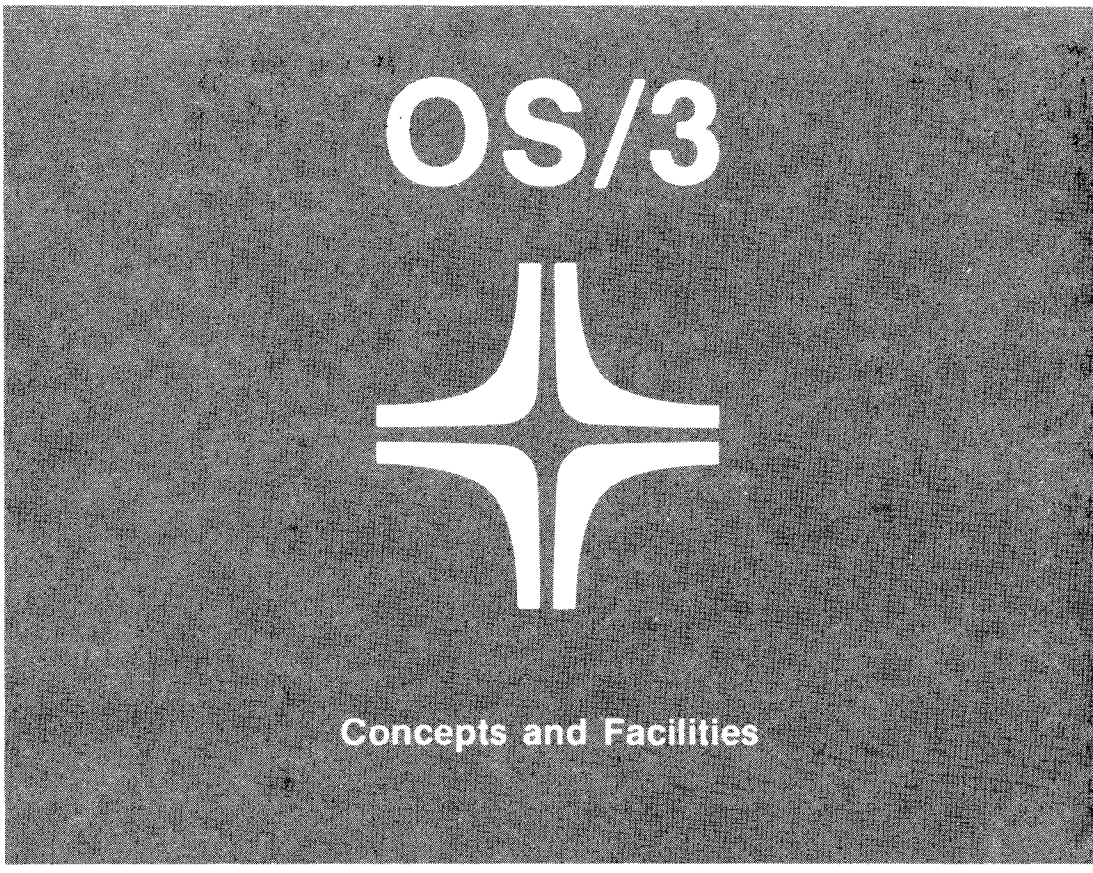
Supervisor



Environment: System 80

SPERRY

UP-8831 Rev. 2

This document contains the latest information available at the time of preparation. Therefore, it may contain descriptions of functions not implemented at manual distribution time. To ensure that you have the latest information regarding levels of implementation and functional availability, please consult the appropriate release documentation or contact your local Sperry representative.

Sperry reserves the right to modify or revise the content of this document. No contractual obligation by Sperry regarding level, scope, or timing of functional implementation is either expressed or implied in this document. It is further understood that in consideration of the receipt or purchase of this document, the recipient or purchaser agrees not to reproduce or copy it by any means whatsoever, nor to permit such action by others, for any purpose without prior written permission from Sperry.

PRINTED IN U.S.A.

PAGE STATUS SUMMARY

ISSUE: UP-8831 Rev. 2
RELEASE LEVEL: 8.2 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		
PSS	1	
Preface	1 thru 3	
Contents	1 thru 3	
1	1 thru 5	
2	1 thru 12	
3	1 thru 9	
4	1 thru 33	
Appendix A	1 thru 3	
User Comment Sheet		

Part/Section	Page Number	Update Level

Part/Section	Page Number	Update Level

All the technical changes are denoted by an arrow (⟹) in the margin. A downward pointing arrow (⇓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (⇑) is found. A horizontal arrow (⟹) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Preface

This manual describes the concepts behind the SPERRY Operating System/3 (OS/3) Supervisor and the facilities the supervisor makes available to OS/3 users. It presents an overview of the supervisor that is intended for two audiences: high-level language programmers and site administrators. The manual is organized as follows:

- Section 1. Basic Supervisor Concepts

  For both audiences. Introduces the supervisor, describes its functions in terms of tasks and multitasking, and outlines its interfaces with user software.

- Section 2. Supervisor Features and Functions

  For both audiences. Describes supervisor features by functional categories.

- Section 3. Supervisor Features and High-Level Languages

  For high-level language programmers. Builds upon the list of features in Section 2 by listing the languages that can call upon each feature: FORTRAN, COBOL, RPG II, BASIC, ESCORT programming language, and job control language (JCL).

- Section 4. Generating an OS/3 Supervisor

  For site administrators. Builds upon the list of features in Section 2 by listing the supervisor generation (SUPGEN) parameters that can be used to generate and modify a working supervisor. In some cases, parameters are shown with guidelines to their use, including their costs in terms of main storage.

- Appendix A. Statement Conventions

  For both audiences. Describes the conventions used to delineate configuration parameter values.

This manual is intended as a general discussion of those supervisor features you directly or indirectly call on, rather than as a detailed guide to all supervisor functions. (Some parts of the OS/3 supervisor, such as spooling and job accounting, are priced separately and therefore may not necessarily be installed in your system.) After reading this manual, you should have a good idea of:

- what supervisor features are available through the OS/3 high-level languages; and
- the supervisor configuration, specified at system installation time, that best suits your needs.

