated data set, preceded by an & sign.

(avgbl,(numbr,increment,dirblks))

Space request, in terms of average block length only, needed for this 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 here 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 space for *,dirblks* requested here exceeds the space for *,dirblks* specified in the dedicated data set.
- If the space request is shown in other than average block length.

unitparms

Unit parameters

Parameters that describe the unit to be used for the temporary data set, if the dedicated data set is *not* used. Here, the unit may be a magnetic tape unit, as well as a DASD.

NEW,DELETE

These disposition parameters must either be coded explicitly or may take effect through default.

dcbparms

DCB parameters required for your temporary data set. Unless specified, you may find that a previous user has left the dedicated data set with undesired DCB parameters.

Procedure 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 including it 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, start the INITD initiator.

Before including the INITD procedure in your system, review the space allocations, unit specifications, and ddnames used in the procedure against your requirements. If they are significantly different, you may wish to code your own.

Existing procedures can be used under INITD initiators. However, the job stream may be affected as described under *Disposition of Temporary Data Sets.* Procedures designed for the dedicated data set feature remain operative without the presence of the dedicated data set feature. In short, the procedure will run under any initiator regardless of whether that initiator has dedicated data sets.

The INITD procedure is as follows:

```
                              Procedure: INITD

//IEFPROC     EXEC    PGM=IEFIIC,PARM='A,LIMIT=13'
//IEFRDER     DD      DSN=&&SWADS,UNIT=SYSDA,                              X
//                    SPACE=(176,(250),,CONTIG),DISP=(NEW,DELETE)
//SYSABEND    DD      SYSOUT=A,SPACE=(TRK,(1,10))
//SYSUT1      DD      DSNAME=&UT1,SPACE=(1700,(200,100),,CONTIG),          X
//                    UNIT=(SYSDA,SEP=IEFRDER)
//SYSUT2      DD      DSNAME=&UT2,SPACE=(1700,(200,100)),                  X
//                    UNIT=(SYSDA,SEP=SYSUT1)
//SYSUT3      DD      DSNAME=&UT3,SPACE=(1700,(200,100)),                  X
//                    UNIT=(SYSDA,SEP=SYSUT2)
//SYSUT4      DD      DSNAME=&UT4,SPACE=(460,(700,100)),                   X
//                    UNIT=(SYSDA,SEP=SYSUT3)
//LOADSET     DD      DSNAME=&LOADSET,UNIT=(SYSDA,SEP=SYSUT1),             X
//                    SPACE=(3600,(100,10))
```

INITD Procedure Statements

Each of the statements shown in the preceding illustration is explained in detail in the following. In addition to describing the reason for or effect of the use of a parameter, the description distinguishes between those parameters that must be coded as shown and those that you may override or substitute for.

The EXEC Statement

The EXEC statement for the procedure is:

```
//IEFPROC    EXEC    PGM = IEFIIC, PARM = 'A, LIMIT = 13'
```

IEFPROC

The step name. Must be coded as shown.

EXEC

The job control statement name. Must be coded as shown. Defines the beginning of a job step.

PGM=IEFIIC

The program to be executed in this job step. IEFSD060 is the name of the initiator program. Must be coded as shown. Whether dedicated data sets are used depends on the DD statements that follow, not on the name of the program.

PARM='A,LIMIT=13'

Parameter list for the initiator program. A is the class of jobs to be processed, LIMIT=13 is the dispatch priority limit for this initiator. These values are included for compatibility with MVT and VS2, but are ignored in VS1.

DD Statement for the Scheduler Work Area Data Set (SWADS)

The DD statement for the SWADS data set used with the INITD procedure is the same as that described previously under the writeup of the INIT procedure.

DD Statements for the Dedicated Utility Data Sets

Four DD statements in the INITD procedure allocate space to four commonly used utility (or scratch) 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

The leading & sign marks the name as that of a temporary data set.

SPACE=

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 first data set, SYSUT1, all the primary space is to be continguous when allocated. The fourth data set is to be allocated space for 700 blocks of 460 bytes initially. When exhausted, space is to be allocated for 100 blocks at a time.

UNIT=

Space is 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

DDName of the dedicated data set.

DD

Data definition statement

DSNAME=&LOADSET

A temporary dataset

SPACE=(3600,(100,10))

Space allocation commonly used in compilers.

UNIT=

Space is to be allocated on a DASD but not the same one as the SYSUT1 data set.

Use of Dedicated Data Sets by Processor 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, let us look at a present DD statement and the change required.

The following is a DD statement from the COBECLG 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 dataset, assuming that the step is running under the INITD procedure, add a DSNAME=&name (or &&name) entry referring to the dedicated data set to be considered for use:

```
//SYSUT1     DD    UNIT = SYSDA, SPACE = (1024, (200, 65)), DSNAME =      X
//                 &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, the organizations are not the same, or 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.

Processor Programs Library Data Sets as Dedicated Data Sets

Processor programs library data sets, such as the COBOL library, for example, may be referred to repeatedly 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 nontemporary data sets. The DD statement in the initiator procedure must have the disposition specified as OLD or SHR 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 referencing the dedicated library must also be OLD or SHR 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 the COBECLG procedure that results in the allocation of the 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 to the initiator procedure and replace it 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 COBECLG procedure, after adding the present one to the initiator procedure, is:

```
//SYSLIB     DD     DSNAME = &SYSLIB, DISP = (SHR, KEEP)
```

The result is one catalog search per initiator start instead of one catalog search every job step. However, keep in mind that this COBECLG 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 you attempt 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 you are using cataloged procedures and run the same procedure twice within the same job. For example: assume that you use the procedure PL1LFLCLG 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.

You can avoid this problem by not using dedicated data sets for jobs that run the same cataloged procedure twice. Alternatively, you could submit each cataloged procedure as separate jobs instead of submitting them as separate job steps within the same job.

You can use the following chart to determine the disposition, by allocation/termination, of temporary dedicated data sets.

If you code DISP =	Allocation/termination treats it as:
NEW	OLD
OLD/SHR	OLD
MOD	OLD
,DELETE	KEEP
,PASS	PASS
,KEEP	KEEP

Output Writer Procedures

A cataloged procedure for output writers requires two job control statements: an EXEC statement and a DD statement. A second DD statement can be provided for writer ABEND. See *DD Statement for Storage Dump* in this section.

- An EXEC statement with the step name IEFPROC specifies the output writer program.
- A DD statement named IEFRDER defines the output data set.
- A DD statement named SYSABEND/SYSUDUMP requests a storage dump (optional).

System Output Writers

IBM supplies two output writer procedures for VS1 named WTR and WTRT. WTR is used with local unit record output devices and WTRT is used with local non-unit record output devices. The two procedures follow. Your installation must modify the writer procedure for use with remote devices (see *Procedure Requirements*).

```
                         Procedure:    WTR

//IEFPROC     EXEC     IEFOSC01,                                        X
//                     PARM = 'PA'
//IEFRDER     DD       UNIT = 1403, VOLUME = (, , , 35),                X
//                     DSNAME = SYSOUT, DISP = (NEW, KEEP),             X
//                     DCB = (LRECL = 133),                             X
//                     RECFM = FM
```

```
                         Procedure: WTRT

//IEFPROC   EXEC   PGM = IEFOSC01,PARM = 'PA'
//IEFRDER   DD     UNIT = 2400, VOLUME = (, , , 35), DSNAME = SYSOUT,   X
//                 DISP = (,KEEP),                                     X
//                 DCB = (RECFM = FM, LRECL = 133, BLKSIZE = 133)
```

Procedure Requirements

When creating your own output writer procedure, you must conform to the procedure format and the statement requirements. Use the IBM-supplied procedure as an example. The statement requirements are explained individually in the following paragraphs. When creating a writer procedure for use with remote unit record output devices, you must provide an EXEC statement, as explained here, and a DD statement as explained in *DD Statement for the Output Data Set for a Remote Device* in this section.

The EXEC Statement

The EXEC statement specifies the output writer program and passes a set of parameters to it. The format for the EXEC statement is:

```
//IEFPROC    EXEC    PGM = IEFOSC01,                                    X
//                   PARM = 'cxxxxxxxx, seprname,#,TR,no,nnnn,y'
```

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

PGM=IEFOSC01
: specifies the output writer program. The name of the program must be IEFOSC01, as shown.

PARM=*'cxxxxxxxx,seprname,#,*TR,*no,nnnn,y'*

is a set of parameters for the output writer program. The parameters are positional and a comma must be present to indicate the absence of a parameter. The various parameters are explained:

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 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. You can specify the name IEFOSC06 to use the output separator supplied by IBM, or you can specify the name of your own program, which must reside in the link library (SYS1.LINKLIB). This subparameter may be omitted, in which case no output separator is used. (Output separators are described in the *Output Separation* section.)

# — the number (1, 2, or 3) of separator pages (printer) or cards (punch) produced by the IBM-supplied output separators. This parameter is valid only if SEPRNAME is present and will default to three if omitted. This parameter is also passed to user-written output separator programs (see *Output Separation,* a preceding section of this publication).

TR — The TR parameter causes the writer to translate any unprintable character to a blank. This parameter is valid only when the output device is a printer and the function is not performed if TR is omitted.

no — the number of lines per page of printed output, with a maximum value of 254. This line count is only in effect when the data set does not contain control characters. This parameter is valid only when the output device is a printer and will default to 60 if omitted.

nnnn — checkpoint interval. The number of pages (for printer) or logical records (for punch/tape devices) between checkpoints. This optional specification causes the writer to checkpoint SYSOUT data sets. SYSOUT data sets that are smaller than the indicated checkpoint interval are not checkpointed. If the number is not a multiple of 10, it is rounded down to the nearest multiple of 10.

Output Device Type

Printer — nnnn=20 for a 1403 would cause a checkpoint in a data set approximately once a minute.

Punch/Tape — nnnn=300 for a 2540 would cause a checkpoint in a data set approximately once a minute.

y — the number of end-of-job separator (1, 2, or 3) pages to be passed to the output separator program. The IBM-supplied separator prints that number of pages. This is valid only if *seprname* has been specified and output is destined for a printer device. Printer or punch destination is based on DD information, control character specification, and the writer parm field for non unit record devices.

DD Statement for the OUTPUT Data Set for a Local Device

Your 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
                                   { ,ALIGN  }
//                  FCB = (image-id { ,VERIFY } )
```

This statement must be named 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. The devices that can be used are: 1403, 1442, 1443, 2400, 2400-1, 2400-2, 2400-3, 2400-4, 2520, 2540, 3211, or 3525 with read feature.

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), 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 stream and the number of buffers acquired by JAM. The RECFM and LRECL subparameters must be present in your procedure. For unit record devices, the BLKSIZE subparameter may be modified to take advantage of command chaining available through JAM. If you omit the BLKSIZE subparameter, JAM will acquire three buffers of the size indicated at system generation time (see the *VS1 SYSGEN* manual for more details). If the BLKSIZE subparameter is coded, the value specified on the DD statement overrides the amount of buffer space required by the writer as specified at sysgen time. BLKSIZE must be a multiple of LRECL, and a BLKSIZE=798 would allow JAM to accumulate six 133-byte print records before initiating the I/O to the output device. Care should be taken to ensure that BLKSIZE, if specified, is not greater than that specified at sysgen. Otherwise, not enough space may be available to allow readers and writers to operate at the level specified at sysgen.

: The BUFNO parameter in the IEFRDER DD card is used to define the number of buffers that JAM obtains. If this parameter is omitted, JAM will obtain three buffers of the specified or defaulted BLKSIZE. If coded, the only valid entry is 1, which causes JAM to obtain one buffer. One buffer can only be specified when BLKSIZE has been coded to indicate that command chaining is not being used. Any other value is invalid and would result in three buffers being obtained. The specification of one buffer with no command chaining results in optimal use of storage since the JAM output buffer will become part or the writer workarea.

The DCB subparameter must be specified as required by QSAM when writing to non-unit record output devices. When starting a writer to a non-unit record device using the IBM-supplied WTR procedure, BLKSIZE must be specified with the START command, that is,

```
S    WTR.W,280, , DCB=BLKSIZE=133
```

When you use the WTRT procedure, it is not necessary to specify BLKSIZE.

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 zero and one to be ignored when comparing characters between the UCS buffer and the print line buffer. This option allows lowercase character codes to be printed in uppercase by an uppercase chain/train. VERIFY causes the UCS image specified to be output for the printer. The UCS parameter is optional and is valid only when the output device is a 3211 printer or a 1403 printer.

FCB=(*image-id* $\begin{bmatrix}\text{,ALIGN}\\ \text{,VERIFY}\end{bmatrix}$)

causes a 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. ALIGN and VERIFY each allow the operator to align forms. VERIFY also causes the FCB image to be output for the printer. The FCB parameter is optional and is valid only when the output device is a 3211 printer.

DD Statement for Storage Dump

Your procedure for the writer can include a DD statement that allows you to receive a SYSABEND or SYSUDUMP storage dump if the writer reaches ABEND. If the SYS1.DUMP data set is properly defined and available to accept the dump, you will receive an SVC dump. Otherwise, you will receive either a SYSABEND or a SYSUDUMP, depending on which DD statement you include.

```
//SYSABEND DD SYSOUT=A

              or

//SYSUDUMP DD SYSOUT=A
```

This statement can be named SYSABEND or SYSUDUMP, as shown. If both DD statements are included, the last one found is used.

SYSOUT=A

specifies the output class for the dump.

DD Statement for the OUTPUT Data Set for a Remote Device

Your writer procedure for use with remote devices must include a DD statement that describes the OUTPUT data set. The format for this statement is the same as the DD statement in the IBM-supplied writer procedure used with local unit record devices except for a difference in the UNIT specification and the addition of another parameter. These parameter requirements are:

UNIT=*aan*
specifies the remote unit record device used to handle the output.

aa — either PR (for printer) or PU (for punch) designates the device type.

n — a numeric character that represents the desired device of that type.

TERM=RT
indicates that the output is going to a remote device.

Direct SYSOUT Writer Procedures

The direct SYSOUT writer is an option in VS1 that causes program output to be written directly from the problem program synchronously, with the exception of system messages, with execution of the problem program. There are two IBM-supplied direct system output writer procedures; DSO and DSOJS. The difference in the two procedures is that DSOJS provides that output separators are written prior to the problem program output. A user-supplied procedure can be invoked, but it must execute the IBM-supplied direct system output writer. Both IBM-supplied procedures require two job control statements; an EXEC statement and a DD statement.

- The EXEC statement is named IEFPROC.
- The DD statement is named IEFRDER and describes the ultimate output data set.

The procedures supplied by IBM follow. If you wish to create your own procedure, follow their formats.

Procedure: DSO

```
//IEFPROC   EXEC   PGM = IEFDSO, REGION = 8K, PARM = (PA, , , A)
//IEFRDER   DD     UNIT = 2400, DSN = SYSOUT, DISP = (NEW, KEEP),        X
//                 LABEL = (, SL), VOL = (, , , 05), DCB = (BUFNO = 3)
```

Procedure: DSOJS

```
//IEFPROC   EXEC   PGM = IEFDSO, REGION = 8K, PARM = (PA, IEFOSC06,3,A)
//IEFRDER   DD     UNIT = 2400, DSN = SYSOUT, DISP = (NEW, KEEP),        X
//                 LABEL = (,SL), VOL = (, , , 05), DCB = (BUFNO = 3)
```

The EXEC Statement

The EXEC statement specifies the direct SYSOUT writer. It is also used to give the writer program operating information.

```
//IEFPROC    EXEC    PGM = IEFDSO, REGION = 8K, PARM = (cx, seprname,
//                   nosep, jjj)
```

IEFPROC

Name of the EXEC statement. It is required as shown.

IEFDSO

Name of the writer program.

REGION=8K

Space required by IEFDSO to start. This parameter is included for compatibility with MVT and VS2. It is not used by VS1.

PARM=

Information for the IEFDSO program.

c — A letter, P for printer, or C for card punch, that describes the ultimate hard-copy medium. Must be given.

x – The SYSOUT class to be processed. If stated here, and in the START command, the latter rules. If not stated here, it must be given in the START command.

,seprname

Output separation program name. This may be omitted, but a comma must be written if other items follow.

IEFOSC06 — Name of the IBM-supplied separator program.

,nosep

Number of output separators. This may be omitted if *seprname* is included, but a comma must be written if other items follow. The default value is three.

,jjj

Job classes to be processed. From one to fifteen letters (A-O) showing the job classes to be processed. If any job classes are named in the START command, they overrule all stated here. If none are named here, the job classes are those assigned to the partition for which this writer is started.

The DD Statement

This DD statement describes the kind of volume to be used and the format of the data set.

```
//IEFRDER    DD    UNIT = name, DSN = anyname, DISP = (NEW, KEEP),          X
//                 LABEL = (, SL), VOL = (, , , volcount, DCB = (list)
```

IEFRDER

Name of the DD statement. Required as shown for IEFDSO.

name

Any form of unit identification may be used, such as 00E, 2400, or TAPE. Multiple parallel units (UNIT=2400,2) cannot be used.

DSN=*anyname*

Name of a non-temporary data set. A name must be given. If stated here and in the START command also, the latter rules. The name is used in the disposition messages at step termination, and must be used to identify the data set if it is to be printed later from tape.

DISP=(NEW,KEEP)

Required disposition.

LABEL=(,SL)

If DSO is being used to write to magnetic tape, standard label tapes are required. The label description may be stated explicitly or may be omitted, in which case SL is assumed.

,,volcount

1-225. The maximum number of volumes a data set to be processed by this writer will have. This determines the amount of job queue space allocated to each SYSOUT data set processed by this writer. After the first five volumes, each subsequent 15 require another job queue record. If omitted, one is assumed. If stated here and also in the START command, the latter rules. This value cannot be given in the DD statement of a job to be processed.

list

The following DCB parameters gain control only if they are not also given in the SYSOUT DD statement or in the DCB macro instruction (that is, default values can be stated in this procedure):

BFALN, BFTEK, BUFL, BUFNO, BLKSIZE, LRECL, RECFM, NCP, HIARCHY, UCS.

The following DCB parameters, if stated here, override all except those given in a START command:

CODE, DEN, MODE, OPTCD, PRTSP, STACK, TRTCH.

Cataloging the Procedure

Use the IEBUPDTE utility program to add your reader, initiator, or writer procedures to the cataloged procedure library (SYS1.PROCLIB). Use of this program is fully explained in the *OS/VS Utilities* publication.

The following example shows the control statements needed to add a reader procedure and an output writer procedure to the procedure library. For this example, the reader procedure is named RDPROC4, and the output writer procedure is named WTPROC2.

The EXEC statement in this example specifies the IEBUPDTE program. The PARM=NEW parameter indicates that all input to the utility program is contained in the data set defined by the SYSIN statement.

The ADD control statement furnishes the name of the member to be added to the procedure library. The three numbers following the member name indicate:

- The level of modification (00 indicates first run).
- The source of the modification (0 indicates user-supplied).
- The printed output desired (ALL indicates print entire updated member and control statements).

The NUMBER statement specifies the sequence numbers for records within the new member. With this statement, the number 00000010 is assigned to the first record of the new procedure, and subsequent records are incremented by 00000010.

```
//NEWPROCS    JOB       09#770,D.P.BROWN
//            EXEC      PGM = IEBUPDTE, PARM = NEW
//SYSPRINT    DD        SYSOUT = A
//SYSUT2      DD        DSNAME = SYS1.PROCLIB, DISP = OLD
//SYSIN       DD        DATA
./            ADD       RDPROC4, LEVEL = 00, SOURCE = 0, LIST = ALL
./            NUMBER    NEW1 = 10, INCR = 10
//IEFPROC     EXEC      PGM = IEFVMA,                                X
//                      PARM = '00101005010E00001A'
//IEFRDER     DD        UNIT = 2400-2, LABEL = (, NL),               X
//                      VOLUME = SER = SYSIN,                        X
//                      DCB = (BLKSIZE = 80, LRECL = 80, BUFL = 80,  X
//                      BUFNO = 1, RECFM = F, TRTCH = C, DEN = 0)
//IEFPDSI     DD        DSNAME = SYS1.PROCLIB, DISP = SHR
./            ADD       WTPROC2, LEVEL = 00, SOURCE = 0, LIST = ALL
./            NUMBER    NEW1 = 10, INCR = 10
//IEFPROC     EXEC      PGM = IEFOSC01,                              X
//                      PARM = 'PAC'
//IEFRDER     DD        UNIT = 2400-2,LABEL=(,NL),VOLUME=(,,,40),    X
//                      DSNAME = SYSOUT, DISP = (NEW, KEEP),         X
//                      DCB = (BLKSIZE = 133, LRECL = 133, RECFM = F, X
//                      TRTCH = C)
/*
```

Example of the Use of Symbolic Parameters in Cataloged Reader, Writer, and Initiator Procedures

Symbolic parameters in a cataloged procedure that is started by the START operator command may be assigned values in the START command that starts the procedure. In this manner, any parameter in the EXEC or in any DD statement may be assigned a value at the time the procedure is started.