For a detailed guide in using supervisor services, refer to the supervisor macroinstructions user guide/programmer reference, UP-8832 (current version).

Other current OS/3 publications referenced in this manual that are helpful when using the supervisor are:

- System service programs (SSP) user guide, UP-8841

  Describes various system utilities (e.g., librarian, linkage editor).

- Consolidated data management concepts and facilities, UP-9978

  Describes the organization and record formats of various file types.

- Basic data management user guide, UP-8068

  Describes the effective use of OS/3 basic data management.

- Interactive services concepts and facilities user guide/programmer reference, UP-9972

  Describes the commands and operating procedures for workstation terminals.

- Spooling and job accounting concepts and facilities, UP-9975

  Describes basic spooling and job accounting concepts and options available to control spooling systems.

- System installation user guide/programmer reference, UP-8839

  Describes the procedures necessary to install, tailor, and maintain OS/3 software in a System 80 environment.

- Security management user guide/programmer reference, UP-9994

  Describes OS/3 system security to security administrators working in a System 80 environment.

- Security maintenance utility user guide/programmer reference, UP-8823

  Describes OS/3 system security to security administrators working in a System 80 environment.

- Operations handbook operator reference, UP-8859

  Describes system operator procedures.

- System activity monitor user guide/programmer reference, UP-9983

  Describes the use of the system activity monitor (SAM) for evaluating system performance.

- 1974 American National Standard COBOL programmer reference, UP-8613

  Describes 1974 ANS COBOL for the applications programmer.

- FORTRAN IV programmer reference, UP-8814

  Describes FORTRAN IV for the applications programmer.

- Report Program Generator II (RPG II) user guide, UP-8067

  Describes RPG II for both novice and experienced applications programmers.

- BASIC programmer reference, UP-9168

  Describes BASIC for the applications programmer.

- ESCORT programming language user guide, UP-8855

  Describes ESCORT programming language for applications programmers.

- Job control user guide, UP-9986

  Describes the job control language used under OS/3.

Contents

4. GENERATING AN OS/3 SUPERVISOR

1. Basic Supervisor Concepts

1.1. GENERAL

The SPERRY Operating System/3 (OS/3) Supervisor is a package of routines that form the heart of OS/3. It is the supervisor that allows other parts of OS/3 to work together and makes possible such useful OS/3 features as multijobbing and spooling.

As far as your user programs are concerned, the supervisor has two main functions:

- It interacts with user programs and symbionts to provide the services and control they need.
- It acts when necessary to handle randomly occurring external events, such as errors, that might otherwise disrupt a user program or even the entire system. Thus, an error occurring in one job may cause only that job to be terminated, leaving other jobs unaffected.

The supervisor is built around executable modules, or routines, each of which has a specialized function. Those routines commonly used by the supervisor always reside in main storage. Other less often used routines, called transients, are stored on the SYSRES volume and are loaded in main storage only when the supervisor needs them. This arrangement promotes supervisor efficiency: it minimizes the amount of main storage the supervisor uses by overlaying unneeded transients with newly loaded transients, and it eliminates the input/output time needed to load the most commonly used routines by keeping them resident.

1.2. TASKS AND MULTITASKING

The OS/3 supervisor does much of its work through the use of tasks. A task is the basic unit of work that can compete with other tasks for control of the central processor. By *processor control* we don't mean that the task executes machine code by using the processor but, rather, that it acts as a logical point of control for a physical sequence of executable machine code – a program – that does use the processor.

To grasp the difference between a task and a program, picture a system in which only one program at a time were allowed in main storage. In this case, no task control would be needed because the program would execute and terminate without interruption. But most programs do not continuously use the processor: you may, for example, request input or output, in effect leaving the processor idle until the I/O channel signals that the I/O operation has finished, at which time the program could resume use of the processor.

If more than one program were allowed in main storage, however, processor time could go from an idle program to one that isn't idle and later return to the first program when it is ready to resume processing. In OS/3, up to 256 such programs can exist concurrently in any one job; to prevent confusion and ambiguity, each program is controlled by a task. This ability to juggle processor control among tasks, maximizing processor use, is called multitasking.

Programs run under tasks as follows:

- Each user job step and symbiont runs under control of one task, called the primary task.
- Each transient runs under control of a task.
- Other system facilities, such as spooling and the integrated communications access method (ICAM), run as tasks.

The only OS/3 routines that do not run under task control are critical supervisor routines, such as error handling, and those routines that themselves manage tasks.

Since only one task at a time can control the processor (meaning the processor executes the code that task defines), some mechanism is necessary to take processor control away from one task and give it to another. In OS/3, that mechanism is the switcher, a critical supervisor routine that maintains a list, called the switch list, of all the tasks currently existing in the system. The switcher can take processor control away from a task and give it back later without disrupting the task's program and can do this as many times or as often as necessary.

So far, we have outlined the basic requirements for multitasking: the ability to create and maintain multiple tasks and a means of allocating processor time among available tasks. But we need more, as the next subsection explains.

1.3. CONTROLLING MULTIPLE TASKS

If all tasks in an OS/3 system were completely independent, all would compete for processor time on an equal basis. The system would quickly become unusable, though, because OS/3 tasks are not independent. For example, task A requests that task B do some work for it before it can continue. If both tasks continue to compete for processor time, what prevents task A from resuming processor control prematurely, before task B finishes? OS/3 solves this and related problems by using the task control block (TCB).

A TCB exists for each task in the system. It contains the physical, nonexecutable data that corresponds to a task's logical control of a program. It defines a task and acts as the task's interface with other tasks and the supervisor. By defining the task and its limits, the TCB prevents it from being disrupted by other tasks in the system, helping to make multitasking possible. At least one TCB exists for each user job (in the job prologue), and a TCB exists for each active transient routine.

When the switcher transfers processor control from one task to another, it uses information contained in the two TCBs involved. The switcher stores the problem registers and program status word (PSW) associated with the first task in its TCB, then loads the registers and current PSW with the same type of information contained in the second TCB, effectively allowing the second task to pick up execution exactly where it left off earlier.

Other control information contained in a TCB can be set to cause the switcher to ignore its task, in effect suspending that task or making it ineligible for processor control. A task remains in this suspended condition until the same control information is reset, causing the task to become active or eligible once again for processor control. While a task is suspended, all other active tasks continue to compete for processor time through the switcher. Often a task is suspended until some needed action is taken; only then is the task's control information reset, and only when that task is ready to resume processor control. It is the concept of task suspension that allows the supervisor to control task access to the switcher and thus to keep task competition from going out of control.

The supervisor has one other mechanism for controlling tasks, that of priority levels. When created, a task is assigned a priority. The switcher, when looking for a task to which it can transfer processor control, scans active tasks from those with the highest priority on down, selecting the active task with the highest priority to get processor control. System tasks (like transients) run at a higher priority than user jobs. Running systems tasks at a higher priority does not adversely affect user jobs, which, in any case, must call on them for various services. Also, system tasks running at a higher priority can begin and finish more rapidly, thus actually improving overall system efficiency.

1.4. USER PROGRAM INTERACTION

The supervisor's ability to maintain multiple tasks serves it well in interfacing with user programs. Since most supervisor services run as tasks themselves, the most common sequence of events is as follows:

1. A user program running under control of a primary task called, for example, task A, calls a transient routine to perform some system service. The supervisor responds by loading the transient in a portion of main storage set aside for transients and establishing the transient as a task with its own TCB called, say, task B.

2. The supervisor suspends task A and sets control information in the task's TCB that marks the task as suspended and unavailable.

3. After task A gives up processor control, the switcher scans the switch list for the highest priority task awaiting control. It may give control to task B or to some other active task but, task A being marked unavailable for the moment, the switcher will not return control to that task.

4. Eventually, the transient routine controlled by task B finishes. Only at this point does the supervisor reset the control information in task A and restore it to active status, task B having finished its work.

1.5. EXTERNAL EVENT INTERACTION

Another use of multitasking is as a response to external events. These events are called interrupts because they interrupt normal processor flow and must be handled in some way before processing can continue. OS/3 recognizes eight types of interrupts:

- Supervisor call – occurs in response to the SUPERVISOR CALL (SVC) machine instruction. Though it is handled as an interrupt, your programs routinely use the supervisor call to request supervisor services, as described in 1.4.

- Exigent machine check – indicates a malfunction in or around the processor. If the malfunction is in the supervisor area, the system is halted. If the exigent machine check indicates an error in the memory area of a job region, the system continues to run, though that memory area is marked as not usable and the job is terminated.

- Repressible machine check – indicates a malfunction in or around the processor from which recovery is possible.

- External interrupt – generated either by the processor interval timer or the system console interrupt key.

- Program check – occurs when the processor attempts to execute a nonexistent instruction or to execute an existing instruction in an illegal manner.

- Program event recording (PER) – provides dynamic monitoring of executing programs by storing information about the current instruction whenever a specified event occurs.

- Input/output – occurs in response to signals from I/O channels.

- Restart – occurs when the restart key on the system console is pressed and can be used to put a stopped processor in the operating state.

Some interrupts, like the supervisor call or input/output, are routinely encountered; others, like program or machine checks, represent serious errors that the supervisor must handle with minimal system interruption. As you will see in the following sections, the supervisor can handle a wide variety of interrupts and program requests.

1.6. OVERVIEW OF SUPERVISOR FEATURES

The services provided to your programs by the supervisor fall into the following categories:

- Task management
- Program management
- Input/output related services
- Spooling
- Logging and accounting
- Diagnostic services

2. Supervisor Features and Functions

2.1. GENERAL

This section presents an overview of the supervisor facilities available with your OS/3 system. Those features described here can either be used through OS/3 high-level programs or be specified at system installation time. For a detailed discussion of how high-level languages use the supervisor, see Section 3. For a detailed discussion of system installation options as they relate to the supervisor, see Section 4.

2.2. PROGRAM INITIATION AND LOADING

Before any of your programs can run, the supervisor has to load it in the portion of main storage allocated to your job by job control. The supervisor initiation and loading facility takes information from the job prologue (tables located in the low-order portion of your job region) that helps it determine where to load your job. The facility then locates your program as a load module on disk, loads it in main storage at the proper location, links the user job step – now a primary task – at the proper execution priority, and passes control to the task switcher.

In some high-level languages, your program/load module can load other load modules whenever needed, often as overlays. In these cases, it is your program that directs the loader how and where to load the new module.

Within a loaded program, the supervisor relocates, if necessary, every address constant the program contains. This lets job control load your program anywhere in your job region while ensuring that it will run correctly. Relocation is not limited to job control; where the facility is available to high-level user programs, it also performs relocation.

The program initiation and loading facility is always included in your system.

2.3. PROGRAM TERMINATION

The supervisor has facilities to end a job step in an orderly way even if the system detects errors in the step. With these facilities, the supervisor causes the system facilities assigned to a job or to a task to be relinquished for assignment to other jobs or to other tasks. Program termination falls into two categories, normal and abnormal termination:

- Normal termination implies normal completion of the job step and continuation of the job. The supervisor allows all task and I/O functions to idle down prior to terminating the program. Normal termination usually occurs when the primary task controlling the job step detaches itself from the switch list after completion of the step. Under some high-level languages, though, you can code a program to terminate the job step under which it runs at any point in its logic. In some cases, too, you can cause the supervisor to generate a main storage dump of your job region upon normal termination of a job step.

- Abnormal termination implies that a system-detected error has occurred from which the program cannot recover. In this case, the supervisor detaches the primary task, delinks all outstanding I/O, and waits for all outstanding system functions to be completed. It provides a printout of the contents of the job region if one of the DUMP options was specified on the OPTION statement and if there is a printer assigned to the job or there is a printer available.

High-level programs usually generate the proper machine language to handle most of the errors your program may encounter (see 2.6). You do have some options available to let you handle errors in other ways, depending on the path your program logic takes.

Normal program termination is always included in your system. With the exception of SYSDUMP (see 2.18), all abnormal termination facilities are always included, too.

2.4. TIMER SERVICES

Supervisor timer services fall into two types: the day clock and the interval timer. The day clock facility maintains the current time and date in a simulated day clock, which is contained in the resident supervisor. The interval timer generates a timer interrupt after a certain amount of time has passed. If interrupt timer island code (see 2.6) is active at the time, control then passes to it.