A cataloged procedure that uses symbolic parameters may also have a PROC statement that shows the default values for the symbolic parameters. Keywords that may be used in a JOB, EXEC, or DD statement cannot be used as symbolic parameters. (For example, you cannot say that DISP is equal to &REGION.) However, subparameter keywords of the DD statement can be used as symbolic parameters. (For example, you may code BUFNO=&BUFNO.)

The following example shows a reader procedure that contains symbolic parameters.

```
//RDPR5      PROC     STIME = 030, MCS = E000, MSGL = 01,
//                    PDSI = 'SYS.1. PROCLIB'
//IEFPROC    EXEC     PGM = IEFVMA
//                    PARM = '001&STIME.05010&MCS&MSGL.A'
//IEFRDER    DD       UNIT = 2400, LABEL = (,NL), VOLUME =SER=SYSIN,
//                    DCB = (BLKSIZE = 80, LRECL = 80, BUFL = 80,
//                    BUFNO = 1, RECFM = F)
//IEFPDSI    DD       DSNAME = &PDSI, DISP = SHR
```

The PROC Statement

In the preceding illustration the PROC statement assigns default values to the symbolic parameters &STIME,&MCS,&MSGL,&PDSI.

The START Command

These same symbolic parameters are assigned values with the following START command:

```
START   RDPR5, STIME = 035, MCS = E000, MSGL = 11, PDSI = 'SYS1. USER'
```

Blocking the Procedure Library

You may, in some cases, improve the use of direct access space and gain performance advantages by blocking the procedure library. It may be blocked at system generation or subsequently by using the operating system utilities. Block size must be a multiple of 80.

The following example shows the control statements needed to block the procedure library using the IEBCOPY and IEHPROGM utility programs. Step C1 of job BLOCK copies the procedure library and blocks it to 400. It deletes the old copy and catalogs the new copy under the name of LIBCOPY. Step R1 renames the procedure library to SYS1.PROCLIB and catalogs it under that name.

```
//BLOCK      JOB     ACCT, D82, MSGLEVEL = 1
//C1         EXEC    PGM=IEBCOPY
//SYSUT1     DD      DSNAME = SYS1.PROCLIB, UNIT = 2314, DISP = (OLD,
//                   DELETE, KEEP)
//SYSUT2     DD      DSNAME = LIBCOPY, UNIT = 2314, VOLUME =                X
//                   SER = 111111,                                          X
//                   DISP = (NEW, CATLG, DELETE), DCB = (RECFM = FB,        X
//                   LRECL = 80,                                            X
//                   BLKSIZE = 400), SPACE = (TRK, (50, 1, 10))
//SYSPRINT   DD      SYSOUT = A
//SYSIN      DD      DUMMY
/*
//R1         EXEC    PGM = IEHPROGM
//DD1        DD      UNIT = 2314, VOLUME = SER = 111111, DISP = OLD
//SYSPRINT   DD      SYSOUT = A
//SYSIN      DD      *
  RENAME     DSNAME = LIBCOPY,VOL=2314=111111,NEWNAME=
             SYS1.PROCLIB
   CATLG     DSNAME = SYS1.PROCLIB,VOL=2314=111111
/*
```

Resident Routines Options

The resident routines options are the BLDL feature, the resident reenterable modules feature, and the RSVC and RERP features. These features permit preloading into storage routines (or at least their addresses) that otherwise would be repeatedly loaded each time the routines are requested. The purpose of these options is to improve performance by reducing or eliminating the access time required to obtain the routines with which these options are concerned.

The link list feature, also described in this section, permits references to the link library to be extended to other data sets.

Section Outline

BLDL and Resident Routines Features

The BLDL, reenterable modules, RSVC and RERP options, when included in your system, enable you to make resident in the pageable supervisor area:

1. All, or a selection of, link or SVC library directory entries.
2. A selected group of access method routines.
3. A selected group of type 3 and 4 SVC routines.
4. A selected group of error recovery procedures.
5. Reenterable routines from the link library, and SVC library.

Placement occurs during the system initialization process. The routines or SVC/link library entries may be pageable or fixed (RERP routines are always fixed) and reside in the pageable supervisor. Additionally, the fixed items are long-term fixed in the high end of real storage. That is, the fixed items occupy space in both virtual and real storage but are not subject to paging.

In the following discussions, the term *resident* is often used. As applied to the resident routines option, resident means existing in the pageable supervisor area and may be pageable or fixed, depending on the option selected (RAM or RAMF, RSVC or RSVCF, BLDL or BLDLF).

These options are included in the system when it is generated. The *System Generation Reference* publication describes the procedure.

You specify:

1. the link library (SYS1.LINKLIB) and SVC library (SYS1.SVCLIB) routines and directory entries
2. the access method routines
3. the type 3 and 4 SVC routines
4. the error recovery procedures to be made resident through lists of linkage library, access method, SVC routine and error recovery procedure load module names placed in the parameter library (SYS1.PARMLIB).

Standard lists (as shown in Figure RRO 1) exist for every option. The standard list (so called because its member name in the parameter library is predefined) is automatically referred to during the initialization process when the option is either selected or defaulted to during sysgen and is neither canceled or modified in response to message IEA101A SPECIFY SYSTEM AND/OR SET PARAMETERS FOR RELEASE xx.yy.sssss. All standard lists, except IEABLD00, are built at sysgen. IEABLD00 is copied from the starter system. Some standard lists contain module names and some lists are null.

A portion of this section discusses the function of each option, the creation of the parameter library lists, and, lists the content of the resident access method modules and resident type 3 and 4 SVC routines standard lists. The *Message Library* publication describes the message (message number IEA101A) and replies associated with the options.

The Resident BLDL Table Option

System issued ATTACH, LINK, LOAD, or XCTL macro instructions requesting load modules from partitioned data sets cause a search of the data set directory for the location of the requested module (the BLDL table operation) and a fetch of the module. The resident BLDL table option eliminates the directory search required during execution of these macro instructions when a load module (whose directory entry is resident) is requested from the linkage or SVC libraries.

This option builds lists of directory entries for use by ATTACH, LINK, LOAD, or XCTL macro instructions requesting linkage or SVC library load modules. During

Option →	Pageable BLDL	Fixed BLDL	Pageable RSVC	Fixed RSVC	Pageable RAM	Fixed RAM	Fixed RERP
How to specify at SYSGEN	Defaulted	CTRLPROG OPTIONS = BLDL	CTRLPROG RESIDNT = TRSVC	CTRLPROG RESIDNT = TRSVC	Defaulted	Defaulted	CTRLPROG RESIDNT = ERP
Standard list associated with the option	IEABLD00	IEABLD00	IEARSV00	IEARSV01	IEAIGG00 IEAIGG02	IEAIGG01 IEAIGG03	IEAIGE00
How to specify at IPL	BLDL = ____	BLDLF = ____	RSVC = ____	RSVCF = ____	RAM = ____	RAMF = ____	RERP = ____
May be specified at IPL whether or not specified at SYSGEN	Yes	Yes	No	No	Yes	Yes	No
Number of lists which may be specified	2	2	1	1	4	4	1
Names on the list	SVC or link library rouroutines	SVC or link library rouroutines	Type 3 and 4 SVC routines	Type 3 and 4 SVC routines	Access method and reenterable link library routines	Access method and reenterable link library routines	Error recovery procedure routines
Library of residence	SYS1.SVCLIB SYS1.LINKLIB	SYS1.SVCLIB SYS1.LINKLIB	SYS1.SVCLIB	SYS1.SVCLIB	SYS1.SVCLIB SYS1.LINKLIB	SYS1.SVCLIB SYS1.LINKLIB	SYS1.SVCLIB

Figure RRO 1. Resident Routines Options

execution of the BLDL operation in the macro instruction routines, the library directory is searched only when the directory entry for the requested load module is not present in the resident BLDL table.

You list, in a member of SYS1.PARMLIB, the names of those linkage or SVC library load modules whose directory entries are to be made resident. 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 be in ascending collating sequence. Creation of parameter library lists is discussed later in this section. The next part of this section provides guidelines for choosing the content of the list.

Notes:

1. Directory entries in the resident 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 operation takes place and the new directory entry for the module is made resident.
2. BLDL and BLDLF are mutually exclusive. By specifying either one during system initialization, the other is overridden.

Selecting Entries for the Resident BLDL Table

Any load module in the linkage or SVC library may have its directory entry placed in the resident BLDL table. Other items you should consider are:

1. Table size (probably only a factor if BLDLF is used).
   Linkage library — Each entry requires 40 bytes.
   SVC library – Each entry requires 32 bytes.
2. Frequency of use of the load module.

Table Size

The BLDL table, when BLDL is specified, occupies pageable storage (virtual storage) and the table size is usually not a problem. The BLDL table, when BLDLF is specified, occupies real storage and each installation should carefully consider how much real storage it can afford to dedicate for this function.

Frequency of Use

Since fixed resident routines reduce the amount of real storage available for paging, your installation's workload should be carefully considered before selecting the BLDLF option. The BLDL option, on the other hand, will always be beneficial for any frequently-used load modules.

For Link Library Lists: The scheduler, linkage editor, and language processor(s) are possible selections for link library lists.

For SVC Library Lists: In general, use any module from the SVC library you would consider for residence (for example, the RAM option). You should *not* create lists for the following because they are not necessary:

- Load 1 of type 3 and 4 SVCS (that is, IGC00xxx).
- Modules selected for RAM, RERP, RSVC, RAMF, or RSVCF usage.

Recommended modules should be chosen from access methods and ERPS. You should *always avoid* placing the following modules in the BLDL list because they have internal BLDL tables and internal directory entries: OPEN, CLOSE, TCLOSE, EOV, FEOV, SCRATCH, ALLOCATE, IEHATLAS, SETPRT, STOW, machine-check handler modules.

You can put the SVC library list in SYS1.PARMLIB using the member name IEABLDnn. This nn will be picked up when the operator specifies the system parameters with the response BLDL=xx,nn.

The code to support the resident BLDL table option is standard in VS1.

List IEABLD00

The IBM supplied standard list IEABLD00 is:

```
SYS1.LINKLIB   DEVMASKT,GO,HEWLOADR,IEEVRCTL,IEEVSTAR,IEFALRET,      X
               IEFIRC,IEFIRET,IEFSDA66,IEFSD060,IEFSD065,IEFSD161,   X
               IEFSD162,IEFSD263,IEFSD510,IEFV4221,IEFWC000,IEFWD000, X
               IEFW21SD,IEFW42SD,IEFX5000,IEWL,IEWLDRGO,IEWLOAD,     X
               IEWLOADR,IEWSZOVR,LINKEDIT
```

Resident Reenterable Modules Options

These options make it possible to pre-load reenterable modules into storage. These two options (the resident access method modules option and the resident link library modules option) use similarly named load lists (IEAIGG..) and share an operator reply (RAM= and RAMF=) at initialization time to refer to their separate lists.

The system has standard resident access method lists (IEAIGG00 and IEAIGG01) and standard resident link library module lists (IEAIGG02 and IEAIGG03) which are used unless you have changed them or ask for the use of other lists in the operator reply to message IEA101A.

To use both access method modules and link library modules options, the operator reply is RAM and/or RAMF=kk,ll,mm,nn. Each pair of letters represents a pair of alphanumeric characters (like 01) that identify a list of either access method or link library modules.

These options place reenterable load modules in storage and create a resident list of the modules. A LOAD, LINK, XCTL, or ATTACH macro instruction requesting any module first scans the resident list. If the module is listed, no fetch operation is required.

You may list, in a member of SYS1.PARMLIB, the load module names of modules to be made resident. Standard lists of most frequently used modules are supplied by IBM. The content of the standard IBM-supplied lists is tabulated at the end of this description. If you desire your modules to be put into the IBM standard lists IEAIGG00-03, you may use the sysgen LINKLIB and SVCLIB macros.

The Resident Access Method Modules Option

The storage space required for each access method module consists of the byte requirements of the module and its associated load request block (LRB). The *Storage Estimates* publication provides the byte requirements for access method modules eligible to be made resident. The code supporting the option is standard in VS1.

Most access method modules placed in storage are "only loadable". ATTACH, LINK, and XCTL macro instructions must not refer to such modules.

You may alter the standard access method list (or create alternative lists) to include other access method modules.

For example, if checkpoint/restart is used, the following access method routines must be resident:

- IGG019C1, IGG019C2, and IGG019C3 if track overflow is used to write the data set.
- IGG019HT (a page fix appendage) if virtual storage is specified for the task.

When a composite console is used, an alternative list should include BSAM modules for card readers and printers.

If you specify either the 3330 or the 2305 I/O devices in your system, add the following modules to the standard RAM list (IEAIGG00):

IGG019C4, IGG019FN, IGG019FP, and IGG019EK
IGG019C0 must also be resident and is on the standard RAM list.

If you use the SAM 'search direct' option, you are strongly advised to make IGG019FN, IGG019FP, and IGG019C4 resident through the standard RAM list. Performance is improved and required partition size is decreased if these modules are resident.

To be eligible for use with the resident access method option, access method load modules must be reenterable.

List IEAIGG00

The content of the IBM-supplied standard list IEAIGG00 is:

Module Name	*Access Method*	*Function*
IGG019FP	SAM	Channel end (appendage), search-direct
IGG019AN	QSAM (SB)	Backward move—format F, FB, U records
IGG019AM	QSAM (SB)	Backward locate—format F, FB, U records
IGG019BE	BSAM	Magnetic tape forward space or backspace
IGG019AG	QSAM (SB)	GET move with CNTRL—format V records (card reader)
IGG019AK	QSAM (SB)	PUT move, format F, FB, U records
IGG019AJ	QSAM (SB)	PUT locate, format V, VB records
IGG019AB	QSAM (SB)	GET locate, format V, VB records
IGG019AL	QSAM (SB)	PUT move, format V, VB records
IGG019AD	QSAM (SB)	GET move, format V, VB records
IGG019BD	BSAM	NOTE/POINT tape
IGG019AC	QSAM (SB)	GET move, format F, FB, U records
IGG019AV	QSAM (SB)	PUT locate for dummy data set
IGG019AA	QSAM (SB)	GET locate, format F, FB, U records
IGG019CJ	SAM	Read length check, format V records (appendage)
IGG019C0	SAM	Channel end (format U)
IGG019C4	SAM	End-of-extent (appendage-find new extent)
IGG019EK	SAM	RPS, SIO, CE, and XE appendages
IGG019FN	SAM	SIO and page-fix (appendage)—RPS
IGG019DJ	QSAM (SB)	GET/PUT/PUTX JES compatibility interface processing
IGG019AQ	QSAM (SB)	GET synchronization routine
IGG019BC	BSAM	NOTE/POINT disk
IGG019DK	BSAM	READ/WRITE/CHECK JES compatibility interface processing
IGG019AI	QSAM (SB)	PUT locate, format F, FB, U records
IGG019AR	QSAM (SB)	PUT synchronization routine
IFGAAABA	SAM	Verify GET/PUT/POINT requests by JES compatibility interface modules
IGG019CI	SAM	Length check, format FB records (appendage)
IGG019BB	BSAM	CHECK (all devices)
IGG019CC	SAM	Schedules I/O for tape, direct access
IGG019BA	BSAM	READ/WRITE (all devices)
IGG019CH	SAM	End-of-extent check (data extent block) (appendage)
IGG019HT	SAM	SIO and page-fix (appendage)
IGG019CD	SAM	Schedules I/O for direct access output

SB=simple buffering
SAM=common sequential access method routines

If the system generation statements specify the use of both MCS and of an IBM 2740 Communication Terminal as an operator's console, list IEAIGG00 is extended by adding the following module names:

IGG019MA	BTAM Read/write module
IGG019MB	BTAM Appendage
IGG019M0	BTAM 2740 module
IGG019MR	BTAM Online Test Control module

If the system generation statements specify that the IBM 3211 Printer is to be included in the system, the following modules are added to the list:

IGG019FR
IGG019FS
IGG019FT

Modules specified with sysgen macro SVCLIB VIRTUAL= , whose names begin with characters other than IGC, IGX, or IGE are also put on list IEAIGG00.

List IEAIGG01

The IBM-supplied standard list IEAIGG01 contains modules specified with the sysgen macro SVCLIB RESIDNT=, whose names begin with characters other than IGC, IGX, or IGE. If there are no module names for list IEAIGG01, it is created as a null member (it exists on SYS1.PARMLIB, but has no entries).

Resident Link Library Modules Option

The storage space required for each link library module consists of the byte requirements of the module and its associated loaded program request block (LPRB). The code supporting the option is standard in VS1.

To utilize the option in your system:

- Add a list or lists of names of reenterable modules to be preloaded, to the parameter library. Each module name must be followed by its alias names (separated by commas).
- Have the operator specify your list or lists in his RAM= or RAMF= reply at IPL time.

  Note: Small (real storage) users should not use RAMF. This further reduces real storage space for paging.
- As an alternative, you can have your modules put on the standard lists IEAIGG02 and 03. A description of these lists follows.

List IEAIGG02

The IBM-supplied standard list IEAIGG02 contains modules specified with the sysgen macro LINKLIB VIRTUAL=. If there are no module names for list IEAIGG02, it is created as a null member (it exists on SYS1.PARMLIB, but has no entries).

List IEAIGG03

The IBM-supplied standard list IEAIGG03 contains modules specified with the sysgen macro LINKLIB RESIDENT=. If there are no module names for list IEAIGG03, it is created as a null member (it exists on SYS1.PARMLIB, but has no entries).

The Resident SVC Routines Option

This option makes any of the type 3 and 4 SVC routine load modules resident in storage. Some, or all, of the modules associated with a SVC service routine may be made resident. Placing the most frequently used SVC load modules of a system service routine, such as OPEN, in storage improves system performance. For type 3 SVC load modules and initial type 4 SVC load modules, the SVC table entries associated with these modules are adjusted to reflect an entry point address rather than a relative track address. A resident SVC load list is used by the XCTL macro instruction for transfer of control between resident type 4 SVC load modules.

You list, in a member of SYS1.PARMLIB, the type 3 and 4 SVC load modules to be made resident. The member names for the standard lists are IEARSV00 and 01. Such standard lists (shown later) are built by IBM in SYS1.PARMLIB of the generated system. The creation of parameter library lists is discussed later in this chapter.

If your system includes the multiple console support (MCS) function, to improve MCS performance you should add to the standard list IGC0007B, the name of the first load module of the SVC 72 routine.

Note: Small (real storage) users should not use RSVCF. This further reduces real storage space available for paging.

Storage Requirements

The *VS1 Storage Estimates* publication provides the byte requirements of type 3 and 4 SVC routines eligible to be made resident. The byte requirement of the code supporting the option is also provided.