Most high-level languages get the current system date and time of day from the day clock, although only the system operator can change them. No interval timer service is available to high-level language programmers.

The full range of timer services is always included in your system.

2.5. PROGRAM LINKAGE

A program may consist of several phases or routines produced by an assembler, compiler, or other language translator and then combined by the linkage editor. Control can be passed from one routine to another within the program. This is referred to as direct linkage. Linkage can proceed through as many levels as necessary. During the execution of a job step, a routine (referred to as the calling program) passes control to another routine (the called program), which can in turn become the calling program passing control to a third routine (the called program), etc. This branch and linking process requires that the contents of certain registers be saved, then restored, so that control can be returned to the calling program.

Program linkage is available from all high-level languages, in conjunction with the linkage editor (see the system service programs user guide, UP-8841 (current version)).

2.6. ISLAND CODE LINKAGE

As you know, there are eight levels of interrupts in OS/3. Some of these interrupts are handled by system routines; however, there are four interrupts that some user programs handle themselves. These interrupts are:

1. Program Check – An operation in your program causes a program check interrupt, such as an addressing error, arithmetic overflow, or operation exception.

2. Interval Timer – A time interval elapses.

3. Abnormal Termination – An error occurs that makes continuation of your program impossible.

4. Operator Communication – The operator entered an unsolicited message at the system console or the workstation (the contents of register 1 at the time the island code gains control indicates whether the operator entered the message from the console or a workstation).

To handle these interrupts, all high-level languages generate closed routines called *island code*, which are linked to your program at execution time. When one of these interrupts occurs, the supervisor stores the contents of the program status word (PSW) and general registers and then transfers control to the appropriate island code routine. If the island code returns control to your program, the supervisor uses this stored information to return control at the point the interrupt occurred.

The purpose of the program check, interval timer, and operator communication island code routines is to handle program contingencies or to notify your program that the interrupt has occurred. In the case of abnormal termination, the function of the island code routine is to terminate either a task or a job step rather than the entire job (normal procedure for abnormal termination if there is no abnormal termination island code routine).

You cannot, generally, write your own island code for high-level language programs. There is little need to; the island code that high-level language processors themselves generate is capable of handling most interrupts in an orderly way.

Island code linkage is always included in your system.

2.7. SYSTEM INFORMATION CONTROL

Each user program is assigned a variable-length storage area within the job region; this area is called the job prologue. This area contains the primary task control block (TCB) and other information so critical that, to preserve system integrity, application programs can only read from or write to one 12-byte prologue field, called the communication region, used to pass information from one job step to the next. Within the communication region, high-level languages let you access only its last byte, called the user program switch indicator (UPSI) byte. In most high-level languages, you can access either the entire UPSI byte or individual bits within it. For more information on the UPSI byte, refer to the job control user guide, UP-9986 (current version).

2.8. CONTROL STREAM READER

The control stream reader allows you to access data that was entered into the system with the job control stream. This provides a convenient method to handle small quantities of input that would normally have been handled as a card or diskette file. Because the data is embedded within the job control stream, there is no need to define a card file, nor is a device assignment set required for the card reader.

This embedded data might consist of transactions or changes to be processed against a master file, source code, or control statements to be processed by a utility routine; or it might consist of PARAM job control statements to introduce parameters that can be used during program execution. Refer to the job control user guide, UP-9986 (current version) for a description of statements within embedded data.

When job control reads the job stream, it stores the embedded data in compressed form in the job's run library file: YRUN. During the execution of the job step, YRUN is read into main storage and may be accessed by instructions available in all high-level languages. Each requested record is expanded to its original form and stored in an input area you specify.

Except for PARAM statements, each retrieved record is an exact image of the source statement, which may be from 1 to 128 bytes. Thus, you can read 80-byte images from punched cards or 128-byte images from diskette.

NOTE:

Although PARAM and other job control statements may be handled as part of an embedded data set, they must still observe the job control statement conventions. Remember that job control statement information cannot extend past character position 71, and that position 72 is used to indicate continuation of a statement.

The control stream reader is always included in your system.

2.9. DISK AND DISKETTE SPACE MANAGEMENT

Space management comprises a group of routines that provide an efficient and completely automatic disk and diskette space accounting capability. These routines relieve your programs of the responsibility of knowing the precise contents of disk and diskette volumes. These routines also resolve competing demands for space allocation and establish standard interfaces with job control, utility, and service programs.

Space management routines perform the following functions:

- Allocate files
- Extend files already allocated (disk only)
- Scratch files that are no longer needed
- Rename files (disk only)
- Obtain label and extent information

Space management is strictly a supervisor function and so is not available to your high-level language programs. It is always included in your system for every disk or diskette you select at system installation time.

2.10. SYSTEM ACCESS TECHNIQUE

The system access technique (SAT) is a specialized block level device handler provided for both disk and magnetic tape files. Any interface between your high-level language programs and SAT is handled automatically, relieving you of the responsibility of managing SAT files. SAT is always included in your system.

2.11. MULTIPLE-INDEXED RANDOM ACCESS METHOD

The multiple-indexed random access method (MIRAM) is a disk access technique that lets you process a single disk file in several different ways. All disk files created by your high-level language programs are MIRAM files, and all language processors generate code that automatically handles file organization, relieving you of that chore. Since MIRAM is a data management function, refer to the consolidated data management concepts and facilities manual, UP-9978 (current version), for more information.

2.12. MULTIJOBBING

With its multijobbing features, the OS/3 supervisor can process concurrently up to 14 jobs for models 3, 4, 5, and 6 or 48 jobs for model 8, each job consisting of one or more job steps that are executed serially. As mentioned in 1.2, a switch list allocates processor time to these jobs based on task priorities (each step of each job representing a primary task), synchronization, and I/O usage. While one task is awaiting the completion of an external event (such as completion of an I/O request), the supervisor activates another task that is ready, thus making maximum use of the processor. Since most programs require support other than merely processing instructions, multijobbing provides you with an effective method to reduce processor idle time and increase system productivity.

Every job step submitted to OS/3 is established as a primary task. The switch list has the capacity to allow you to specify up to 60 levels of processing priority for tasks. The maximum number of task priority levels that the supervisor will recognize is established at system generation time. The technical limit is 60; however, a more practical number of 3 to 15 is sufficient to achieve a high degree of processor utilization. When a task is interrupted to perform external processing (external to the instruction processor), it frees the processor, and OS/3 searches the switch list for the highest priority task that is not waiting for an external event to be completed. This task could be in the same job or it could be from any other job currently being processed.

Multijobbing is always included in your system, and system installation options let you control the relative priorities of user and some system tasks.

2.13. MESSAGE DISPLAY AND LOGGING

Successful operation of a computer system requires frequent communication. You use job control statements and assembler instructions to tell the CPU what to do, and how and when to do it. The operating system tells the operator what to do and tells you what was done and when. The operator gets a message from the supervisor (or from you) and answers a question or performs an action.

OS/3 provides several methods by which you can communicate with the operating system and with the console operator. These consist of a system log, display to the operator, and display to the workstation, which can be used singly or in combination.

A system log file is maintained by the supervisor spooling function. Job logs are subfiles of the system log file and receive all log and accounting information for the job, including messages you write to the log by using instructions in your program. Other logs maintained by spooling include a console log for the console workstation and additional logs, one for each active workstation.

You can display a message to the operator at the system console. The message may be for information only, or you may request a reply by the operator. Also, you can combine a log entry and a display. In this case, the message displayed and any reply from the operator are written to the system log and also to a console log if one is configured at system generation. In addition, you can display data to or receive data from a workstation.

Not all of these facilities are available to all high-level language programmers; see 3.2.11 for details.

2.14. INTERACTIVE SERVICES

In addition to the message display facilities discussed in 2.13, OS/3 provides you with the means to make even more extensive use of system workstations through interactive services. These services are discussed in the interactive services concepts and facilities user guide/programmer reference, UP-9972 (current version). Briefly, these services include:

- Interactive job control
- Interactive data utilities
- General editor
- Screen format services
- Interactive command set

2.15. SPOOLING

Spooling is the technique of buffering data files for low-speed input and output devices to a high-speed storage device independently of the program that uses the input data or generates the output data. Data from card readers or from remote sites is stored on disk for subsequent use by the intended program. Data output by the program is stored on disk for subsequent punching or printing. The spooling function also handles diskette files. It treats input from diskette as though it were from a card reader and output to a diskette as though it were a card punch. In this description of spooling, any reference to a card reader, card input, or card file also includes diskette input; any reference to a card punch, card output, or card file also includes diskette output.

Spooling enhances system performance:

- by releasing large production programs and system software from the constraint of the slower speed devices, thereby freeing the main storage occupied by these programs sooner; and

- by driving the slower speed devices at their rated speed on a continuous basis, thereby making full use of the devices during the time that is normally lost to systems overhead or to job steps not using printers.

In addition to buffering data files, spooling maintains, for each job in the system, a job log that contains log and accounting information for the job. Spooling also maintains individual logs for the console workstation and each additional active workstation in the system; each of these logs records the system mode communications between the system and a console or workstation during a work session.

The spooler is the hub of the spooling facility and is a part of the supervisor. It provides record level input and output to and from the spool file for each element in the system needing access to that file. It intercepts all input/output commands to the virtual printer, punch, and card reader (which, to your programs, look no different from their real counterpart devices) and accesses the high-speed storage device, the disk, when necessary. Besides handling program input and output spooling, it also holds the system log, which contains job and accounting records for each job in the system.

Besides the spooler, the spooling facility also includes:

- the spool file, on which the spooler stores all spooled input and output;

- the input reader, the spooling element that reads input to be spooled from a card reader; and

- the output writer, the spooling element that writes spooled output from the spool file to a printer, or card punch, and writes print or punch output to a tape, disk, or diskette for temporary storage (redirected output).

Spooling is a supervisor option, and at system installation time you may specify options associated with it. For more information on spooling, refer to the spooling and job accounting concepts and facilities manual, UP-9975 (current version).

2.16. PRINTERLESS SYSTEMS

When the OS/3 system has a spooling facility, it can be configured and used without a physical printer. To achieve this, supervisor initialization is modified to recognize an "indirect" printer. Rather than producing output to a printer, such a system generates printer files. These printer files can be used for printing on an OS/3 system that does have a physical printer. For more information, refer to the system installation user guide/programmer reference, UP-8839 (current version).

2.17. JOB ACCOUNTING

The job accounting package consists of resident routines that are linked with the supervisor and elements of the job step processor at system installation time. These routines provide a count of the facilities utilized by each job step during its execution within the system. The message logging facility of the spooling function transfers this data from main storage to disk as part of the output spoolfile. The output writer prints the job step and job values as part of the normal message log output for each job. Optionally, the output writer can write the accounting information to a standard magnetic tape file or SAT disk file for offline processing by user-developed accounting routines or by OS/3 data utility routines. You can assign an account number by using the JOB job control statement that is carried along with the accounting records. This enables you to accumulate statistics from the disk or tape file for computer time and resources charged against an account number, which could represent a project, department, cost center, etc.

Data collected by job accounting for each job step and for the entire job includes the time of usage for the central processing unit, the wall clock duration of the job steps, the number of SVC calls, the amount of main storage allocated and actually used, the number of transient calls, and the number of device calls. Job accounting is a part of spooling, and so is available only when spooling is configured.

2.18. DIAGNOSTIC AND DEBUGGING FACILITIES

The supervisor has several facilities to help you find and remove errors from your programs. These occur in the following categories:

- Main Storage Displays

  These include snapshot dumps of selected main storage locations as well as dumps of whole job regions (EOJ dumps and JOBDUMPs) and even the whole of main storage (SYSDUMP). The snapshot dumps are available to some high-level language users; the job and system dumps, to all users. System dumps are an option you can include at system installation time, while job region and snapshot dumps are always included.