List IEARSV00

The content of the IBM supplied standard list IEARSV00 is:

Module Name	*Function*
IGG0191L	Open—Access method executor
IGG0199L	Open—Access method executor
IGC0001C	ABEND—Control module
IGC0007B	Router and control module
IGG0CLC2	Catalog—Locate processing
IGG0CLC7	Catalog—Third load of update, exit processing, and error handling
IGG0CLF2	Catalog—SYSCTLG and BPAM directory formatter
IGC0003B	Allocate—Initialization
IFG0202G	Close—Tape volume disposition function
IGG0193G	Load required BDAM module
IGC3503D	DISPLAY/MONITOR processor
IGG0193E	Open executor number 3
IGG0203A	BDAM executor
IGC0203E	WTP routine
IGG0193A	Open executor number 1
IGG0193C	Open executor number 2
IFG0197A	Open—Access method executor return function
IFG0200X	Close—Access method executor
IGC0E01C	ABEND—Subtask processing
IGG0201N	RCI close intercept
IFG0195C	Open—No tape label positioning function
IFG0195K	Open—Standard label INPUT/MOD header label 2 function
IFG0195H	Open—Standard label INPUT/MOD header label 1 and 2 function
IGC1203D	Reply processor routine (MCS)
IGC0403D	Command routine
IFG0200Z	Close—Tape standard trailer label function
IFG0196T	Open—Standard label header label writing
IFG0196Q	Open—Tape standard label date protection
IFG0196N	Open—Standard label output security function
IFG0194H	Open—Tape volume verification function
IGC0003E	WTO/WTOR/WTP (SVC 35 processor)
IGC0103E	WTOR processor
IGC0201F	Purge (SVC 16—third load)
IFG0552R	EOV—Tape input standard trailer label and volume disposition functions
IFG0195B	Open—Standard label position function; INPUT/MOD header label 1 function
IFG0194F	Open—Tape mount verification function
IFG0193B	Open—Tape initial common function
IFG0202F	Close—Tape volume disposition function
IGG0201X	Close—Access method executor
IGG0201A	Close—Access method executor
IGG0191G	Open—Access method executor
IFG0194I	Open—Tape final common function
IGG01993	Open—Access method executor
IGG01991	Open—IOB and buffer construction
IGG01915	Open—Access method executor
IGG0290D	Scratch—Format 4 DSCB (VTOC) updating
IGG0290C	Scratch—Format 5 DSCB (free space) updating
IGG0290B	Scratch—Format 6 DSCB updating, split-cylinder data sets
IGG0299A	Scratch—DSCB removal for formats 1, 2, and 3
IGG0290A	Scratch—Password protection interface, VTOC search
IGG0290F	Scratch—Volume mounting and verification
IGG0290E	Scratch—Mount message building
IGG0325H	Allocate (non-ISAM)—Updating format 4 DSCB and error handling
IGG0325G	Allocate (non-ISAM)—Update format 5 DSCB
IGG0325E	Allocate (non-ISAM)—Build format 1 and 3 DSCBs, non-split cylinder data sets
IGG0325D	Allocate (non-ISAM)—Search free space
IGG0325B	Allocate (non-ISAM)—Request conversion and type determination
IGG0325A	Allocate—Duplicate format 1 DSCB search
IGG029R1	RPS—Set up module
IGC0003D	Chain manipulator
IGG0191C	Open—Access method executor
IGG0193I	Open—Access method executor
IGG0191I	Open—Access method executor
IGG01911	Open—Access method executor

Module Name	*Function*
IGG0191J	Open—Access method executor
IFG0232Z	TCLOSE—Direct access final function
IFG0232D	TCLOSE—Direct access input and output functions
IGG0191O	Open—Access method executor
ICC0G01C	Alternate entry point for ABEND control module
IGC0B01C	ABEND—WTO purge processing
IFG0201R	Close—Direct access write DSCB and output user labels function
IGG0191D	Open—Direct access executor
IGG0201W	Close—Access method executor
IFG0553P	EOV—Direct access input initial function
IGG0191F	Open—Access method executor
IGG0199W	Open—Access method executor
IGG0193K	JES open executor function
IGG0199G	Open—Access method executor
IFG0551H	EOV—Initialize work area and determine device type
IGG0203K	JES close executor function
IGG0201Y	Close—Release work areas and buffers
IGG0201Z	Close—SAM executor
IGG01917	Open—Second load of load executor
IGG01910	Open—Load executor (first load)
IGG0198L	Open—Access method executor
IGG0196B	Open—Main executor (second load)
IGG0191B	Open—Main executor (first load)
IFG0552X	EOV—Direct access input concatenation/end-of-data function
IFG0551F	EOV—Initial read JFCB function
IFG0202E	Close—Write file mark
IGG0196A	Open—DEB construction (second load)
IGG0191A	Open—DEB construction (first load)
IGG0191N	Open—Access method executor
IFG0195J	Open—DSCB to JFCB merge (direct access)
IFG0195A	Open—DSCB to JFCB merge (direct access)
IFG0194E	Open—Unit selection and DSCB read
IFG0196M	Open—Merge DCB to JFCB
IFG0196L	Open—Merge and DCB exit routine
IFG0196J	Open—JFCB to DCB merge
IGC0107B	1052 processor module
IFG0202J	Close—Restore system function
IFG0202K	Close—Restore user function
IFG0196X	Open (final)—JFCB to DSCB merge, SYSOUT limit, write JFCB functions; EXCP appendages
IFG0200Y	Close—Access method interface and write DSCB
IGG0200G	Close—Access method executor return/interface
IFG0202L	Close—Final load
IFG0200W	Close—Access method interface
IFG0200V	Close—Initialization and read JFCB and DSCB
IGC0002*	Close—Initial load
IGC0001F	Purge routine
IGC0001I	Open—Initial load
IGC0005E	EOV—Initial load
IGG0196I	Open—Access method executor
IFG0193A	Open—Volume serial function
IFG0196V	Open—Access method determination
IFG0198N	Open—Rewrite JFCB and final load
IFG0196W	Open—Access method executor
IGG0190S	Open—Access method executor

*The last (eighth) character is a 12 and 0 punch. This character has no assigned graphic in EBCDIC. In BCD, the graphic is ? (the question mark).

Also, modules specified with the sysgen macro SVCLIB VIRTUAL= , whose names begin with IGC or IGX, are put on list IEARSV00.

List IEARSV01

The IBM-supplied standard list IEARSV01 contains modules specified with the sysgen macro SVCLIB RESIDNT= , whose names begin with IGC or IGX. If there are no module names for list IEARSV01, it is created as a null member (it exists on SYS1.PARMLIB, but has no entries).

The Resident Error Recovery Procedure Option

This option places error recovery procedures in fixed storage. Some, or all, of the modules associated with the handling of an I/O error may be made resident. If an I/O device frequently requires ERP processing, system performance improves if the error recovery procedures are made resident. The list of those error recovery procedures that may be made resident is contained in the *Storage Estimates* publication. An I/O supervisor request for an error recovery procedure will result in a search of the resident error recovery procedure list. If the error recovery procedure is resident, no fetch operation is required.

You list, in a member of SYS1.PARMLIB, the module names of error recovery procedures to be made resident. The member name for the standard list is IEAIGE00. The error recovery procedures should be listed by expected frequency of use; the least used module is first in the list. *Note*: The format of the IBM-supplied IEAIGE00 list contains the required library name, SYS1.SVCLIB, and *no* error recovery procedure names, unless the user specifies the names of ERPs on the sysgen macro SVCLIB. After system generation, IEAIGE00 can be updated to indicate which error recovery procedures are to be made resident or an alternate list can be created. The creation of parameter library lists is discussed later in this section.

Storage Requirements

The *VS1 Storage Estimates* publication provides the byte requirements of error recovery procedures that may be made resident. The byte requirement of the code supporting the option is also provided.

List IEAIGE00

The IBM-supplied standard list IEAIGE00 contains modules specified with the sysgen macro SVCLIB RESIDNT= or VIRTUAL=, whose names begin with IGE. If there are no module names for list IEAIGE00, it is created as a null member (it exists on SYS1.PARMLIB, but has no entries).

Creating Parameter Library Lists

Use the IEBUPDTE utility program to construct the required lists of load module names in the parameter library. Standard member names for these lists are shown in Figure RRO 1, plus LNKLST00 for the link library list option.

These are the member names that the nucleus initialization program reads from SYS1.PARMLIB if the option was either specified or defaulted to at sysgen and neither canceled nor modified by the operator's response to message IEA101A.

Note: The nucleus initialization program (NIP) will search 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 operator will receive message IEA211I "OBTAIN FAILED FOR SYS1.PARMLIB DATA SET". Message IEA208I "fff FUNCTION INOPERATIVE" will follow. The fff parameter–RAM, BLDL, RSVC, or RERP–shows which of the functions cannot be implemented. Processing will continue; however, any resident functions dependent on parameter lists contained in the parameter library will be omitted from the system nucleus.

Except for LNKLST00, your input format (to IEBUPDTE) for the lists is the same for all options, consisting of library identification followed by the load module names. You use 80-character records with the initial or only record containing the library identification. Continuation is indicated by placing a comma after the last name in a record and a nonblank character in column 72. Subsequent records must start in column 16.

The initial record format (with continuation) is:

```
 1                                                                    72
                SYS1.LINKLIB
[b . . .]       SYS1.SVCLIB              b . . . name1,name2,name3, . . . X
```

Subsequent records do not contain the library name.
SYS1.LINKLIB indicates that linkage library load module names follow.
SYS1.SVCLIB indicates that SVC library module names follow.

You may also construct alternative lists and place them in the parameter library. Member names for these alternative lists are of the form:

IEABLDxx for the BLDL option
IEAIGGxx for the resident access method option
IEARSVxx for the resident SVC routine option
IEAIGExx for the resident error recovery procedure option
LNKLST00 for the link library list option

where xx can be any two alphameric characters.

Use of the alternative lists is indicated by the operator at initialization time. The operator may indicate that the standard list is to be used; that alternative lists are to be used; or that the option(s) will not be used. In the last case, no resident BLDL table, access method routines except several standard ones, SVC routines or error recovery procedures are made resident.

Example

The following coding illustrates the format and content of a BLDL option list that might be used to support the resident BLDL table option. The operator, at IPL time, would have to indicate the member name, IEABLDAE to the system. The load module names listed are from the assembler (E), linkage editor, and scheduler components of the operating system. Note that the module names are listed in ascending collating sequence as required for the resident BLDL option. Resident access method or SVC modules should be listed in order of anticipated frequency of use.

```
//BLDLIST EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD DSNAME=SYS1.PARMLIB,DISP=OLD
//SYSIN DD *
./        ADD        NAME=IEABLDAE,LIST=ALL
./        NUMBER     NEW1=01,INCR=02
SYS1.LINKLIB  GO,IEEGESTO,IEEGK1GM,IEEICIPE,IEEIC2NQ,IEEIC3JF,          X
              IEEQOT00,IEFINTQS,IEFK1,IEFSD008,IEFW21SD,IEFXA,          X
              IETASM,IETDI,IETE1,IETE2,IETE2A,IETE3,IETE3A,IETE4M,      X
              IETE4P,IETE4S,IETE5,IETE5A,IETE5E,IETE5P,IETINP,IETMAC,   X
              IETPP,IETRTA,IETRTB,IET07,IET071,IET08,IET09,IET09I,      X
              IET10,IET10B,IET21A,IET21B,IET21C,IET21D, IEWL,IEWSZOVR
./        ENDUP
/*
```

Note: During initialization the operator reply "L" may be used in conjunction with a list specification and causes the content of the list to be printed. You should use this feature initially (especially with extensive lists) to easily identify format errors; for example, a 9 character name, or incorrect name specifications.

Example of the ERP Option List

The following coding illustrates the format and content of an ERP option list that may be used to support the resident ERP option. The operator, at IPL time, would have to indicate the member name, IEAIGE01, to the system. The load module names listed are the optical reader ERPS, write-to-operator, statistics update, I/O purge, OBR and SDR/CCR modules. The system must be sysgened with the RERP option to load any list.

```
//ERPLIST EXEC PGM=IEBUPDTE,PARM=NEW
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD DSNAME=SYS1.PARMLIB,DISP=OLD
//SYSIN DD *
./      ADD        NAME=IEAIGE01,LIST=ALL
./      NUMBER     NEW1=01,INCR=02
 SYS1.SVCLIB       IGE0011B,IGE0011C,IGE0011D,                     X
                   IGE0025C,IGE0125C,IGE0225C,                     X
                   IGE0025D,IGE0025E,IGE0025F,                     X
                   IGE0125F,IGE0525F
./      ENDUP
/*
```

Link Library List Feature

The link library list (LNKLST00) enables you to concatenate up to 16 data sets, on multiple volumes, to form SYS1.LINKLIB. LNKLST00 is included in the system when it is generated as a required member of SYS1.PARMLIB. If SYS1.PARMLIB does not include the member LNKLST00, SYS1.LINKLIB is used as the system link library and a warning message is provided.

Note: The amount of space required for SYS1.PARMLIB is discussed in the *VS1 Storage Estimates* publication.

LNKLST00 contains one member, SYS1.LINKLIB. After system generation you will have the option of adding members via the IEBUPDTE utility program. Each member may have up to 16 extents. After making additions to SYS1.SVCLIB, SYS1.LINKLIB, or data sets concatenated to LINKLIB via LNKLST00, and before using the additions, IPL should be performed to update the description of the link and/or SVC library in storage.

Your input format (to IEBUPDTE) consists of 80 character records. Continuation is indicated by placing a comma after the last name in a record and a nonblank character in column 72. Subsequent records must start in column 16. The initial format is:

```
[b . . . ] SYS1.LINKLIB
```

To add member names to LNKLST00, replace the initial record with:

```
[b . . . ] SYS1.LINKLIB,name1,name2,name3, . . .
```

The appropriate *OS/VS Message Library* publication describes the NIP messages associated with LNKLST00.

Output Separation

In vs1, the system output writer can use the output separator facility to write separation records prior to writing the output of each job and optionally for printer-destined output, at the conclusion of writing the output. 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.

This section describes the output separator that is supplied by IBM, and tells how to write your own.

Section Outline

Output Separation

In VS1, both the system output writer and the direct SYSOUT 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 end-of-job separator option, available only under the system output writer, assures the programmer that he has received all the output from his job. The separation records are written upon conclusion of writing a job's output.

You can use the output separator that is supplied by IBM, or you can create and use your own output separator programs.

Note: User-written output separator routines are not supported to RTAM devices.

Using an Output Separator

The output separator function operates under control of both the system output writer and the direct SYSOUT writer. To use the function, the separator program must reside in the link library (SYS1.LINKLIB), and its name must be included as a parameter in either of the output writer procedures (the second part of the PARM field in the EXEC statement). Cataloged procedures for both writers are fully described in the *System Reader, Initiator, and Writer Cataloged Procedures* section. If the separator function is selected, the end-of-job separator option may be specified by placing the number of trailing separators to be written in the parameter field of the writer procedure.

Functions of the IBM Output Separator

The IBM-supplied output separator resides in the link library (SYS1.LINKLIB). When its name, IEFOSC06, is specified as a parameter in an output writer cataloged procedure, that output writer uses it to separate job output. The type of separation provided by the separator depends on whether the output is punch-destined or printer-destined.

Punch-Destined Output

For punch-destined output, the IBM-supplied separator provides one to three specially punched cards (deposited in stacker 1) prior to the punched card output of each job (see the description of the PARM values of the EXEC statement in the section *Output Writer Procedures*). Each of these separator cards is punched in the following format:

Columns 1 to 34	– blanks
Columns 35 to 42	– jobname
Columns 43 to 44	– blanks
Column 45	– output classname
Columns 46 to 80	– blanks

Note: No end of job separators are available for punch-destined output.

Printer-Destined Output

For printer-destined output, the IBM-supplied separator provides one to three specially printed pages prior to printing the output of each job (see the description of the PARM values of the EXEC statement in the section *Output Writer Procedures*). 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 preceding, 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 1 to 3 separator pages, and on the fold following the last page. This feature provides easy separation of job output in a stack of printed pages.

For printer-destined output with the IBM-supplied separator, you must include a channel 9 punch 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, you must also offset the printer registration.

End-of-job separators provided by the IBM-supplied routine are identical to those printed prior to the output, except that no asterisks are printed after the last page. For remote devices that do not support channel 9 (for example, the IBM 2780), the output separator consists only of block letters (the line of asterisks is not printed).

Creating an Output Separator Program

You can write your own output separator program by using the information provided by either output writer and by conforming to the requirements explained in the following text. Your separator program, when added to the link library (SYS1.LINKLIB), is invoked by specifying its name as a parameter in the EXEC statement of the output writer cataloged procedure.

Parameter List

Either output writer provides your separator program with a 4-word parameter list of needed information. When your 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. Byte 1 contains the number of separator pages or cards (in binary) requested, and bytes 2 and 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 your separator program uses to determine the type of output unit. The first bit to the left is set to 1 if the output unit is a 1442 punch device. 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:

111	1442 punch device
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
xxx . 1 . xx	channel 9 not supported on this device
xxx 1	DSO
xxx . . . 10	Separator routine entered *after* writing 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 your 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 your separator program.

Programming Considerations

If you are using the (asynchronous) system output writer, your separator program, if specified in the output writer cataloged procedure, is brought in by a LINK macro instruction issued from module IEFOSC01 or IEFOSC02 of the output writer. Your separator program may be any size, but space must be provided at sysgen or overridden at IPL with a PARMLIB entry (see the appropriate *VS1 SYSGEN* publication for more details). If your job falls into a job class using the (synchronous) direct SYSOUT writer, your separator program (if specified in the procedure) is brought into virtual storage by use of a LOAD macro instruction. After performing separation on all devices required for the SYSOUT data sets in that step, the program is released by means of a DELETE macro instruction.

CAUTION

Since the separator program operates with the supervisor protection key, but in the program mode, your separator program must insure data protection during its execution.

When writing a separator program, you must observe the following programming requirements:

- Your program must conform to the standard linkage conventions. 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 your program receives control, the address of a standard save area is in register 13.
- Your 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.)
- Your 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 your error exit routine in case an uncorrectable I/O error occurs while writing your program's output.
- Your program should use the RETURN macro instruction to return control to the output writer. Before returning, your program must free any main storage it obtained during its operation, and your 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 your error exit routine is entered).

Output from the Separator Program

Your 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 your output, if desired. You can use an IBM-supplied routine that constructs block characters (explained later). You can punch as many separator cards or print as many separator pages as you deem necessary.

The output from your 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, ANSI, or none).

For printer-destined output, you can begin your separation records on the same page as the previous job output, or skip to any subsequent page. However, your 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 your separation records, your separator program should write sufficient blank records to force out the last separation record. This also allows your 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, your 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. Your 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. Your 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, your 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 you enter the routine, you must provide the address of a 4-word parameter list 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.

The Shared Direct-Access Device Option

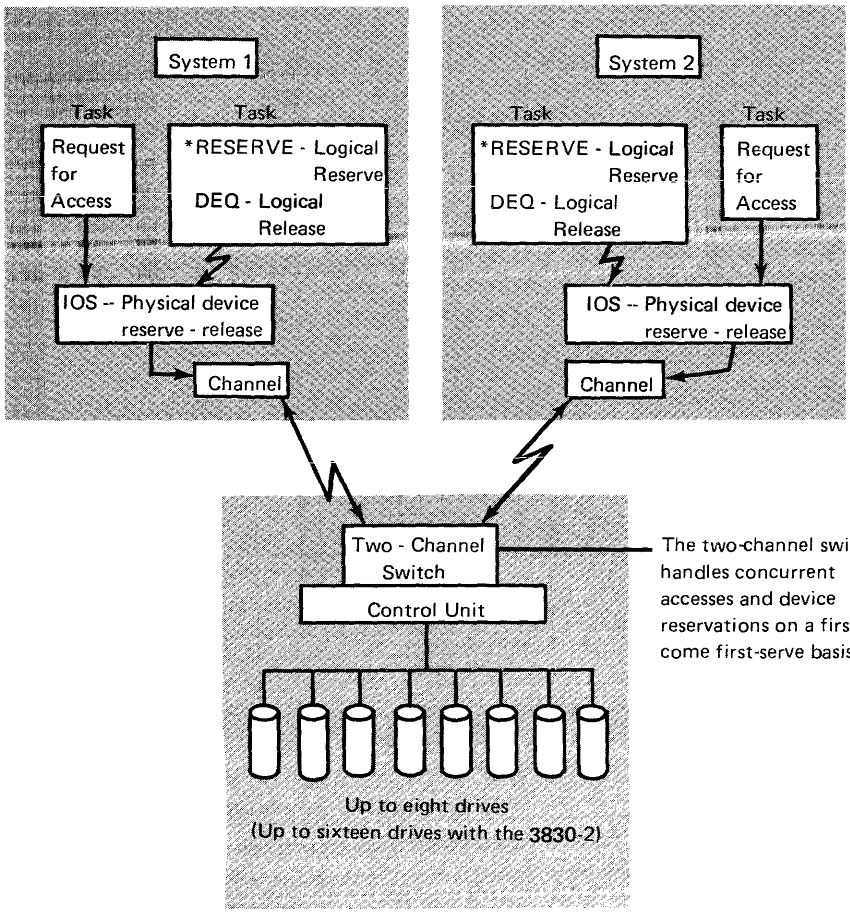
This section describes the Shared Direct Access Storage Device option (Shared DASD) of VS1. It describes the functions of the option, its operating environment, and volume acceptability. It also explains operating procedures and data set considerations that the systems programmer must be aware of in using the option. The section also describes a procedure for finding unit control block addresses necessary for using the RESERVE macro instruction: it also shows an assembler language subroutine that issues a RESERVE and can be called by a higher level language.