- Checkpoint/Restart

  Hardware and software malfunctions can cause your job to terminate before its normal completion. Another reason for termination could be that the operator cancelled your job because a high-priority job required all the facilities of the computer. If the job is small, you can rerun it without any really great loss. But, what if it is a long or complex job, where rerunning the job could increase both processing time and cost, thereby reducing productivity? OS/3 has provided the checkpoint facility, which allows you to periodically generate records containing the operational status of your job. Each such record is called a *checkpoint* and is written to a checkpoint file on disk, format label diskette, or magnetic tape. Should a job step fail, you can use the last checkpoint written before the failure to restart the job at the point where the checkpoint was taken rather than starting the entire job all over again.

You might want to create a checkpoint record at some specific occurrence, such as the end of a magnetic tape reel in a multivolume input file, or after processing a specific number of records.

The capability to generate checkpoint records is a function of the supervisor, and the capability to use these checkpoint records to restart a job is a function of job control (through the RST job control statement).

The checkpoint facility is available only to COBOL high-level language programmers. It is always included in your system.

- Monitor/Trace

  One means of debugging a program is the monitor/trace supervisor facility. This routine monitors each instruction in a program before and after it is executed and then signals if an event has occurred, such as:

  – a specified main storage location has been accessed; or

  – a specified location has been reached.

  How the supervisor reacts to these events is determined by the high-level language that calls the monitor. You may get a printout of the locations you specify and the option of continuing the program or terminating it. Not all the monitor's capabilities are available to all programs; see 3.2.13 for details. The monitor/trace facility is always included in your system.

- System Debugging Aids

  The supervisor has several debugging aids that can help identify system problems a system dump by itself cannot uncover. These include a supervisor debug option that monitors the entire operating system; a console debug option that halts processing on an I/O, transient, or loader error; transient, symbiont, or shared code halt routines that halt whenever a specified transient, symbiont, or shared code module is loaded; and a soft-patch symbiont that applies temporary patches to transients, shared code modules, object modules, load modules, and the resident supervisor. All these debugging aids require some knowledge of how the supervisor operates internally. All of them are always included in your system.

2.19. RESOURCE MANAGEMENT

The resource management facility is a supervisor feature that is dynamically tuned to the system environment to accommodate user personnel at all levels. This is a versatile facility encompassing use by security administrators, console operators, and workstation users.

Resource management permits both the security administrator and the console operator to control system use (e.g., memory assignment, device assignment, and CPU dispatching priority) through various resource parameters; these parameters are specified during the system generation process. Commands are available that allow the security administrator or console operator to change these values during the operation of the system.

2.20. SECURITY MANAGEMENT

The security management facility is a supervisor feature that includes:

- Profile Management

  Deals with user, execution, and program profiles stored in the system dictionary. Various commands such as CREATE, DISPLAY, ALTER, PURGE, etc, allow authorized personnel (security administrator and/or the individual user) to establish limits or restrictions as an added dimension of security for managing profiles.

- Security Logon Service

  Used to identify your attempt to gain access to the system. Logon denies system access to those interactive and batch users not authorized by a process that checks the system dictionary for valid user-ids, passwords, and account numbers. (Both password and account number input are system generated.) Once you have access to the system, logoff will end your session.

Refer to security management user guide/programmer reference, UP-9994 (current version), or security maintenance utility user guide/programmer reference, UP-8823 (current version).

2.21. SHARED CODE MANAGEMENT

Many run-time modules run as shared code, which means that one module can be used by several jobs simultaneously. This feature not only saves main storage space but also saves the time it would otherwise take to load a separate copy of the module for each job that needed it. All shared code modules are provided by Sperry and are included in the library file YSCLOD on your system resident volume. These modules include screen format services, interactive services, data management, the general editor, the RPG editor, BASIC, and other system programs.

Shared code management is available for system programs only, not for application programs. The system administrator can, at his discretion, make certain shared code modules, or groups of modules, resident. A list of resident modules can be specified at system generation time, and this list can be modified at each subsequent IPL (initial program load). At IPL time, the supervisor allocates main storage for the modules in the modified list, loads them, and makes them resident. Once a shared code module becomes resident, it remains resident until the next IPL.

Shared code management is always available in your system. System installation parameters are available to modify shared code for improved performance; for more information on specifying resident shared code modules at IPL time, see the operations handbook operator reference, UP-8859 (current version).

2.22. DYNAMIC BUFFER MANAGEMENT

The dynamic buffer management facility is a supervisor feature that can dynamically allocate extra main storage to system routines needing it. With this facility, system routines need not wait to obtain buffers that they need to continue processing. This in turn enhances the performance of your user programs, especially those using interactive services.

Because dynamic buffer management is limited to system routines, user programs have no direct control over it. You can, however, specify parameters at system installation or IPL time for its most efficient operation.

2.23. SYSTEM MONITORING FACILITIES

Sperry makes available to you facilities for recording hardware malfunctions and monitoring your system's normal activity. These facilities are the error log and the system activity monitor.

The error log is a file on the system resident volume (identifier YELOG) that contains records of hardware errors kept for the statistical and historical use of Sperry customer engineers. When a hardware error occurs, the error logging function, which is a part of the supervisor, stores pertinent information in the error log. (Models 3, 4, 5, and 6 require at least three resident error log buffers, while model 8 requires at least six resident error log buffers.) Sperry customer engineers can read these records from the file into main storage for processing and storage as permanent records, using these records in maintaining customer equipment. Error logging is always included in your OS/3 system.

Another useful system monitoring facility is the system activity monitor (SAM), a system symbiont that monitors and records your system's activity. It is intended for use by the system administrator and installation manager to aid in the detection of production bottlenecks, to optimize production job mixes, and to identify and change system variables that influence system performance.

For more information on the system activity monitor, see the system activity monitor user guide/programmer reference, UP-9983 (current version). You may optionally include the system activity monitor in your system at system installation time.

3. Supervisor Features and High-Level Languages

3.1. GENERAL

This section outlines those supervisor features that you can take advantage of through the high-level languages OS/3 provides. In the following discussion, each feature has a list of its availability in the following high-level languages:

- COBOL
- FORTRAN
- RPG II
- BASIC
- ESCORT
- JCL

For any feature, only those languages are listed that make use of that feature along with the specific instructions or directives needed. Refer to the Preface for a complete list of appropriate manuals with more information about a feature.

The COBOL compiler conforms to the specifications of the *American National Standard COBOL, X3.23–1974* and contains extensions, many based on supervisor facilities, that enhance the capabilities of COBOL beyond the basic requirements of the standard. The FORTRAN compiler accepts FORTRAN IV, which includes the *American National Standard FORTRAN X3.9–1966* and the IBM System 360/370 DOS FORTRAN IV languages as subsets. JCL is included in this discussion together with the languages because it, too, takes advantage of many supervisor features.

3.2. SUPERVISOR FEATURES AND LANGUAGE USE

3.2.1. Program Initiation and Loading

- COBOL

  Available through the CALL instruction.

- FORTRAN

  Available through the CALL FETCH and CALL LOAD instructions.

- BASIC

  Programs residing in an OS/3 library file may be loaded into your BASIC work space prior to execution by means of the OLD command, or dynamically loaded at run time using the CHAIN statement. A library file may be added to the contents of the work space with the MERGE command.

- JCL

  Used with the // EXEC statement

3.2.2. Program Termination

- COBOL

  Normal termination of dynamically loaded programs is available through the CANCEL instruction. Abnormal termination by the user is not available.

- FORTRAN

  Normal termination is available through the CALL EXIT instruction (without a dump) or the CALL DUMP instruction (with a dump). Abnormal termination by the user is not available.

- ESCORT programming language

  Normal termination from ESCORT language is available by use of the EXIT selection on the menu; this returns control to interactive services. Abnormal termination by use of the EXIT selection is also available if ESCORT language is run in supervisor debug mode. In this case, you are given the option of obtaining a system dump.

- JCL

  Program termination is not a JCL function, but JCL allows you to generate one of several types of dumps if abnormal termination occurs, no matter what language the program was written in. For the dumps, you can use the // OPTION DUMP, // OPTION JOBDUMP, // OPTION SYSDUMP, // OPTION GDUMP, // OPTION GJOBDUMP, and // OPTION GSYSDUMP statements. In addition, you can specify // OPTION statements that generate a program check if binary overflow (BOF), decimal overflow (DOF), significance (SIG), or exponent underflow (XUF) errors should occur.

3.2.3. Timer Services

- COBOL

  You can access the system's calendar date, Julian date, and time by using the ACCEPT statement.

- RPG II

  You can access the system's calendar date and time by using the TIME operation.

- BASIC

  You can set a processor time limit for your currently running program with the TIME statement, and get the elapsed running time for a program with the TIME function. In addition, you can access the system's calendar date with the DAT$ function and the current time of day with CLK$ function.

- ESCORT programming language

  You can access the system's calendar date using the DATE$, DAY$, MONTH$, and YEAR$ utility fields. The current time is available by use of the TIME$ utility field.

- JCL

  You can set a maximum execution time for your job by using the max-time parameter in the // JOB statement. Within the WRTBIG job procedure, you can print the date and time your job was run. And, to temporarily change the system's calendar date until the end of a job, you can use the // SET DATE statement.

3.2.4. Program Linkage

- COBOL

  You can call a subroutine that is linked to your program and pass data to and from it by using the CALL statement.

- FORTRAN

  You can call a subroutine or function linked to your program and pass data to and from it. For a subroutine, you use CALL and SUBROUTINE statements in the calling and called programs, respectively. For a function, you use the function name and FUNCTION statements, respectively.

- BASIC

  You can call a subroutine with the CALL statement and define parameters to be passed with a SUB statement entered as the first statement of the subroutine. The subroutine can reside in the calling program file or it can reside in an OS/3 library file named with the LIBRARY statement.

- RPG II

  You can call an internal subroutine, one associated with the calling object module, by using the EXSR, BEGSR, and ENDSR operations. You can call an external subroutine, one compiled separately from, but linked to, the calling program, by using the EXIT, BEGSR, and ENDSR operations. To pass data to and from an external subroutine, you use the ULABL and RLABL operations.

3.2.5. Island Code Linkage

- FORTRAN

  You can write routines to which program control passes if two types of program exceptions are found. These are CALL OVERFL for an arithmetic overflow or underflow and CALL DVCHK for a divide check. Under no other conditions can you write or access any island code.

3.2.6. System Information Control

- COBOL

  You can change or test the UPSI byte either as a whole or bit by bit by using the SYSSWCH clause. You can also change or test the entire communication region by using the SYSCOM clause.

- FORTRAN

  You can test portions of the UPSI byte by using CALL SSWTCH. The remainder of the communications region is inaccessible to your program.

- RPG II

  You can test or change the UPSI byte by using indicators U1 through U8 in your program. You cannot change or test the remainder of the communications region.

- JCL

  You can change the UPSI byte with // SET UPSI and the communication region with // SET COMREG. In addition, you can test the UPSI byte by using // SKIP.

3.2.7. Control Stream Reader

- COBOL

  You can access the control stream by using the ACCEPT statement and the SYSIN clause.

- FORTRAN

  Most card input to your programs comes through the control stream reader. To access it, you use the READ statement for I/O units 1 or 5.

- RPG II

  You can access the control stream reader with a file description specification statement specifying device type CTLRDR.

3.2.8. Disk and Diskette Space Management

- COBOL, FORTRAN, RPG II

  You cannot directly use the space management facility in a user program; this facility is a data management function. See the consolidated data management concepts and facilities manual, UP-9978 (current version), for more information.

- JCL

  You allocate and extend space for a disk or diskette file by using the // EXT statement.

3.2.9. System Access Technique

- JCL

  You can allocate space for a SAT file by using the // EXT statement. You will usually use this statement to allocate files that are solely managed by system programs. Examples of these are user library files handled by the SAT librarian and checkpoint files handled by the checkpoint/restart function (3.2.13).

3.2.10. MIRAM

You can access MIRAM files from your programs. For more information, refer to the consolidated data management concepts and facilities manual, UP-9978 (current version). BASIC can read a MIRAM file written in any format but will write only unkeyed records; see the current version of the BASIC programmer reference, UP-9168, for more information.

- JCL

  You can allocate space for MIRAM files by using the // EXT statement.