Section Outline

The Shared Direct-Access Device Option

The Shared DASD option 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. Briefly, the sharing is accomplished by a two-channel or four-channel (3330 and 3333 only) switch which allows a shared control unit to be switched between channels from different systems. The switching is controlled by program use of the RESERVE macro instruction which reserves a shared device or volume for the use of one system until it is freed by the program's issuing a DEQ macro instruction. If a RESERVE macro instruction is used before the system in which the macro instruction is used has access to the shared device, the macro instruction will take effect only after the system gains access to the device.

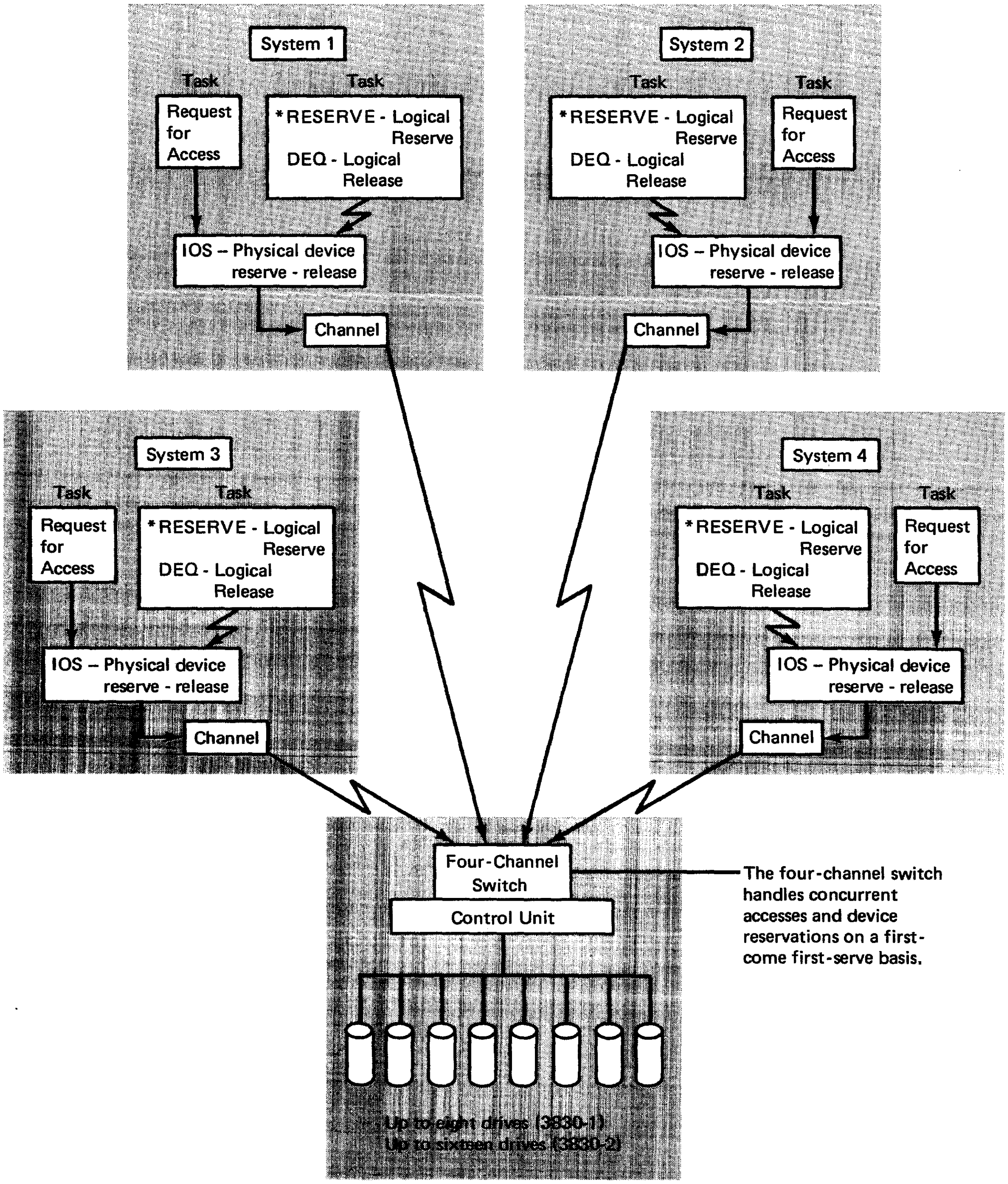
The Shared DASD facility can only be included in a system at system generation time. A two-channel switch facility is shown in Figure SHR 1. A four-channel switch facility is shown in Figure SHR 2.



The two-channel switch handles concurrent accesses and device reservations on a first-come first-serve basis.

* In multiprogramming systems, the RESERVE macro instruction also serializes use of the same resource between tasks in the system.

Figure SHR 1. General Shared DASD Environment


System 1
Task
Request for Access
Task
*RESERVE - Logical Reserve
DEQ - Logical Release
IOS – Physical device reserve - release
Channel
System 2
Task
*RESERVE - Logical Reserve
DEQ - Logical Release
Task
Request for Access
IOS – Physical device reserve - release
Channel
System 3
Task
Request for Access
Task
*RESERVE - Logical Reserve
DEQ - Logical Release
IOS – Physical device reserve - release
Channel
System 4
Task
*RESERVE - Logical Reserve
DEQ - Logical Release
Task
Request for Access
IOS – Physical device reserve - release
Channel
Four-Channel Switch
Control Unit
Up to eight drives (3830-1)
Up to sixteen drives (3830-2)

The four-channel switch handles concurrent accesses and device reservations on a first-come first-serve basis.

* In mulitprogramming systems, the RESERVE macro instruction also serializes use of the same resource between tasks in the system.

Figure SHR 2. General Shared DASD Environment, 4-Channel Switch

System Configuration

The Shared DASD option can be used with any combination of configurations of the operating system. Identical operating system configurations are not necessary for systems to share devices *unless* they share the system data set SYS1.LINKLIB. The option requires no additional equipment except the 2-channel switch, the 4-channel switch, or the IBM 2844 Auxiliary Storage Control unit, which does not require the 2-channel switch. Any of your installation's applications data sets can be shared; SYSCTLG can be shared when it does not reside on a systems residence volume. The following system data sets cannot be shared:

SYS1.SVCLIB
SYS1.NUCLEUS
SYS1.LOGREC
SYS1.SYSJOBQE
PASSWORD data set
SYS1.SYSPOOL
SYS1.SWADS
SYS1.PAGE
SYSCTLG (on system residence volume)
SYS1.ACCT
SYS1.MANX
SYS1.MANY

Devices that Can Be Shared

The following control units and devices are supported by the Shared DASD option:

1. IBM 2314 or 2319 Direct Access Storage Facility equipped with the 2-channel switch—IBM 2314 Disk Storage Module.
2. IBM 2314 Direct Access Storage Facility combined with the IBM 2844 Auxiliary Storage Control–IBM Disk Storage Module. Device reservation and release are supported by this combination with or without the presence of the 2-channel switch. Two channels–one from System A and one from System B–may be connected to the combination. In addition, the 2-channel switch may be installed in either or both of the control units, thus permitting as many as four systems to share the devices.
3. IBM 2835 Storage Control Unit with 2-channel switch–IBM 2305-2 Fixed Head Storage Facility.
4. IBM 3830-1 Storage Control Unit with 2-channel or 4-channel switch—IBM 3330 Disk Storage.
5. IBM 3830-2 Storage Control Unit with 2-channel or 4-channel switch—IBM 3333 Disk Storage and Control.

Alternate channels to a device from any one system may only be specified for the IBM 2314 Direct Access Storage Facility.

Volume/Device Status

The Shared DASD option requires that certain combinations of volume characteristics and device status be in effect for shared volumes or devices. One of the following combinations must be in effect for a volume or device:

System A	*Systems B,C,D*
1. Permanently resident	Permanently resident
2. Reserved	Reserved
3. Removable	Offline
4. 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 or detected if present.

Volume Handling

Volume handling on the Shared DASD option must be clearly defined since operator actions on the sharing systems must be performed in parallel. You should make sure that operators understand the following rules when the Shared DASD option is in effect:

1. Operators should initiate all shared volume mounting and dismounting operations. The system will dynamically allocate devices unless they are in reserved or permanently resident status. Only the former of the two can be changed by the operator.
2. Mounting and dismounting 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.
3. 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
   a. Specifying mount characteristics of shared devices in PRESRES (See the section *The PRESRES Volume Characteristics List.*)
   b. Varying all sharable devices off line prior to issuing start commands and then following parallel mount procedures described in the appropriate *Operator's Library* publication.

Sharing Application Data Sets

As indicated previously, all application data sets can be shared, but you must give special consideration to the classification of these data sets. It is recommended that you classify your shared data sets as read only or read/write. A read-only data set may be read by all sharing systems but is never updated by them. A read/write data set may be read or written—updated by all sharing systems. Read-only data sets are not reserved for the duration of their use; read/write data sets must be reserved for data set protection.

If a data set is seldom updated, but is read often, it is wise to classify it as read only. Minimizing reservation of devices will minimize the interference between systems.

A shared data set may be updated, effecting a device reservation for the write operation only, if the records being read are independent of each other. An example of such a data set with independent records is a private job library. Such a library may be reserved for the write operation only as long as members are not being deleted.

A system update time should be defined for updates to read-only data sets. For system update time the operator must vary offline, on all but one system, the device upon which the data set resides. Then the system update may be performed on the system to which that device is dedicated without any need to reserve the device. Processing of data sets by the linkage editor and utility programs constitutes update runs—the data sets they process are regarded as read/write data sets. You may want to prepare a routine that will issue a RESERVE macro instruction, invoke the program to be executed, and issue a DEQ macro instruction after program execution.

There is no protection for shared data sets across job steps. That is, the RESERVE and DEQ for a data set must be done within each step (task); if devices are still reserved at the end of a task, device release is effected. Therefore, it is possible for one system to reserve a device and update a data set on that device between the execution of two steps in the other systems which are using that data set. There is no guarantee that a data set will remain unchanged between execution of steps.

Reserving Devices

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 the RESERVE macro instruction must also use the DEQ macro instruction to release the device; two RESERVE instructions for the same resource without an intervening DEQ will result in an abnormal termination unless the second one 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 in all operating system configurations will release devices reserved by a terminating task.

The SMC Parameter of the ENQ Macro Instruction

The Set-Must-Complete (SMC) parameter available with the ENQ macro instruction may also be used with RESERVE; this parameter is discussed in the section *The Must-Complete Function.*

RESERVE Macro Instruction

The use of the RESERVE macro instruction is explained here:

```
[symbol]    RESERVE   (qname address, rname address,     [E] ,
                                                         [S]
                                        [       (TEST )]
            [rname length], SYSTEMS)    [, RET = {USE  }] ,UCB = pointer address
                                        [       (HAVE )]
```

qname

is the address in storage of an 8-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

is the address in 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
S

specify either exclusive control of the resource (E); or shared control with other tasks in the system (S). E is the default condition.

rname length

is 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, that is the resource is shared between systems.

RET=

specifies a conditional request for all of the resources named in the RESERVE macro instruction. If the operand is omitted, the request is unconditional. The types of conditional requests are as follows:

TEST

tests the availability status of the resources but does not request 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

specifie that control of the resources is requested only if a request has not been made previously for the same task.

Return codes are provided by the control program only if RET=TEST, RET=USE, or RET=HAVE is 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 (see the *Supervisor Services and Macros* publication).

UCB=*pointer address*

This keyword specifies either:

1. The address of a fullword that contains the address of the Unit Control Block (UCB) for the device to be reserved.
2. 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, you may wish to write an assembler language subroutine to issue the RESERVE macro instruction. You should pass to this subroutine the following information: *ddname, qname address, rname address, rname length,* and RET parameter.

The EXTRACT Macro Instruction

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. This section explains some procedures for finding the UCB address.

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 precaution 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(s).

Volume Assignment

Since exclusive control is by device, not by data set, you must 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 X_1 and X_2 reside on device X and data sets Y_1 and Y_2 reside on device Y. Task A in system A reserves device X in order to use data set X_1; task B in system B reserves device Y in order to use data set Y_1. Now task A in system A tries to reserve device Y in order to use data set Y_2 and task B in system B tries to reserve device X in order to use data set X_2. Neither can ever regain control and thus, will never complete normally and the job(s) should be canceled. In any environment in which job step time limits are specified, the task(s) 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, since the DADSM functions are transient SVCs, 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 macro instruction cannot complete the operation, therefore an interlock occurs. In this particular case, since no access to SVCLIB is possible, both systems will eventually enter an enabled wait state.

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 macro instruction. Prior to issuing an EXTRACT macro instruction, the user sets up an answer area in 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 issue the following form of the macro instruction:

```
EXTRACT answer-area address, FIELDS=TIOT
```

The address of the TIOT is then returned by the control program, right-adjusted, in the full word answer area.

The TIOT is constructed by job management routines and resides in 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 which he intends to issue the RESERVE macro instruction.

The UCB address may also be obtained via the DEB and 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.)

Following are suggested procedures for finding the UCB address of the device to be reserved.

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 in 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 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 EDAM data sets the user may reserve the device at any point in his 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 be 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 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 prime data and independent overflow areas reside. The following procedures may be used to achieve this:

1. Open the data set successfully.
2. Locate the actual device address (MBBCCHH) 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 highest level index not in core. This extent entry is also a pointer to the UCB address.
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 may be used by FORTRAN, COBOL, or assembler language programs to issue the RESERVE and DEQ macro instructions. Parameters that must be passed to the RESDEQ routine, if the RESERVE macro instruction is to be issued, are:

DDNAME
: Address of the eight character name of the DDCARD for the device that you wish to reserve.

QNAME
: Address of an 8-character name.

RNAME LENGTH
: Address of one byte (a binary integer) that contains the RNAME length value.

RNAME

Address of a name from 1 to 255 characters 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            PARAMETER LIST ADDRESS POINTER
           L        3,0(9)         ADDRESS OF PARAMETER LIST
           CLC      0(4,3),=F'0'   DDNAME PARAMETER OR WORD OF ZEROS
           BE       WANTDEQ        WORD OF ZEROS IF DEQ IS REQUESTED
*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     L        5,0(3)         ADDRESS OF DDNAME
           CLC      0(8,5),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(3)             ADDRESS OF QNAME
           MVC      QNAME(8),0(7)  MOVE IN QNAME
 *PROCESS FOR DETERMINING THE RNAME AND THE LENGTH OF RNAME
           L        7,8(3)             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(3)            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'
           END
```

System Macro Instructions

This section contains the description and formats of macro instructions that allow you either to modify control blocks or to obtain information from control blocks and system tables.

Section Outline

How to Read a Job File Control Block

To accomplish the functions that are performed as a result of an OPEN macro instruction, the OPEN routine requires access to information that you have supplied in a data definition (DD) statement. This information is stored by the system in a job file control block (JFCB).

Usually, the programmer is not concerned with the JFCB itself. In special applications, however, you may find it necessary to modify the contents of a JFCB before issuing an OPEN macro instruction. To assist you, the system provides the RDJFCB macro instruction. This macro instruction causes a specified JFCB to be read into main storage from the job queue in which it has been stored. Format and field description of the JFCB is contained in the *VS1 System Data Areas* publication.

When subsequently issuing the OPEN macro instruction, you must indicate, by specifying the TYPE=J option, that you have supplied a modified JFCB to be used during the initialization process.

The JFCB is returned to the job queue by the OPEN routine or the OPENJ routine, if any of the modifications in the following list occur. These modifications can occur only if the information is not originally in the JFCB.

- Expiration date field and creation date field merged into the JFCB from the DSCB.
- Secondary quantity field merged into the JFCB from the DSCB.
- DCB fields merged into the JFCB from the DSCB.
- DCB fields merged into the JFCB from the DCB.
- Volume serial number fields added to the JFCB.
- Data set sequence number field added to the JFCB.
- Number of volumes field added to the JFCB.

If you make these, or any other modifications, and you want the JFCB returned to the job queue, you must set the high-order bit of field JFCBMASK+4 to one. This field is in the JFCB. Setting the high-order bit of field JFCBMASK+4 to zero does not necessarily suppress the return of the JFCB to the job queue. If the OPEN or OPENJ routines have made any of the preceding modifications, the JFCB is returned to the job queue. To inhibit writing the JFCB back to the job queue during an OPENJ, the field JFCBTSDM should be set to X'08' prior to issuing the OPEN macro.

OPEN—Prepare the Data Control Block for Processing (S)

The OPEN macro instruction initializes one or more data control blocks so that their associated data sets can be processed.

A full explanation of the operands of the OPEN macro instruction is contained in the *OS/VS Data Management Macro Instructions* publication. The TYPE=J option, because it is used in conjunction with modifying a JFCB, should be used only by the system programmer or only under his supervision.

CIRB—Create IRB for Asynchronous Exit Processing

The CIRB macro instruction is included in SYS1.MACLIB and must be included in your system at system generation time if you intend to use it. The issuing of this macro instruction causes a supervisor routine (called the exit effector routine) to create an interruption request block (IRB). 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.

Name	Operation	Operand
[symbol]	CIRB	$\{EP=addrx\}$, KEY = $\left\{\begin{matrix}\underline{PP}\\ SUPR\end{matrix}\right\}$, MODE = $\left\{\begin{matrix}\underline{PP}\\ SUPR\end{matrix}\right\}$, [STAB = code,] $\left\{SVAREA = \begin{matrix}\underline{NO}\\ YES\end{matrix}\right\}$, [WKAREA = value]

EP

specifies the entry point address of the user's asynchronous exit routine.

KEY

specifies whether the user's asynchronous routine will operate with a CPU protection key established by the supervisory program (SUPR) or with a protection key obtained from the task control block of the task for which the macro instruction is issued (PP).

MODE

specifies whether the user asynchronous routine will be executed in the problem program (PP) state or in a supervisory (SUPR) state.

STAB

indicates the status condition of the interruption request block. The 'code' parameter may be either of the following:

(RE) to indicate that the IRB is reusable in its current form.

(DYN) to indicate that the storage area assigned to the IRB is to be made available (that is, freed) for other uses when the asynchronous exit routine is completed.

SVAREA

specifies whether a register save area (of 72 bytes) is to be obtained from the storage assigned to the problem program. If it is, the address of this save area is placed in the IRB. The asynchronous exit routine then follows the system register saving convention of using the SAVE and RETURN macro instructions. In this manner, a generalized subroutine can be used as an asynchronous exit routine.

WKAREA

specifies the number of double words (given as a decimal value) required for an area in which the routine issuing the macro instruction can construct interruption queue elements.

SYNCH—Synchronous Exits to Processing Program

The SYNCH macro instruction is a system macro instruction that permits control program supervisor call (SVC) routines to make synchronous exits to a processing program.

Name	Operation	Operand

[symbol]	SYNCH	{ entry-point }
		{ (15) }

entry-point

specifies the address of the entry point for the processing program that is to be given control.

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, you use the SYNCH macro instruction when a control program in the supervisor state is to give temporary control to a processing program routine, and you expect the processing program to return control to the supervisor state. The program to which control is given must be in 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.

STAE—Specify Task Asynchronous Exit

The STAE macro instruction permits control to be returned to a user exit routine when a task is scheduled for ABEND. When you issue the STAE macro instruction, a STAE control block (SCB) is created and initialized with the address of your user exit routine. If you issue multiple STAE requests within the same program, the SCB associated with the last issued STAE request becomes the active SCB: it will be the first to gain control when an ABEND is scheduled. If the active SCB is cancelled, the preceding SCB, if there is one, will become the active SCB.

Notes:
- You cannot cancel or overlay an SCB not created by your program.
- The execution of a LINK macro instruction does not cancel the active SCB for the program in control.

STAE—Execute and Standard Form

Name	Operation	Operand
[symbol]	STAE	{ 0 | exit address } , { OV | <u>CT</u> } [,PARAM = list address] [,XCTL = { YES | <u>NO</u> }] [,PURGE = { <u>QUIESCE</u> | HALT | NONE }] [,ASYNCH = { <u>NO</u> | YES }] MF = (E, [remote list address] [(1)])

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 last SCB created is canceled and the previously created SCB becomes current. The address may be loaded into one of the general registers (r_1) 2 through 12.

Note: If you use the execute form of the macro and specify a zero, the exit address in the parameter list will be zeroed.

OV

indicates that the parameters passed in this STAE macro instruction are to overlay the data currently in the SCB.

CT

indicates the creation of a new active SCB.

PARAM=

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 (r_2) 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.

PURGE=QUIESCE

indicates that all active input/output operations will be purged with the quiesce option. If this fails, active input/output operations will be purged with the halt option.

Note: If you use the execute form of the STAE macro instruction and omit the PURGE parameter, QUIESCE will not be the default; the option specified for the preceding use of STAE will be used.

PURGE=HALT

indicates that all active input/output operations will be purged with the halt option.

PURGE=NONE

indicates that all active input/output operations will not be purged

ASYNCH=NO

indicates that asynchronous exit processing will be prohibited while STAE exit processing is being done.

ASYNCH=YES

indicates that asynchronous exit processing will be allowed while STAE exit processing is being done.

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.

Note: When using the execute form of the STAE macro instruction and omitting the ASYNCH parameter, the option specified for the preceding use of STAE will be used.

STAE—List Form

Use the list form of the STAE macro instruction to construct program parameter lists. The description of the execute and standard form applies to the list form with the following exceptions:

exit address

any address that may be written in an A-type address constant.

MF=L

indicates the list form of the STAE macro instruction.

You should be aware of several conditions when you use the PURGE and ASYNCH parameters of the STAE macro instruction:

- If your user exit routine requests a supervisor service that requires asynchronous interruptions to complete its normal processing, you must specify ASYNCH=YES.
- You must specify ASYNCH=YES if you use an access method that requires asynchronous interruptions to complete its normal processing and you have specified PURGE=QUIESCE.
- If you are using the Indexed Sequential Access Method (ISAM) and specify PURGE=HALT, only the I/O event for which the PURGE is done will be posted. Subsequent ECBS will not be posted; this causes the ISAM CHECK routine to treat purged input/output operations as waiting input/output operations and you will never get past the CHECK in your program.
- You must specify ASYNCH=YES when you have the following combination of conditions: an access method that requires asynchronous interruptions to complete its normal processing, a specification of PURGE=NONE, and a request of CHECK in your user exit routine.

- If you specify PURGE=HALT and an ISAM data set is being updated when a failure occurs, part of the data set may be destroyed.
- If quiesced input/output operations are not restored and you are using ISAM, the ISAM CHECK routine will treat purged input/output operations as waiting input/output operations and part of the ISAM data set may be destroyed if it is being updated when a failure occurs.
- If input/output operations are allowed to complete while your exit routine is in progress and there is a failure in the I/O processing, you will encounter an ABEND recursion when the I/O interrupt occurs. This can be misleading because it will appear that your exit routine failed while the actual cause of the failure was in the I/O processing.

Programming Notes

When control is returned to the user after the STAE macro instruction has been issued, register 15 contains one of the following return codes:

Code	*Meaning*
00	An SCB is successfully created, overlaid, or canceled.
04	Storage for an SCB is not available.
08	The user is attempting to cancel or overlay a non-existent SCB, or is issuing a STAE in his STAE exit routine.
0C	The exit routine or parameter list address is invalid.
10	The user is attempting to cancel or overlay an SCB not associated with his level of control.

When a program with an active STAE environment encounters an ABEND situation, control is returned to the user through the ABEND/STAE interface routine at the STAE exit routine address. The register contents are as follows:

- *Register 0:*

Code	*Indication*
0	Active I/O at time of ABEND was quiesced and is restorable.
4	Active I/O at time of ABEND was halted and is not restorable.
8	No I/O was active at the time of the ABEND.

- *Register 1:* Address of a 104-byte work area:

0	STAE exit routine parameter list addr or 0	ABEND completion code
8	PSW at time of ABEND	
16	Last P/P PSW before ABEND	
24	Registers 0-15 at time of ABEND (64 bytes)	

If problem program issued STAE:

88	Name of ABENDing program or 0	
96	Entry point addr of ABENDing program	0

If supervisor program issued STAE:

88	Request block address of ABENDing program	0
96	0	

- *Registers 2-12:* Unpredictable.
- *Register 13:* Address of a supervisor save area.
- *Register 14:* Address of an SVC 3 instruction.
- *Register 15:* Address of the STAE exit routine.

Registers 13 and 14, if used by the STAE exit routine, must be saved and restored prior to returning to the calling program. Standard subroutine linkage conventions are employed.

If storage was not available for the work area, the register contents upon entry to the STAE exit routine are as follows:

- *Register 0:* 12.
- *Register 1:* ABEND completion code, as in the TCBCMP field.
- *Register 2:* Address of STAE exit parameter list.

The STAE exit routine may contain an ABEND, but must not contain either a STAE or an ATTACH macro instruction. At the time the ABEND is scheduled, the STAE exit routine must be resident as part of the program issuing STAE, or brought into storage via the LOAD macro instruction.

Scheduling of STAE 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.

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), 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 the partition.) The STAE control block is freed and the request block queue is purged of all RBS from the RB of the program that is being terminated up to, but not including, the RB of the program that issued the STAE macro instruction. This is done by placing 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 purged 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. Examples of these effects are as follows:

- Subtasks created by a program to be purged.
- Resources allocated by the ENQ macro instructions.
- DCBS that exist in dynamically acquired storage.

When your 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 operations are restorable. If there are quiesced restorable input/output operations, you can restore them, 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 requests on the I/O restore chain.

You 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 of the first I/O block on the I/O restore chain. You can use this address as a starting point for issuing EXCP for the I/O requests that you want to restore.

In supervisor mode, you may want the failing task to remain in its present status and not be reestablished. A retry routine may be scheduled without a purge of the RB chain by returning to the ABEND/STAE interface routine with an 8 in register 15, and registers 0 and 1 initialized as previously described. 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. If you want to maintain STAE protection against ABEND, you must re-establish an active SCB within the retry routine, or you must issue multiple STAE requests prior to the time that the retry routine gains control.

The STAE 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. Abnormal termination is to continue.
4	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, this 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 STAE retry routine is executed under the PRB that issued the STAE being processed for this abnormal termination.

Like the STAE exit routine, the STAE retry routine must be in storage when the exit routine determines that retry is to be attempted. If not already resident within your program, the retry routine may be brought into storage by the LOAD macro instruction by either the user's program or exit routine.

Upon entry to the STAE 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.
- *Registers 2-13:* Unpredictable.
- *Register 14:* Address of an SVC 3 instruction.
- *Register 15:* Address of the STAE 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 by the EXIT macro instruction, the EXIT routine cancels all SCBs related to the terminating program. If the program terminates by the XCTL macro instruction, the EXIT routine cancels all SCBs related to the terminating program except those SCBs that were created with the 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.

ATTACH—Create a New Task

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 six keyword parameters to those described for the implicit form in the *OS/VS Supervisor Services and Macros* publication. Only the added six parameters are shown and explained in this description.

These six parameters can be used only with tasks whose protection key is zero. If they are used with other tasks, the default values are used.

Name	Operation	Operands
[Symbol]	ATTACH	...,JSTCB= $\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$,SM= $\left\{\begin{matrix}\text{SUPV}\\ \underline{\text{PROB}}\end{matrix}\right\}$,SVAREA= $\left\{\begin{matrix}\underline{\text{YES}}\\ \text{NO}\end{matrix}\right\}$,KEY= $\left\{\begin{matrix}\text{ZERO}\\ \underline{\text{PROB}}\end{matrix}\right\}$,GIVEJPQ= $\left\{\begin{matrix}\text{YES}\\ \underline{\text{NO}}\end{matrix}\right\}$,JSCB = jscbaddr

. . .

Ordinary ATTACH macro instruction parameters. See the description in the *OS/VS Supervisor Services and Macros* publication.

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. A new job step is required if the ownership of programs is to pass from the attaching to the attached task, that is, if you are coding GIVEJPQ=YES in the macro instruction. (Also, see the following *note.*)

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 you are coding GIVEJPQ=YES 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 tasks share subpool zero, the save area is obtained there, otherwise it is obtained from a new 2K byte block.

NO–No save area is needed.

,KEY

Protection/key of the newly created (attached) task.
ZERO–Zero.
PROB–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, you must use JSTCB=YES. (Also see the following note.)
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 acquired by it, are freed.

NO–Ownership of programs used by the attaching task remain with that 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 block of the attaching task is also used for the new task.

Note: If the task to be attached is to be a separate step (JSTCB=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). The following table summarizes these combinations.

		JSTCB =	
		YES	NO
GIVEPJQ =	YES	Valid	Invalid
	NO	Valid	Valid

IMGLIB—Open or Close SYS1.IMAGELIB

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.

Name	Operation	Operand
[symbol]	IMGLIB	OPEN, dcb addr CLOSE

OPEN
: specifies that SYS1.IMAGELIB is to be opened and the address of the DCB returned in register one.

CLOSE
: specifies that IMAGELIB is to be closed.

dcb addr
: is either the address of the IMAGELIB DCB or is a register containing the IMAGELIB DCB address.

QEDIT—Linkage to SVC 34

The QEDIT macro instruction generates the required entry parameters and the linkage to SVC 34 for the following uses:

- Dechaining and freeing of a 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 format of the QEDIT macro instruction and an explanation of the operands are as follows:

Name	Operation	Operand
[symbol]	QEDIT	ORIGIN = address [, BLOCK = address] [, CIBCTR = number]

ORIGIN

The address of the pointer to the first CIB on the 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.

,BLOCK

The address of the CIB that is to be freed from the CIB chain for a task.

,CIBCTR

An integer (from 0 to 255) to be used as a limit for the number of CIBS to be chained at any one time for a task.

address

Any address valid in an RX instruction or one of the general registers (2-12) previously loaded with the indicated address. The register must be designated by a number or symbol added within the parentheses.

EXTRACT—Provide Information from TCB Fields

The EXTRACT macro instruction provides 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 SMI 1.

The standard form of the EXTRACT macro instruction is written as follows. Information about the list and execute forms follows this description.

Name	Operation	Operand
[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. The number of fullwords required is the same as the number of fields specified in the FIELDS operand, unless FIELDS=(ALL) is coded. FIELDS=(ALL) requires seven fullwords.

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.

'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 an address of 0.

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 SMI 1. If the information from a field is not requested, the associated fullword is omitted. If ALL is specified, the answer area includes all the fields in Figure SMI 1 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 0. Fields for which no address or value has been specified in the task control block are set to 0.

ALL—requests information from the GRS, FRS, RESERVED, AETX, PRI, CMC, and TIOT fields.

GRS—the address of the general register 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 floating-point register 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 0.

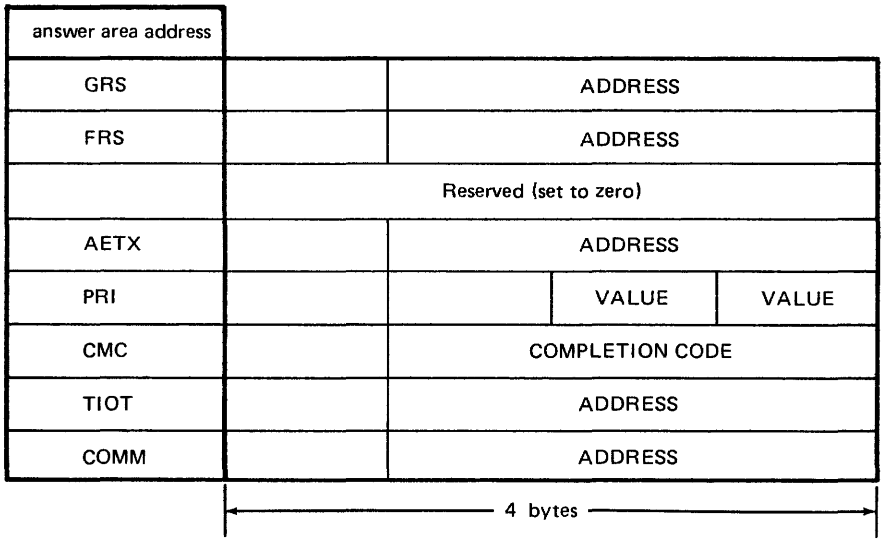
CMC—the task completion code. If the task is not complete, the field is set to 0.

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.

Note: You 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, you must provide a 7-word answer area to accommodate the GRS, FRS, RESERVED, AETX, PRI, CMC, and TIOT fields. The ALL code does not include COMM.

For example, if FIELDS=(TIOT,GRS,PRI) is coded, a 3-word answer area is required, and the extracted information appears in the answer area in the same relative order as shown in Figure SMI 1. (That is, GRS is returned in the first word, PRI in the second word, TIOT in the third word.)



answer area address				
GRS		ADDRESS		
FRS		ADDRESS		
	Reserved (set to zero)			
AETX		ADDRESS		
PRI			VALUE	VALUE
CMC		COMPLETION CODE		
TIOT		ADDRESS		
COMM		ADDRESS		

Figure SMI 1. Field Order for the EXTRACT Answer Area

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 explains the function of each operand. The description of the standard form also indicates which operands are totally optional and which are required in at least one of the pair of list and execute forms. The format description below indicates the optional and required operands in the list form only.

Name	Operation	Operand
[symbol]	EXTRACT	[answer area address] [,{ tcb location address 'S' }] [,FIELDS=(codes)] ,MF=L

symbol
is any symbol valid in assembler language.

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 form 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 explains the function of each operand. The description of the standard form also indicates which operands are always optional and which are required in at least one of the pair of list and execute forms. The following format description indicates the optional and required operands in the execute form only.

Name	Operation	Operand
[symbol]	EXTRACT	[answer area address] [,{ tcb location address 'S' }] [,FIELDS=(codes)] ,MF=(E,{ control program list address (1) })

symbol
is any symbol valid in assembler language.

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 the description of the standard form of the macro instruction. Any previous FIELD operands are canceled and must be respecified if required for this execution of EXTRACT.

$$\text{MF=}\left(\text{E, }\left\{\begin{array}{l}\textit{control program list address}\\ \textit{(1)}\end{array}\right\}\right)$$

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 code MF=(E, (1)).

WTO/WTOR—Write to Operator

A special operand allows key-0 users to route messages to remote work stations. The user must specify the queue identification number of the remote station.

WTO—Standard Form

The format of the standard form of the WTO macro is:

Name	Operation	Operands
[symbol]	WTO	{ 'message' } { ('text'[,line type]),... } [,ROUTCDE=(number[,number...])] [,DESC=(number[,number...])] [,MSGTYP={N | Y | JOBNAMES | STATUS | ACTIVE}] [,MCSFLAG=(name[,name...])] [,QID=nnnnn]

Only the two operands described here are required for RES support. The use of all the other operands is explained in *OS/VS Supervisor Services and Macro Instructions,* GC27-6979.

'message'

is the text of the message to be sent to the RES work station. The maximum length is 125 characters. The text may contain any characters that can be used in a C-type DC instruction.

QID=*nnnnn*

is the decimal queue identification number of the remote user who is to receive the message. If the QID is not equal to zero, only the indicated remote station will receive the WTO(R). If the QID equals zero, normal routing is used.

Specification of this parameter causes the macro to generate the MSGTYP field and to place the binary QID immediately following MSGTYP. Byte 0, bit 3 in MSGTYP is set on (set to 1) to indicate that a QID is present. Byte 0, bit 3 of MCSFLAG is set on to indicate that the MSGTYP field exists.

WTO produces the following return codes:

Return Code	*Meaning*
'00'	No errors
X'04'	Number of lines in parameter list is zero or greater than ten. If zero, request ignored. If greater than ten, only ten are printed.
X'08'	Passed message id cannot be matched with any existing unended MLWTO chain. Request ignored.
X'0C'	Invalid line type encountered. Message terminated at that point.
X'10'	Request had routing code of 11 (WTP). Not allowed for MLWTO. Request ignored.
X'14'	In MLWTO request, queue to hardcopy only bit on in the MCSFLAG field. Request ignored.
X'80'	Unauthorized user of remote WTO(R) (not supervisor or key-0 user). Request ignored.
X'84'	QID invalid (too large). Request ignored.
X'88'	Receiver not logged on. Request ignored.
X'8C'	RTAM unable to find space. Request ignored.
X'90'	RTAM unable to queue message. Request ignored.

WTO—List Form

The format of the list form of WTO is:

Name	Operation	Operands
[symbol]	WTO	{('text'[,line type]),... 'message'} [,ROUTCDE=(number[,number...])] [,DESC=(number[,number...])] [,MSGTYP={N Y JOBNAMES STATUS ACTIVE}] [,MCSFLAG=(name[,name...])] [,QID={nnnnn Y N}]

The QID operand is required for RES support.

QID=

nnnnn — is the decimal queue identification number of the remote user who is to receive the message.

Y — indicates that the 2-byte QID field should be generated and set to X'0000'.

N — indicates that the QID field should not be generated.

WTO—Execute Form

The execute form of the WTO macro is:

Name	Operation	Operands
symbol	WTO	MF= (E, {control program list address (1)}) [,QID= {address (reg)}]

The QID operand is required for RES support.

QID=

address — specifies the address of the 2-byte binary QID. The macro moves the QID from the specified location to the QID field in the parameter list.

(*reg*) — specifies the register containing the address of the 2-byte binary QID. The macro moves the QID to the QID field in the parameter list.

WTOR—Standard Form

The format of the standard form of the WTOR macro is:

Name	Operation	Operands
[symbol]	WTOR	'message', reply address, length of reply ,ecb address [,ROUTCDE=(number [,number...])] [,DESC=(number [,number...])] [,MSGTYP= {N Y}] [,MCSFLAG=(name [,name...])] [,QID=nnnnn]

Only the operands described here are required for RES support. The use of the other operands is explained in *OS/VS Supervisor Services and Macro Instructions,* GC27-6979.

'message'

is the text of the message to be sent to the RES work station. The maximum length is 125 characters. The text may contain any characters valid in a C-type DC instruction.

reply address

is the virtual address of the area into which the reply is to be placed.

length of reply

is the length of the reply message (minimum of one byte).

ecb address

is the address of the ECB (event control block) to be used to indicate completion of the reply.

QID=*nnnnn*

is the decimal queue identification number of the remote user who is to receive the message. If the QID is not equal to zero, only the indicated remote station will receive the WTO(R). If the QID equals zero, normal routing is used.

Specification of this parameter causes the macro to generate the MSGTYP field and to place the binary QID immediately following MSGTYP. Byte 0 bit 3 in MSGTYP is set on (set to one) to indicate that a QID is present. Byte 0, bit 3 of MCSFLAG is set on to indicate that the MSGTYP field exists.

WTOR produces the following return codes:

'00'	No errors
'80'	Unauthorized issuer of remote WTO(R)—not supervisor of key-0 user
'84'	QID invalid—too large
'88'	Receiver not logged on
'8C'	RTAM unable to find space
'90'	RTAM unable to queue message

WTOR—List Form

The list form of WTOR is:

Name	Operation	Operands
symbol	WTOR	'message', [reply address] ,[length of reply], [ecb address] ,MF=L [,ROUTCDE= (number [,number...])] [,DESC= (number [,number...])] $\left[,MSGTYP= \begin{Bmatrix} N \\ Y \end{Bmatrix}\right]$ [,MCSFLAG= (name [,name...])] $\left[,QID= \begin{Bmatrix} nnnnn \\ Y \\ N \end{Bmatrix}\right]$

The QID *operand* is required for RES support.

QID=

nnnnn — is the decimal queue identification number of the remote user who is to receive the message.

Y — indicates that the 2-byte QID field should be generated and set to X'0000'.

N — indicates that the QID should not be generated.

WTOR—Execute Form

The execute form of the WTOR macro is:

Name	Operation	Operands
[symbol]	WTOR	,[reply address], [length of reply] ,[ecb address] ,MF=(E, {control program list address | (1)}) [,QID= {address | (reg)}]

The QID operand is required for RES support.

QID=

address — specifies the address of the 2-byte binary QID. The macro moves the QID from the specified location to the QID field in the parameter list.

(*reg*) — specifies the register containing the address of the 2-byte binary QID. The macro moves the QID to the QID field in the parameter list.

Adding SVC Routines to the Control Program

This section provides detailed information on how to write an svc routine and insert it into the control program portion of vs1.

Documentation of the internal logic of the supervisor and its relationship to the remainder of the control program can be obtained through your IBM Branch Office.

Section Outline

Writing SVC Routines

Because your SVC routine will be a part of the control program, you must follow the same programming conventions used in SVC routines supplied with VS1.

Four types of SVC routines are supplied with VS1 and the programming conventions for each type differ. The general characteristics of the four types are described in the following text, and the programming conventions for all types are shown in tabular form.

Characteristics of SVC Routines

All SVC routines operate in the supervisor state. You should keep the following characteristics in mind when deciding what type of SVC routine to write:

- *Location of the routine*—Your SVC routine can be either in virtual storage at all times as part of the resident control program, or on a direct access device as part of the SVC library. Types 1 and 2 SVC routines are part of the resident control program, and types 3 and 4 are in the SVC library.
- *Size of the routine*—Types 1, 2, and 4 SVC routines are not limited in size. However, you must divide a type 4 SVC routine into load modules of 2048 bytes or less. The size of a type 3 SVC routine must not exceed 2048 bytes.
- *Design of the routine*—Type 1 SVC routines must be reenterable or serially reusable; all other types must be reenterable. If you wish to aid system facilities in recovering from machine malfunctions, your SVC routines should be refreshable.
- *Interruption of the routine*—When your SVC routine receives control, the CPU is masked for all maskable interruptions but the machine check interruption. All type 1 SVC routines must execute in this masked state. If you want to allow interruptions to occur during the execution of a type 2, 3, or 4 SVC routine, you must change the appropriate masks. If you expect that a type 2, 3, or 4 SVC routine will run for an extended period of time, it is recommended that you allow interruptions to be processed where possible.

Programming Conventions for SVC Routines

The programming conventions for the four types of SVC routines are summarized in Figure SVC 1. Details about many of the conventions are in the reference notes that follow the table. The notes are referred to by the numbers in the last column of the table. If a reference note for a convention does not pertain to all types of SVC routines, an asterisk indicates the types to which the note refers.

Conventions	Type 1	Type 2	Type 3	Type 4	Reference Code
Part of resident control program	Yes	Yes	No	No	
Size of routine	Any	Any	≤ 2048 bytes	Each load module ≤ 2048 bytes	
Reenterable routine	Optional, but must be serially reusable	Yes	Yes	Yes	1
May allow interruptions	No	Yes	Yes	Yes	2
Entry point	Must be the first byte of the routine or load module, and must be on a doubleword boundary				
Number of routine	Numbers assigned to your SVC routines should be in descending order from 255 through 200				
Name of routine	IGCnnn	IGCnnn	IGC00nnn	IGCssnnn	3
Register contents at entry time	Registers 3, 4, 5, and 14 contain communication pointers; registers 0, 1, and 15 are parameter registers				4
May contain relocatable data	Yes	Yes	No*	No*	5
Can supervisor request block (SVRB) be extended	Not applicable	Yes*	Yes*	Yes*	6
May issue WAIT macro instruction	No	Yes*	Yes*	Yes*	7
May issue XCTL macro instruction	No	No	No	Yes*	8
May pass control to what other types of SVC routines	None	Any	Any	Any	
Type of linkage with other SVC routines	Not Applicable	Issue supervisor call (SVC) instruction			
Exit from SVC routine	Branch using return register 14				
Method of abnormal termination	Use resident abnormal termination routine	Use ABEND macro instruction or resident abnormal termination routine			9

Figure SVC 1. Programming Conventions for SVC Routines

Reference Code	*SVC Routine Types*	*Reference Notes*
1	all	If your SVC routine is to be reenterable, you cannot use macro instructions whose expansions store information into an inline parameter list.
2	all	Write SVC routines so that program interruptions cannot occur. If a program interruption does occur during execution of an SVC routine, the routine loses control and the task that called the routine terminates. If a program interruption occurs and you are modifying a serially reusable SVC routine, a system queue, control blocks, etc., the modification will never complete; the next time the partially modified code is used, the results will be unpredictable.
3	all	You must use the following conventions when naming SVC routines: • *Types 1 and 2* must be named IGCnnn; nnn is the decimal number of the SVC routine. You must specify this name in an ENTRY, CSECT, or START instruction. • *Type 3* must be named IGC00nnn; nnn is the signed decimal number of the SVC routine. This name must be the name of a member of a partitioned data set. • *Type 4* must be named IGCssnnn; nnn is the signed decimal number of the SVC routine, and ss is the number of the load module minus one. For example, ss is 01 for the second load module of the routine. This name must be the name of a member of a partitioned data set.
4	all	Before your SVC routine receives control, the contents of all registers are saved. For type 4 routines, this applies only to the first load module of the routine. In general, the location of the register save area is unknown to the routine that is called. When your SVC routine receives control, the status of the registers is as follows: • *Registers 0 and 1* contain the same information as when the SVC routine was called. • *Register 2* contains unpredictable information. • *Register 3* contains the starting address of the communication vector table.

Reference Code	*SVC Routine Types*	*Reference Notes*
		• *Register 4* contains the address of the task control block (TCB) of the task that called the SVC routine. • *Register 5* contains the address of the supervisor request block (SVRB), if a type 2, 3, or 4 SVC routine is in control. If a type 1 SVC routine is in control, register 5 contains the address of the last active request block. • *Registers 6 through 12* contain unpredictable information. • *Register 13* contains the same information as when the SVC routine was called. • *Register 14* contains the return address. • *Register 15* contains the same information as when the SVC routine was called. You must use registers 0, 1, and 15 if you want to pass information to the calling program. The contents of registers 2 through 14 are restored when control is returned to the calling program.
5	3,4	Because relocatable address constants are not relocated when a type 3 or 4 SVC routine is loaded into virtual storage, you cannot use them in coding these routines; nor can you use macro instructions whose expansions contain relocatable address constants. Types 1 and 2 are not affected by this restriction since they are part of the resident control program.
6	2,3,4	You can extend the SVRB, in 8-byte increments, from 96 bytes up to 144 bytes. The extended area is available as a work area during execution of your routine only if you specify the extension during the system generation process. When your SVC routine receives control, register 5 contains the address of the SVRB to which the extended save area is appended.
7	2,3,4	You cannot issue the WAIT macro instruction unless you have changed the system mask to allow I/O and external interruptions. If you have allowed these interruptions, you can issue WAIT macro in-

Reference Code	*SVC Routine Types*	*Reference Notes*
		structions that await either single or multiple events. The event control block (ECB) for single-event waits or the ECB list and ECBS for multiple-event waits must be in pageable storage.
8	4	When you issue an XCTL macro instruction in a routine under control of a type 4 SVRB, the new load module is brought into a transient area. The contents of registers 2 through 13 are unchanged when control is passed to the load module; register 15 contains the entry point of the called load module.
9	all	Type 1 SVC routines must use the resident abnormal termination routine to terminate any task. The entry point to the abnormal termination routine is in the communication vector table (CVT). The symbolic name of the entry point is CVTBTERM. Types 2, 3, and 4 SVC routines must use the ABEND macro instruction to terminate the current task, and must use the resident abnormal termination routine to terminate a task other than the current task. Before the resident abnormal termination routine is entered, the CPU must be masked for all maskable interruptions but the machine check interruption, and registers 0, 1, and 14 must contain the following: ● *Register 0* contains the address of the TCB of the task to be terminated. ● *Register 1* contains the following information: Bit 0 is a 1 if you want a dump taken. Bit 1 is a 1 if you want to terminate a job step. Bits 2-7 are zero. Bits 8-19 contain the error code. Bits 20-31 are zero. ● *Register 14* contains the return address. The resident abnormal termination routine exits by branching to the address contained in register 14. The contents of register 15 are destroyed by the abnormal termination routine.

Inserting SVC Routines into the Control Program

You insert svc routines into the control program during the system generation process.

Before your svc routine can be inserted into the control program, the routine must be a member of a cataloged partitioned data set. You must name this data set sys1.name, where the name is a name of your choice.

The following text gives a description of the information you must supply during the system generation process. You will find a description of the macro instructions required during the system generation process in the *VS1 SYSGEN* publication.

Specifying SVC Routines

You use the SVCTABLE macro instruction to specify the svc number, the type of svc routine, and, for type 2, 3, or 4 routines, the number of double words in the extended save area.

Inserting SVC Routines During the System Generation Process

To insert a type 1 or 2 svc routine into the resident control program, use the RESMODS macro instruction. You must specify the name of the partitioned data set and the names of the members to be inserted into the control program. Each member can contain more than one svc routine.

To insert a type 3 or 4 svc routine into the svc library, use the SVCLIB macro instruction. You must specify the name of the partitioned data set and the names of members to be included in the svc library. The member names must conform to the conventions for naming type 3 and 4 routines; that is, IGC00nnn and IGCssnnn.

How to Use the Tracing Routine

This section describes the function of the tracing routine, and provides a detailed description of the information made available by the tracing routine.

Section Outline

How to Use the Tracing Routine

The tracing routine is a vs1 feature which you can use as a debugging and maintenance aid. The tracing routine stores, in a table, information pertaining to the following conditions:

- SIO instruction execution.
- SVC interruption.
- I/O interruption.
- Dispatching interruption.

You can include the tracing routine and its table in the control program during the system generation process. This is done using the TRACE option in the CTRLPROG macro instruction. The format of this option requires you to supply the number of entries in the table. Each table entry can contain information relating to one of the traced conditions. When the last entry in the table is filled, the next entry will overlay the first.

Table Entry Formats

Table entry formats are as follows:

	0			X'8'		X'10'	X'12'
I/O Interrupt	I/O old PSW			Channel Status Word		2cuu	
SIO	SIO cc	Device Address	CAW	Channel Status Word		30xx	
SVC	SVC old PSW			Reg 0	Reg 1	00SVC#	
Dispatch	New PSW			New TCB	Old TCB	10xx	

xx represents meaningless information.

SIO cc is the Start I/O condition code.

For a Start I/O CSW to be valid, SIO cc must be one, otherwise the CSW contents are meaningless.

Location of the Table

The addresses of the last entry made in the table, the beginning of the table, and the end of the table are contained in a 12-byte field. The address of this field is contained in the fullword starting at location 20. The format of the field is as follows:

0 31	0 31	0 31
Address of the Last Entry	Address of the Table Beginning	Address of the Table End

If requested through the SDATA operand of the SNAP macro, the dump lists the SIO, SVC, I/O, and dispatching interruptions table entries, starting with the oldest. A number is assigned to each entry and the oldest entry is 0001.

The Time Slicing Facility

This section describes the time slicing facility, a system generation option available with vs1. Use of this facility allows the grouping of tasks of equal priority or partitions into a time-slice group so that each task within the group is limited to a fixed interval of CPU time each time it is given control. The facility is included in the system mainly to provide a method of controlling response time of a task.

Included in the section are a description of the facility, how it fits into the system, and the applications for which it is most effective. It also describes the prerequisite actions that must be taken, the use of the time slicing facility, and its operating characteristics.

Section Outline

The Time Slicing Facility

The time slicing facility allows the user to establish a group of tasks (called the time-slice group) or partitions that are to share the use of the CPU, each for the same, fixed interval of time. When a member of the time-slice group has been active for the fixed interval 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 length of time. In this way, all member tasks are given an equal slice of CPU time, and no task or partition within the group can monopolize the CPU.

In VS1, only partitions that are assigned to the time-slice group will be time-sliced, and they are time sliced only when the first partition in the group is the highest-priority ready task. Dispatching of the partitions continues within the group until all the partitions are in a waiting state, or until a partition with a higher priority is in a ready state.

The group of tasks to be time sliced (selected by priority or partition range) and the length of the time slice are specified by the installation at system generation time. This can be modified in VS1 through the DEFINE command. Any task or partition in the system that is not defined within the time-slice group is dispatched under the current priority structure; that is, the task or partition is dispatched only when it is the highest priority ready task or partition on the TCB queue.

System Configuration and System Relationships

The time slicing facility can be used with any VS1 configuration. The time slicing facility is especially useful in a graphics environment or in any application of a conversational nature where concurrent tasks may involve conversation between the user and the problem program through a terminal. Establishing a time-slice group within this environment enables those tasks to be performed with a uniform response time.

Prerequisite Actions

Time slicing is specified in the TMSLICE parameter of the CTRLPROG system generation macro instruction. The group(s) of tasks or partitions to be time sliced and the length of the time slice are specified in this parameter.

In VS1, a group of contiguous partitions defines the time-slice group. All tasks scheduled into those partitions are time sliced and are treated as though they had the same dispatching priority. Only *one* group of tasks can be specified to be time sliced. For example, a time-slice group for VS1 might be specified during system generation, as follows:

```
        CTRLPROG            TMSLICE = (P4 - P6, SLC - 256)
```

In this example, partitions P4, P5, and P6 make up the time-slice group and are assigned a time slice of 256 milliseconds for each and every task executing in these partitions.

System Initialization Time

If time slicing has been selected during system generation, the group of tasks to be time sliced and the length of the time slice can be modified during system initialization. In VS1, modifications to the time-slicing specifications are made in much the same way as other partition modifications. At system initialization, changes can be indicated by replying 'YES' to the message: 'IEE801D CHANGE PARTITIONS?'. After system initialization, changes can be indicated through the DEFINE command. In both cases, changes are actually made by responding to the message: 'IEE002A ENTER DEFINITION' or 'IEE803A CONTINUE DEFINITION' with the new TMSL reply. With this reply, the operator can request a list of current time-slicing specifications, change the range of time-slicing partions and the time interval, or cancel time-slicing specifications altogether.

How to Invoke the Time Slice Facility

Time slicing is invoked through either the JOB statement or through the use of the ATTACH and CHAP macro instructions.

If a task is part of the time slice group because its jobclass is assigned to a time slice partition, the task gains control according to the position of the time slice partition with respect to other partitions.

If a task becomes part of the time slice group through the use of ATTACH or CHAP, the task gains control according to the priority used with ATTACH or CHAP. The task gains control, as part of the time slice group, when the partition with the same priority gains control (even though the task resides in a partition that is not part of the time-slice group). Equally, a task that is time sliced may use ATTACH or CHAP with a priority that does not fall within the range of priorities assigned to the time-slice group. The attached or changed task is not part of the time-slice group even though it resides in a time slice partition.

Time Slicing's Effect on the ATTACH and CHAP Macro Instructions

New tasks can be introduced into a time-slice group through the use of the ATTACH and CHAP macro instructions, when the attaching or new priority selected is equal to that of a time-slice group. These new tasks conform to all the rules for time slicing.

The CHAP macro instruction may remove a task from a time-slice group. If it does, this terminates all that task's time-slice characteristics. The ATTACH macro instruction may create a task that is not a member of a time-slice group, even though the originating task was.

Using the Time Slice Facility

The time-slice group is composed of a group of contiguous partitions and all tasks scheduled into those partitions are time sliced. Also, each partition in the system is assigned to at least one job class. Since a job is scheduled into a partition according to the CLASS parameter on the JOB statement, careful consideration should be given to the job-class assignment in order to enable the user to control the use of time slicing at his installation. For example,

1. Partitions P0-P2 have been assigned as the time-slice partitions

2. The partitions have been assigned the following job classes:

   P0=G, P1=G, P2=(G,D), P3=B, P4=(B,C,D)

In this example, the user can ensure that a job will be time sliced by specifying CLASS=G on the JOB statement. This specification guarantees that the scheduler will initiate the job only into a partition assigned to CLASS G, that is, P0, P1, or P2. Since P0-P2 have been designated as time-slice partitions, that job will be time sliced.

> *CAUTION*
> Note that if the CLASS parameter of a job was D, the job may or may not be time sliced, depending on whether it is initiated into partition P2 or P4. See the appropriate *Message Library* publication (message IEE802A) for information on warning the operator about such situations.

Time slicing is assigned both by partition (as shown) and by dispatch priority of the jobclasses assigned to the time slice partitions. If a program uses the ATTACH or CHAP macro instruction, the priority used with ATTACH or CHAP determines whether the attached or changed task is time sliced, not the partition in which it resides. (However, a program cannot exceed the limit priority assigned its jobclass.) See the *Supervisor Services and Macros* publication for a discussion of dispatch and limit priority.

Operating Characteristics

The time-slicing mechanism operates within the structure of the current dispatcher. A priority is assigned to a group of tasks that are to be time sliced. The time slicing occurs among the tasks in the group only when the priority level of the group is the highest priority level that has a ready task. Each task or partition in the group is dispatched for the specified time slice. The time slicing continues until either all tasks or partitions are waiting, or a task or partition of higher priority than that of the group becomes ready.

The dispatcher will recognize that a priority level is one that is being time sliced; it will determine which task or partition within the group is to be dispatched and then dispatch that task or partition for the maximum time interval. If the time slice task loses control prior to the expiration of its interval (because an *implicit or explicit* wait is issued, or because a higher priority task or partition becomes ready), *the remainder of the time is not saved.* That is, when control returns to the time-slice group, the next ready task or partition in the group is given control, not the interrupted task or partition.

Effect of System Tasks on Time-Slice Groups

The time slicing option is included in the system mainly to provide a method of controlling response time of a task. However, since it is being implemented in a priority dispatcher, any task of a higher priority than that of the time-slice group will be dispatched first, if it is ready. Note also that the time-slicing mechanism applies only to the problem program priorities, 0-13. Priorities 14 and 15 are reserved for the system and cannot be time sliced. Therefore, the response time of a time-slice task can be affected by the processing of system tasks, such as readers, writers, master scheduler, etc., which will always run at a higher priority than the time-slice group. Therefore, to guarantee response time, the time slice group should be defined in the high priority partitions.

Non-interactive jobs should not be run concurrently and time sliced since this may significantly decrease performance.

Writing System Output Writer Routines

This section provides guidelines for writing your own output writer routines for your vs1 operating system.

Section Outline

Writing System Output Writer Routines

When a job is executing, system messages and data sets specifying the SYSOUT parameter (for example, in the DD statement) are recorded on direct access devices, unless the job falls into a job class assigned to a direct SYSOUT writer. In that case, both messages and data addressed to a SYSOUT data set are written directly to the device for the direct SYSOUT writer for that job class. (Messages for jobs canceled on the input queue and jobs failed by the reader, and data produced by system tasks cannot be processed by direct system output writers.)

When the job completes (assuming it doesn't use a direct SYSOUT writer), 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 transcribing system messages and data sets to the output device. The 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 you may write a nonstandard data set writer routine.

Note: User-written output writer routines are not supported to RTAM devices.

Output Writer Functions

Before writing or modifying an output writer routine, you should be familiar with the functions performed by the standard data set writer for VS1. (For the remainder of this section the VS1 data set writer is referred to as the standard writer.) 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, checkpointing the data set if requested, 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. The standard writer in VS1 is attached (by the ATTACH macro instruction) at START time and no longer uses a subtask for normal data set processing.

Conventions to be Followed

To use your own output writer routine, you must specify the name of your routine as a parameter in the SYSOUT operand of a DD statement (see the *Job Control Language* publication). (This parameter is ignored if the job falls into a jobclass assigned to a direct SYSOUT writer.) Your routine must be in the system library (SYS1.LINKLIB). A writer routine is not limited in size except that size may influence the partition requirements of the system output writer (see the *OS/VS1 Storage Estimates* publication).

Byte 0	Output Device Indicator. Bit 0 (High-order bit): If this bit is on (set to 1), the output unit is a 1442 punch. Bit 1 If this bit is on, the output unit is either a punch or a tape with a punch as the ultimate 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 (i.e., do not contain information significant to data set writers, but must be left intact.)
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 your writer, the input data set is not open.)
Bytes 12-15	Pointer to the CANCEL ECB for this writer. The ECB will be posted by the CANCEL command, and the user writer may take any desired action.

Figure WWT 1. Parameter List Referred to by Register 1

Your routine is attached (by 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. Your output writer routines must be reenterable.

When your routine is finished, it must return control to the standard writer by using the RETURN macro instruction.

After job management routines perform initialization requirements and open the output data set into which your writer routine will put records, control is given to your routine by the ATTACH macro instruction. At this time, general registers 1 and 13 contain information that your program must use. Register 1 contains the storage address of a 16-byte list. Figure WWT 1 describes the information in this parameter list. Your writer should check the CANCEL ECB that is passed in the parameter list if the writer is to be cancelable.

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. Based on the indications given in Figure WWT 1, the high-order three bits of byte 0 signify the type of output device as follows:

111 . . . . . 1442 punch unit
011 . . . . . 2520 punch unit or 2540 punch unit
001 . . . . . 1403 printer, 1443 printer, or 3211 printer unit
010 . . . . . tape unit with ultimate punch destination
000 . . . . . tape unit with ultimate printer destination

When your writer gets control, it must preserve the contents of registers 0 through 12, and 14. Register 13 contains the address of a standard register save area where you are to save the contents of these registers. You can save the contents of register 13 by using the SAVE macro instruction.

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. The *OS/VS Data Management Macro Instructions* publication describes this macro instruction.)

Your data set writer must provide a SYNAD routine to process errors associated with the output as well as the input data set.

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 your 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).

General Processing Performed by Standard Output Writer

This section provides a general description of the procedures followed by the standard writer. If you write your own writer routine, you may wish to delete, modify, or add to some of these procedures, depending on the characteristics of your data set(s). However, your procedures must be consistent with operating system conventions.

SAVING REGISTER CONTENTS

Upon entering the writer program, your program must save the contents of the general registers, as previously discussed.

OBTAINING 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. You should obtain storage 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 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 ANSI control characters are used in the output data set records, the control character effect is considered before the printer prints a record. See *Control Character Transformations.*

   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 ANSI 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 your data sets) is recommended.

2. After an input data set is properly opened and any necessary printer initialization is completed, the writer obtains records from the input data set. The standard writer uses a unique interface to the spool manager to open, close, and obtain each input record. As each record is obtained, its control character must be adjusted, if necessary, to agree with that required for output. If a control character is added to the input record, it is placed to the left of the first byte of data. The standard writer receives only undefined format records from the spool manager and passes them to the JES-oriented access method (JAM). The record format specified for the output data set is ignored for unit record devices. For a tape output device, the correct format is constructed by JAM before the record is written. 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 necessary right-hand bytes of the input record are truncated. If the input record length is smaller than the output record length, JAM 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, JAM constructs each output record by adding variable record control information to the output record. The record control information is four bytes long and is added 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, JAM makes the proper adjustment.

   Note:
   The input and output interfaces utilized by the standard writer are not available to user writer routines. The writer constructs the normal QSAM interface before attaching the user routine. Since the output data set is previously opened, a user writer routine must adhere to the established conventions. The output data set is opened to receive records from the PUT macro instruction operating in the locate mode. It is the responsibility of the user's routine to verify that correct control characters and record formats are passed to QSAM.

   The input DCB for the user routine is initialized to use the GET macro in locate mode. This can be overridden before OPEN, if desired.

3. The third aspect to consider 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 (for example, 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 set 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.)

CHECKPOINTING OUTPUT DATA SETS

If the system output writer procedure specifies a SYSOUT data set checkpoint interval, each SYSOUT data set using that writer procedure is checkpointed at the specified intervals. Should the system be re-IPLed with warmstart prior to completion of writing out a data set, the operator (in reply to message IEF595D DDD WTR, RESUME AT CHECKPOINT—REPLY Y OR N) can specify that the writing of the data set resume at the last checkpoint or at the beginning of the data set.

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. You must close all input data sets.

RELEASING 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.

Control Character Transformations

To help determine what you can do with a writer routine, the control character transformation features of the standard writer are described.

Effectively there are nine control character combinations that can occur between input data set records and output data set records. Both data sets may have records whose control characters are either ANSI (American National Standards Institute) type (acc) or machine type (mcc), or the records may not con-

	Machine Control Characters			ANSI
Stacker Unit	2540 Punch	2520 Punch	1442 Punch	Control Characters
1. P1	00000001	00000001	10000001	11100101 (V)
2. P1 Column Binary	00100001	00100001	10100001	
3. P2	01000001	01000001	11000001	11100110 (W)
4. P2 Column Binary	01100001	01100001	11100001	
5. RP3	10000001			
6. RP3 Column Binary	10100001			

Figure WWT 2. Control Character Translation for Punch Unit Output

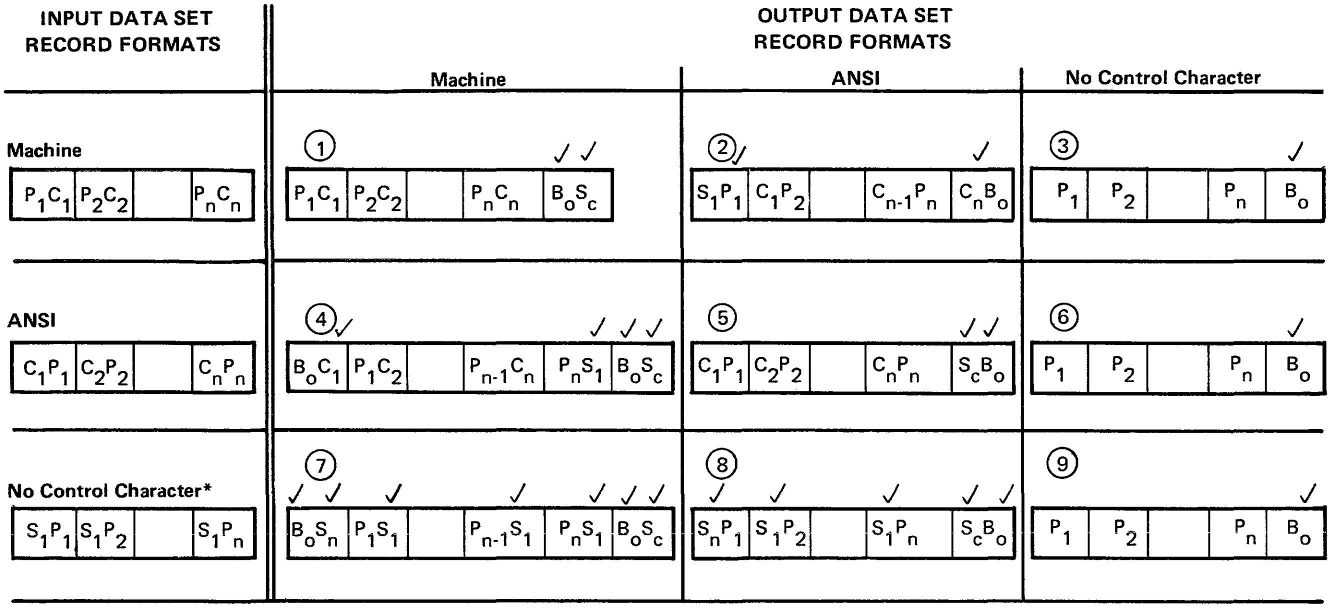
tain any control characters. However, within any given data set, the records all must contain the same form of control character. The standard writer has procedures to handle control character transformations for both an output to a card punch unit and an output to a pinter unit.

Card Punch Unit

If an input data set record does not have a control character, the standard writer produces one that indicates output into pocket 1 of the punch. If the output unit is a tape unit and the ultimate destination is a punch unit, the standard writer assumes that the punch unit is either a 2540 or a 2520 unit and sets a control character accordingly. The standard writer translation of punch-type control characters is given in Figure WWT 2. In this table, the first three columns of figures are machine control character codes, and the right hand column of figures represents ANSI control character codes. Each record that requires a control character has one of these 8-bit codes attached to it. Input records whose control characters are mcc and are shown in horizontal rows 1, 2, 5, and 6 are given the acc code of 'V' if they are placed in an output data set that has acc. An mcc given in rows 3 or 4 is changed to an acc code of 'W'. However, if translation is from an acc input to an mcc output, the standard writer translates the control character into the appropriate mcc on the same horizontal row.

Printer Unit

When the output unit is a printer, the standard writer prevents overprinting between data sets. If the successive data sets contain records with the same type of control character, there is no overprinting problem. If the control characters vary from one data set to the next, the standard writer solutions are applications of the technique illustrated by Figure WWT 3. In this figure, the possible forms of the input record control characters are given in the left hand column. The three right hand columns (containing cases 1-9) represent the possible forms of the output record control characters. Within each of the 12 main sections of the figure is shown a symbolic representation of a data set whose records possess the indicated form of control character. Each record consists of a print line representation and a control character representation (where appropriate). For records with acc, the control character is shown preceding the print line, since the effect of the control character occurs before the line is printed. For records with mcc, the converse is shown. An input record with no control character is treated as if it had an acc. Because of this variance in the printer's mechanical action, whenever there is a control character transformation, the standard writer places a transformed control character with an output data set record other than the record to which the character was attached in the input data set.


INPUT DATA SET RECORD FORMATS
OUTPUT DATA SET RECORD FORMATS
Machine
ANSI
No Control Character
Machine
P_1C_1 P_2C_2 P_nC_n
① ✓ ✓
P_1C_1 P_2C_2 P_nC_n B_oS_c
②✓ ✓
S_1P_1 C_1P_2 $C_{n-1}P_n$ C_nB_o
③ ✓
P_1 P_2 P_n B_o
ANSI
C_1P_1 C_2P_2 C_nP_n
④✓ ✓ ✓ ✓
B_oC_1 P_1C_2 $P_{n-1}C_n$ P_nS_1 B_oS_c
⑤ ✓ ✓
C_1P_1 C_2P_2 C_nP_n S_cB_o
⑥ ✓
P_1 P_2 P_n B_o
No Control Character*
S_1P_1 S_1P_2 S_1P_n
⑦ ✓ ✓ ✓ ✓ ✓ ✓ ✓
B_oS_n P_1S_1 $P_{n-1}S_1$ P_nS_1 B_oS_c
⑧ ✓ ✓ ✓ ✓ ✓
S_nP_1 S_1P_2 S_1P_n S_cB_o
⑨ ✓
P_1 P_2 P_n B_o

✓	=	Writer generated.
*	=	No control character on input causes the standard writer to generate an ANSI control character as indicated.
B_o	=	A print line of blanks.
C_1-C_n	=	Control characters of records 1-N of a given data set.
P_1-P_n	=	Print lines of a given data set.
S_1	=	A control character causing a 1-line space.
S_c	=	A control character causing spacing to be suppressed.
S_n	=	A control character causing a skip to channel 1.

Figure WWT 3. Symbolic Representation of Record Formats

In Figure WWT 3, cases 1 and 5 represent situations in which there is the same type of control character in the output as there is in the input. Thus, for records 1 through n, there is no change in the record format. However, there is a provision to allow for the possibility that two consecutive input data sets may have different control characters. In this case, a minimum separation between the data sets as they appear in the output data set is provided as indicated by the printing of blanks and suppressing the spacing of the printer to allow another control character to take effect. The 'extra' record (S_c B_o or B_o S_c) provides the more important function of forcing out the last record of the current data set before the writer's processing of that data set is done.

In cases 2 and 4 of Figure WWT 3, the output data set records have different control characters than the input data set records. Case 2 shows that the standard writer generates a 1-line space control character to precede the first print line of the output. When the output is written, each control character of an input record is then attached to the next record. The last input record control character (C_n) is attached to a line of blanks (B_o). In case 4, the first input record control character is attached to a line of blanks, and each of the other control characters is attached to a preceding record, as indicated. The last input record (P_n) has a writer-generated space 1-line control character attached to it before the buffering and forcing record (B_o S_c) generated by the writer is put out.

Cases 7 and 8 show that the standard writer first generates a 'skip to channel 1' control character and then generates '1 line space' control characters for all but the last control character. The last control character is the space suppression character shown as part of the buffering or forcing record generated.

Cases 3, 6, and 9 show that if no control characters are required in the output data set, the records are printed consecutively and a line of blanks generated by the writer is printed after the final record in a data set. Any control characters appearing in the input data set are dropped in the output data set.

Notice that in all cases involving control characters in the output data set, the standard writer allows for (1) an output record to force the printing of the last record of an input data set and (2) a means of minimum buffering between data sets by using generated control characters and print lines in conjunction with the actual data set control characters.

The standard writer translation of printer-type control characters is given in Figure WWT 4. In this table, the type of action indicated is given in the left-hand column. The middle column and the right-hand column show, respectively, the bit settings of the control character byte for machine type and ANSI type control characters corresponding to the entries in the left-hand column. A control character transformation is effected by changing the bit-configuration of the control character byte as indicated in the table.

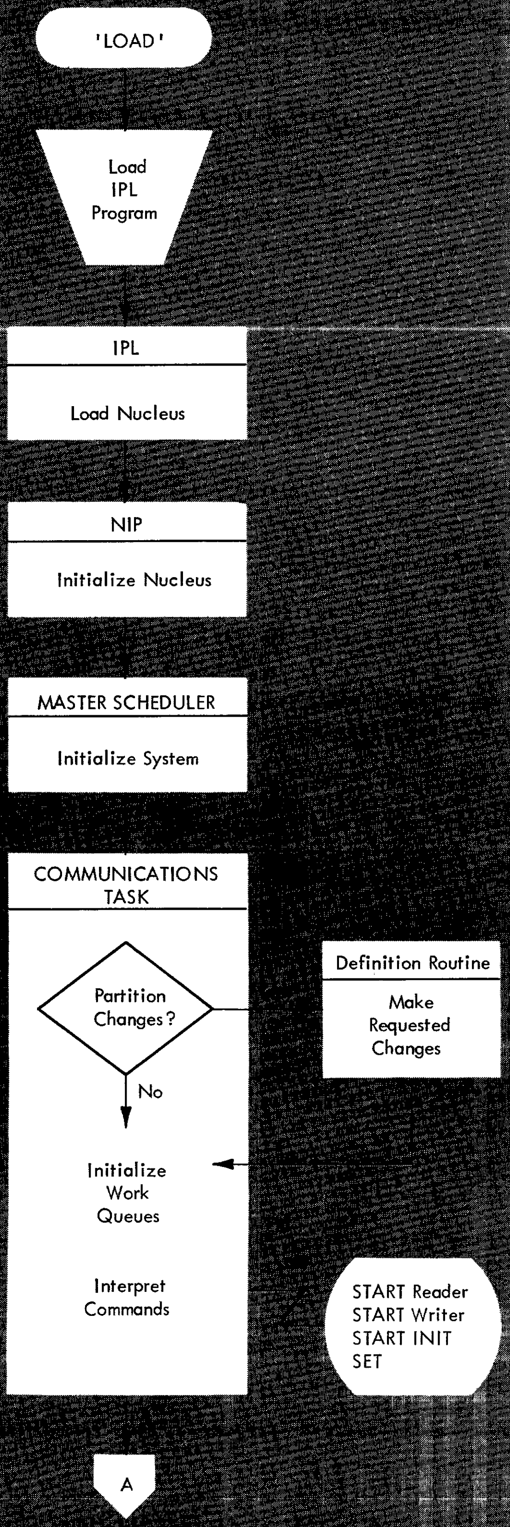
When machine control characters are used which indicate spacing or skipping without writing (bit 6 set to 1, for example, write and space 0-00000011) the standard writer generates the indicated ANSI control character and also generates a blank record of the proper type and length.

Action Desired	Machine Type Control (1403, 1443, 3211 Printers)	ANSI Type Control
Write space 0	00000001	01001110
Write space 1	00001001	01000000
Write space 2	00010001	11110000
Write space 3	00011001	01100000
Write skip to channel 1	10001001	11110001
Write skip to channel 2	10010001	11110010
Write skip to channel 3	10011001	11110011
Write skip to channel 4	10100001	11110100
Write skip to channel 5	10101001	11110101
Write skip to channel 6	10110001	11110110
Write skip to channel 7	10111001	11110111
Write skip to channel 8	11000001	11111000
Write skip to channel 9	11001001	11111001
Write skip to channel 10	11010001	11000001
Write skip to channel 11	11011001	11000010
Write skip to channel 12	11100001	11000011

Figure WWT 4. Control Character Translation for Printer Unit Output

Appendix A: Theory of Operation

Figure APA 1 describes the overall processing flow through each job cycle. The paragraphs in the figures describe the processing performed by various components of the control program as it loads the nucleus, reads control statements, initiates the job step, causes processing to begin or end in other partitions, and terminates the job step.

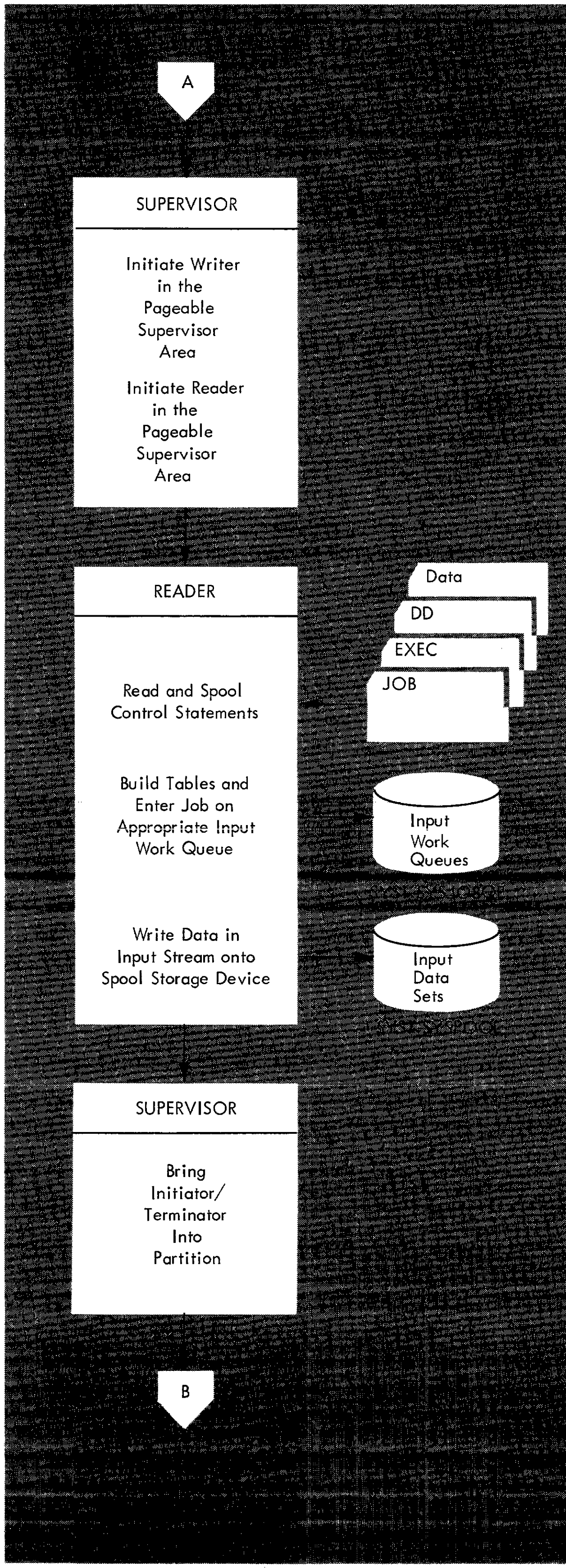


To load the nucleus, the operator sets the LOAD UNIT switches to the device on which the system residence volume is mounted and presses the LOAD button on the operator control panel. This causes an IPL record to be read and to be given control. This record causes the second IPL record to be read, which in turn, enables the rest of the IPL program to be read into real storage.

The IPL program searches the volume label of the system residence volume to locate the volume table of contents (VTOC). The VTOC is then searched for the address of the nucleus data set (SYS1.NUCLEUS). The nucleus is brought into the system area, and NIP is brought into the pageable area. NIP receives control from the IPL program. It performs both required and optional initialization for control program operation including initializing the Communication Vector Table (CVT), and general system initialization, such as determining user options. After completing its processing, NIP passes control to the master scheduler task (MST) which initializes virtual storage, including JES.

Partitions are established by the master scheduler at system initialization according to the sizes and job class(es) established at system generation by the PARTITNS macro instruction. The MST also places a copy of the Initiator/Terminator into each partition. The communications task receives control from the MST and communicates with the operator to request any partition changes. After the requested changes, if any, have been made by the definition routines, the work queues are initialized. The automatic commands are displayed, and the READY message is issued.

Figure APA 1. VS1 Theory of Operation (Part 1 of 4)


A
SUPERVISOR
Initiate Writer in the Pageable Supervisor Area
Initiate Reader in the Pageable Supervisor Area
READER
Read and Spool Control Statements
Build Tables and Enter Job on Appropriate Input Work Queue
Write Data in Input Stream onto Spool Storage Device
Data
DD
EXEC
JOB
Input Work Queues
SYS1.SYSJOBQE
Input Data Sets
SYS1.SYSPOOL
SUPERVISOR
Bring Initiator/ Terminator Into Partition
B

When the required SET command is entered, the communications task calls the master scheduler command scheduling routine to have the command executed. An automatic START reader command or a subsequent operator-entered START reader command initiates a reader previously loaded in the pageable supervisor area. If a START writer command is entered, a writer is initiated and made dispatchable in the pageable supervisor area.

When the reader gets control, it reads control statements, commands, data, and procedures from the input stream. This information is placed on the appropriate direct-access storage device data set (SYS1.SYSPOOL). Information from the JOB, EXEC, and DD statements controls the execution of each job step.

The reader also builds disk entry records and accounting records for each job and places them in the input work queue (SYS1.SYSJOBQE) corresponding to the CLASS parameter of the JOB statement.

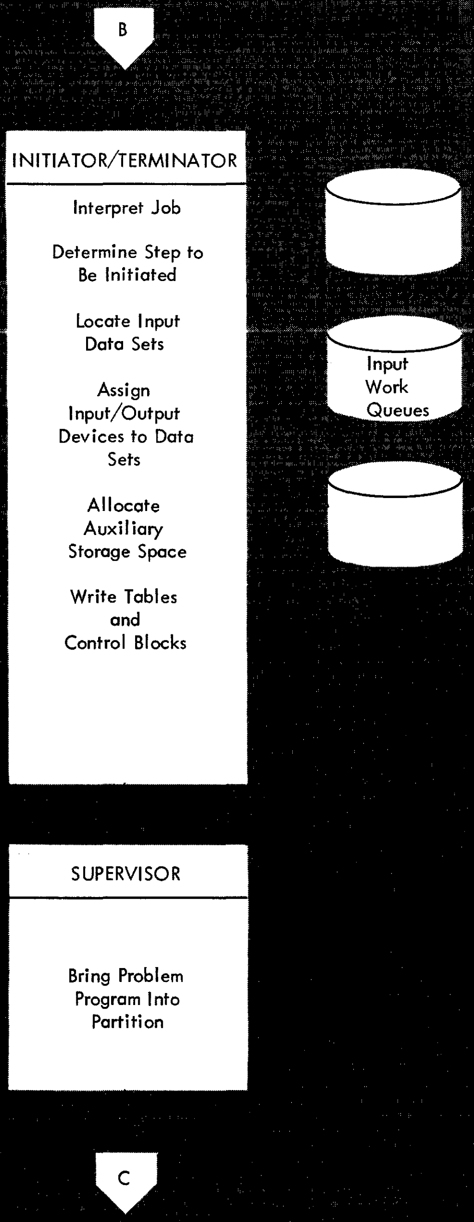
After the reader has completed processing all input for a job and has entered the job on an input work queue, all initiators that are waiting for that job class are posted.

After receiving control, the initiator prepares to initiate the highest priority job in its primary input work queue. Using information that the reader extracted from the DD statement, the initiator processes the user accounting routine and then passes control to the interpreter, which runs as a subroutine of the initiator. The interpreter locates and interprets the JCL for the job and builds the following tables:

- Job control table (JCT) for the job being read.
- Step control table (SCT) for the step being read.
- Data set enqueue table (DSENQ) for the job being read.
- Job file control block (JFCB) and step input/output table (SIOT) for each data set being used or created by the job step.
- Volume table (VOLT) containing each volume serial number to be used by the job.
- Other tables are constructed if needed to completely interpret the JCL for the job.

The interpreter places these tables and control blocks in the Scheduler Work Area Data Set (SWADS). Information from these tables and control blocks is updated with information in the data control block (DCB) and data set control block (DSCB) or volume label when a data set is opened during step execution.

Figure APA 1. VS1 Theory of Operation (Part 2 of 4)


B
INITIATOR/TERMINATOR
Interpret Job
Determine Step to Be Initiated
Locate Input Data Sets
Assign Input/Output Devices to Data Sets
Allocate Auxiliary Storage Space
Write Tables and Control Blocks
Input Work Queues
SUPERVISOR
Bring Problem Program Into Partition
C

The interpreter then returns control to the initiator, which does the following:

Locates Input Data Sets: The Allocation routine, running as a subroutine of the initiator/terminator, determines the volume containing a given input data set by examining the JFCB, or by searching the catalog. This search is performed by a catalog management routine entered from allocation.

Allocates I/O Devices: A job step cannot be initiated unless there are enough I/O devices to fill its needs. Allocation determines whether the required devices are available, and makes specific assignments. If necessary, messages are issued to the operator to request the mounting of volumes.

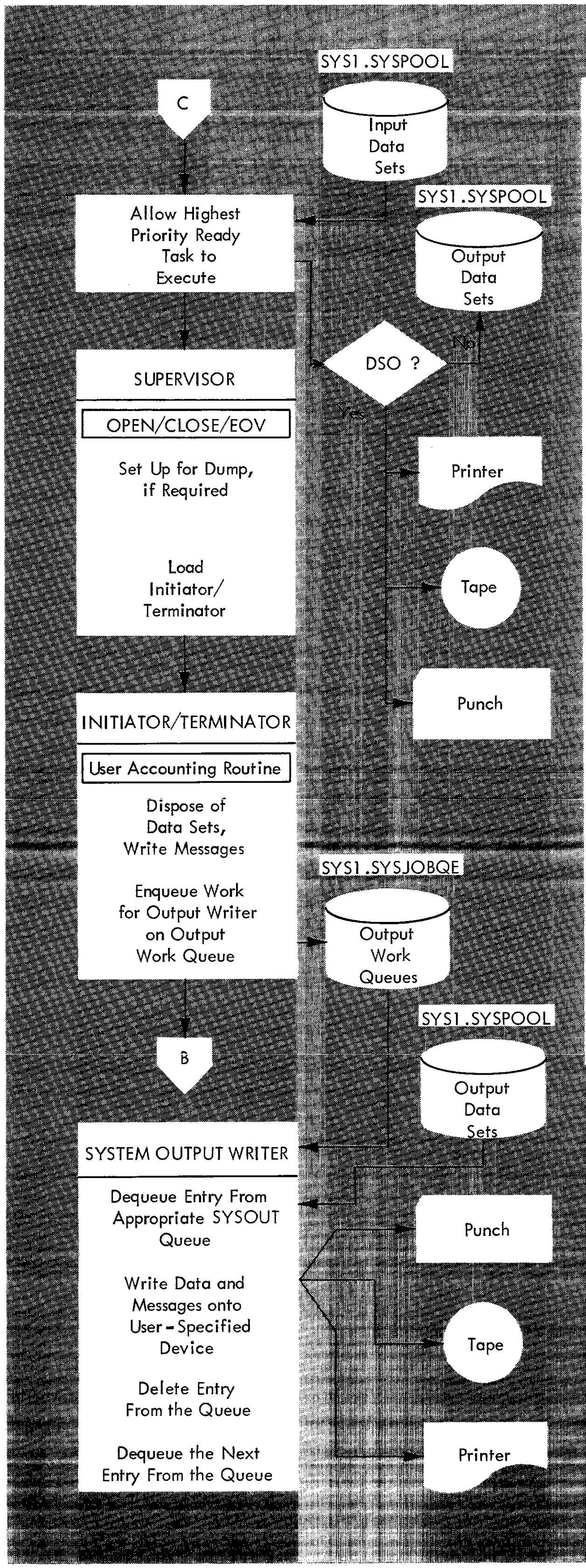
Allocates Auxiliary Storage Space: Direct access volume space required for new data sets of a job step is acquired by the allocation routine, which uses the Direct Access Device Space Management (DADSM) routines.

The JFCBs, which contain information concerning the data sets to be used during step execution, are written on auxiliary storage. This information is used when a data set is opened, closed, and when the job step is terminated.

The initiator transfers control to the problem program it is initiating.

The problem program can be an IBM-supplied processor (e.g., COBOL, linkage editor), or a user-written program. The problem program uses control program services for operations such as loading other programs and performing I/O operations.

Figure APA 1. VS1 Theory of Operation (Part 3 of 4)


SYS1.SYSPOOL
C
Input Data Sets
Allow Highest Priority Ready Task to Execute
SYS1.SYSPOOL
Output Data Sets
DSO ?
No
Yes
SUPERVISOR
OPEN/CLOSE/EOV
Set Up for Dump, if Required
Load Initiator/ Terminator
Printer
Tape
Punch
INITIATOR/TERMINATOR
User Accounting Routine
Dispose of Data Sets, Write Messages
Enqueue Work for Output Writer on Output Work Queue
SYS1.SYSJOBQE
Output Work Queues
SYS1.SYSPOOL
B
Output Data Sets
SYSTEM OUTPUT WRITER
Dequeue Entry From Appropriate SYSOUT Queue
Write Data and Messages onto User-Specified Device
Delete Entry From the Queue
Dequeue the Next Entry From the Queue
Punch
Tape
Printer

The problem program processes until it terminates either normally or abnormally, though it may not retain exclusive control of the CPU. Control always is received by the highest priority task that is ready to execute.

When the problem program terminates, the supervisor receives control. The supervisor uses the OPEN/CLOSE/ EOV routines to close any open data control blocks.

Under abnormal termination conditions, the supervisor may also provide special termination procedures, such as a storage dump. The supervisor passes control to the initiator/terminator, which is brought into the partition in which termination is to occur.

The initiator/terminator releases the I/O devices, and disposes of data sets used and/or created during the job step by reading tables prepared during initiation (JCT, SCT, TIOT, etc.). These tables include information such as disposition of data sets. It then executes an installation accounting routine if one is provided.

At termination of a job not using direct system output processing, an entry is made on the user specified output work queue (SYS1.SYSJOBQE); later the problem program output data can be written by a system output writer from a system direct-access storage device (SYS1.SYSPOOL) to a user-specified device. The initiator/terminator then initiates the next job.

An output writer operates concurrently with readers, problem programs, and other writers. When the START command is issued for a writer, the writer dequeues the first entry in the specified output (SYSOUT) class queue. If no requests have been enqueued in that output queue from the problem programs, the writer is placed in a wait condition until a job is terminated that has system messages or output data sets in the specified class. After the entry is dequeued from the output queue, the writer transmits the data sets to the specified card punch, magnetic tape unit, or printer. When the last record has been processed, the writer deletes the queue entry before dequeuing the next entry

Figure APA 1. VS1 Theory of Operation (Part 4 of 4)

Glossary

active page: A page in real storage that can be addressed.

active page queue: A queue of pages in real storage that are currently assigned to tasks. Pages on this queue are eligible for placement on the available page queue. See also available page queue, hold page queue.

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.

available page queue: A queue of the pages whose real storage is currently available for allocation to any task. See also active page queue, hold page queue.

basic control (BC) mode: A 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. See also extended control (EC) mode.

BC mode: See basic control mode.

change bit: A bit associated with a page in real storage; the change bit is turned ON by hardware whenever the associated page in real storage is modified. In VS1, there is a change bit in the storage key associated with each 2K storage block.

channel program translation: In a channel program, replacement by software of virtual addresses with real addresses.

control registers: A set of registers used for operating system control of relocation, priority interruption, program event recording, error recovery, and masking operations.

DAT: See 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.

dormant state: A state in which the active pages of a job have been paged-out.

DSS: See 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 partitions that are assigned to job steps and system tasks. See also pageable dynamic area, nonpageable dynamic area.

dynamic support system (DSS): An interactive debugging facility that allows authorized maintenance personnel to monitor and analyze events and alter data.

EC mode: See 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 address: An address that identifies the location of a page in a page data set. This address is computed from the page number each time a page is to be transferred between real storage and external page storage.

external page storage: The portion of auxiliary storage that is used to contain pages.

external page storage management: A set of routines in the paging supervisor that control external page storage.

fixed: In VS1, 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 page: A page in real storage that is not to be paged out.

hold page queue: A queue to which pages in real storage are initially assigned through operations such as page-in or page reclamation. See also active page queue, available page queue.

input stream control: See *JES* reader.

invalid page: A page that cannot be directly addressed by the dynamic address translation feature of the central processing unit.

JECS: See job entry central services.

JEPS: See job entry peripheral services.

JES: See job entry subsystem.

JES reader: The part of the job entry subsystem that controls the input stream and its associated job control statements. Synonymous with input stream control.

JES writer: The part of the job entry subsystem that controls the output of specified data sets. Synonymous with output stream control.

job entry central services (JECS): The part of the job entry subsystem that provides centralized storage and retrieval of: (1) system input and output data for each job, (2) control tables representing jobs, and (3) job tables used during job execution.

job entry peripheral services (JEPS): The part of the job entry subsystem that schedules and performs reader and writer operations.

job entry subsystem (JES): A system facility for spooling, job queueing, and managing the scheduler work area data sets.

lock/unlock facility: A supervisor facility that controls the execution of instruction strings when a disabled page fault occurs.

main storage: See real storage, virtual storage.

memory: See real storage, virtual storage.

missing page interruption: See page fault.

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. Synonymous with V=R dynamic area.

nonpageable partition: In VS1, a subdivision of the nonpageable dynamic area that is allocated to a job step or system task that is not to be paged during execution. In a nonpageable partition, each virtual address is identical to its real address. Synonymous with V=R partition.

OS/VS: See Operating System/Virtual Storage.

Operating System/Virtual Storage: A compatible extension of the System/360 Operating System that supports relocation hardware and the extended control facilities of System/370.

OUTLIM (output limiting) facility: A facility that monitors the number of logical records produced for SYSOUT data sets.

output stream control: See JES writer.

page: (1) A fixed-length block of instructions, data, or both, that can be transferred between real storage and external page storage. (2) To transfer instructions, data, or both between real storage and external page storage.

page control block (PCB): A control block that indicates the status of a paging request.

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 as nonpageable so that it remains in real storage.

page number: The part of a virtual storage address needed to refer to a page.

page reclamation: The process of making addressable the contents of a page in real storage that has been marked invalid. Page reclamation can occur after a page fault or after a request to fix or load a page.

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.

page wait: A condition in which the active request block for a task is placed in a wait state while a requested page is located in real storage or is brought into real storage.

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. Synonymous with V=V dynamic area.

pageable partition: In VS1, a subdivision of the pageable dynamic area that is allocated to a job step or system task that can be paged during execution. Synonymous with V=V partition.

paging: The process of transferring pages between real and external page storage.

paging device: A direct access storage device on which a page data set (and possibly other data sets) are stored.

paging rate: The average number of page-ins and page-outs per unit of time.

paging supervisor: A part of the supervisor that allocates and releases real storage space for pages and initiates page-in and page-out operations.

PCB: See page control block.

PER: See program event recording.

PGT: See page table.

PQA: See protected queue area.

program event recording (PER): A hardware feature used to assist in debugging programs by detecting program events.

protected queue area (PQA): In VS1, an area located at the high address end of each virtual storage partition.

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.

real storage page table (RSPT): In VS1, a table that contains an entry for each 2K storage block in real storage. This table is the centralized information interface for real storage management.

reference bit: A bit associated with a page in real storage; the reference bit is turned "ON" by hardware whenever the associated page in real storage is referred to (read or stored into). In VS1, there is a reference bit in the storage key associated with each 2K storage block.

relocate hardware: See dynamic address translation.

request parameter list (RPL): A list of parameters that accompanies a request for job entry subsystem services.

scheduler work area data set (SWADS): In VS1, a data set on auxiliary storage that contains most of the job management control blocks (such as the JCT, JFCB, SCT, and SIOT). There is one SWADS for each initiator.

segment: A continuous, 64K area of virtual storage that 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 table entry (STE): An entry in the segment table that indicates the length, location, and availability of a corresponding page table.

segment translation exception: 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.

SGT: See segment table.

space record: A record that separates page slots in a page data set.

SQA: See system queue area.

STE: See segment table entry.

static CP area: In VS1, those portions of virtual storage that are allocated, during system generation and initial program load, to control program functions.

storage block: A 2K block of real storage to which a storage key can be assigned.

supervisor lock: An indicator used to inhibit entry to disabled code while a disabled page fault is being resolved.

SWADS: See scheduler work area data set.

system lock: In VS1, an indicator in the communications vector table, used to inhibit the dispatching of any task, except paging supervisor tasks.

system queue area (SQA): An area of virtual storage reserved for system-related control blocks.

thrashing: A condition in which the system can do little useful work because of excessive paging.

translation tables: Page tables and segment tables.

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: An area of virtual storage that has the same range of addresses as real storage and is used for a program or part of a program that cannot be paged during execution.

V=R dynamic area: See nonpageable dynamic area.

V=R partition: See nonpageable partition.

V=V dynamic area: See pageable 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 loctions.

virtual storage partition: In VS1, a division of the dynamic area of virtual storage, established at system generation.

working set: The set of a user's pages that must be active in order to avoid excessive paging.

index

OS/VS1
Planning and Use Guide
GC24-5090-1

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.*

	Yes	*No*
• Does the publication meet your needs?	□	□
• Did you find the material:		
Easy to read and understand?	□	□
Organized for convenient use?	□	□
Complete?	□	□
Well illustrated?	□	□
Written for your technical level?	□	□

- What is your occupation? ____________________________________

- How do you use this publication:

As an introduction to the subject?	□	As an instructor in class?	□
For advanced knowledge of the subject?	□	As a student in class?	□
To learn about operating procedures?	□	As a reference manual?	□

Your comments:

If you would like a reply, please supply your name and address on the reverse side of this form.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your 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.

Cut or Fold Along Line

Fold Fold

First Class
Permit 170
Endicott
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 G60
P. O. Box 6
Endicott, New York 13760

Fold Fold

If you would like a reply, *please print:*

Your Name ______________________
Company Name ______________________
Department ______________________
Street Address ______________________
City ______________________
State ______________ *Zip Code* ________

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/VS1 Planning and Use Guide (File No. S370-34) Printed in U.S.A. GC24-5090-1

GC24-5090-1

OS/VS1 Planning and Use Guide (File No. S370-34) Printed in U.S.A. GC24-5090-1

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)