3.2.11. Message Display and Logging

- COBOL

  You can send messages to the system console by using the SYSLOG clause and send messages with operator replies by using the SYSCONSOLE clause. In addition to the above facilities, you can display messages on workstations by using one of the following clauses:

  – SYSTERMINAL – for the master workstation

  – SYSWORK – for the workstation assigned to your program through JCL

  – SYSFORMAT – for the workstation associated with screen format services

- FORTRAN

  For a STOP or PAUSE instruction, you can specify a decimal integer that is displayed on the system console when the instruction is executed. PAUSE displays the integer and stops program execution until the operator continues it; STOP displays the integer and terminates the program.

  FORTRAN allows you to treat a workstation as an I/O device; consequently you can move data into and from it with the usual FORTRAN statements.

- RPG II

  You can send information to or get information from the system console by using the DSPLY operation and a file description specification statement with the device name CONSOLE. With DSPLY, you can either display a message and continue processing or display a message and cause a halt until the operator resumes processing. During the halt, the operator may enter data that the program accepts when it resumes processing.

  You can send data to or receive data from a workstation by using screen format services and a file description specification statement with device name WORKSTN.

- JCL

  Your job can send messages to the system console or to any workstation. If you want an operator reply, you use // PAUSE; if not, you use // OPR.

3.2.12. Spooling and Job Accounting

- COBOL

  No program modifications are needed to use input and output files with spooling configured in your system. With spooling, you have the additional capability of copying SYSLOG and SYSCONSOLE messages (see 3.2.11) to your spool file, from which they can later be printed or even stored on disk or magnetic tape. In addition, you can output your own data, independent of any console display, to the job log for your job by using the SYSLST clause.

- FORTRAN

  With spooling, console messages displayed through the STOP or PAUSE instruction (see 3.2.11) may be copied to your spool file for later output to printer, disk, or magnetic tape. As for input or output files used by your programs, no modifications are needed to use spooling.

- RPG II

  With spooling, console messages and responses displayed through the DSPLY operation (see 3.2.11) may be copied to the spool file for later output to printer, disk, or magnetic tape. As for input or output files used by your programs, no modifications are needed to use spooling.

- BASIC

  With spooling, you can run BASIC in batch mode by spooling in a card file containing exactly the statements you would use in an interactive BASIC session.

- ESCORT programming language

  Spooling with ESCORT language involves only printer output. No modifications to your ESCORT program are needed to take advantage of spooling with printer output files.

- JCL

  The spooling facility stands at the center of several powerful JCL statements. You can, for example, use // SPL to output program data to a spool file on disk or diskette, // DST to send spool output to a remote device, or // DATA to load card images to the spool file for later input to a program. In addition, the parameters in the // JOB statement help you control spool output.

3.2.13. Diagnostic and Debugging Aids

- COBOL

  You can use the following supervisor features in your COBOL program:

  - You can get an EOJ dump, JOBDUMP, or SYSDUMP if your program goes through abnormal termination (see the JCL description).
  - You can use checkpoint/restart with the RERUN clause.
  - You can use debugging statements to monitor data items or procedures while your program is running.

- FORTRAN

  You can get an EOJ dump, JOBDUMP, or SYSDUMP if your program goes through an abnormal termination (see the JCL description). In addition, FORTRAN lets you include a variation of the PAUSE instruction in which an operator response of DUMP terminates the program with a dump.

- RPG II

  You can use the DEBUG operation to tell you what specified indicators and fields contain at various points in your program. You can get an EOJ dump, JOBDUMP, or SYSDUMP if your program terminates abnormally. In addition, you can tell RPG II to output a formatted error analysis dump if an error occurs in your program. Run-time facilities let you display or change data while the program is executing. The operator control feature lets the system operator take action if your program sets any halt indicators; the operator can continue program execution, bypass the remainder of the program cycle, or terminate the program altogether.

- BASIC

  BASIC facilities like the syntax checker handle program and run-time errors dynamically during the BASIC interactive session rather than through the supervisor.

- ESCORT programming language

  ESCORT facilities handle program errors dynamically during the ESCORT interactive session rather than through the supervisor. These facilities display error messages prior to run time, warning of illogical conditions in the program being coded.

- JCL

  You use the JCL options DUMP, JOBDUMP, or SYSDUMP to specify the type of dump to be taken should your program terminate. In addition, you use the // RST job control statement to rerun a COBOL program from a checkpoint.

4. Generating an OS/3 Supervisor

4.1. SUPERVISORS – OURS OR YOURS?

So far we've shown you what makes up a supervisor and how you can use its features in your own programs. Before you can use a supervisor, though, you have to have one to use. And you've got a choice:

- Your OS/3 system comes with its own ready-to-use supervisor, named either SY@BAS (models 3, 4, 5, and 6) or SY#BAS (model 8). With it you can usually begin normal system operations immediately. In fact, the IPL (initial program load) procedure that you use to start up your system calls one of these supervisors unless you specify another.

- You can generate your own supervisor embodying some or all of the features we've described. Supervisor generation is done by the SUPGEN phase of system installation, under the control of parameters you specify. These parameters and the whole system installation process are described in more detail in the system installation user guide/programmer reference, UP-8839 (current version).

You may be wondering whether to use our supervisor or generate your own (or more than one; that's possible too with system installation). SY@BAS and SY#BAS have both been designed to meet most of the processing requirements at an average OS/3 site. They may, however, be too big or too small for your needs: too big in having features you don't ordinarily use and can do without; too small to satisfy special requirements you may have.

In either of these cases, you may want to generate your own supervisor. If you do, the bulk of this section describes the parameters you use to define one. But even if you choose to use either SY@BAS or SY#BAS instead, you may change your mind later, so it could be worth your time now to find out what the capabilities of these two supervisors are. That way, you can get a better idea of what you might want to change later. You can find this information both in this section and in the system initialization section of the current software release description (SRD).

NOTE:

This section presents the supervisor-related parameters and defaults set in SUPGEN at system installation time. If you compare these values to the ones used in supervisor SY@BAS or SY#BAS, as described in the current SRDs, you'll find they often differ. The difference arises because SUPGEN default values are chosen for generating a small supervisor, while SY@BAS and SY#BAS parameter values have another aim altogether: a supervisor that can be used in a majority of processing environments.

SUPGEN parameters tailor a supervisor to your needs. Some parameters add modules that increase the main storage requirements of your supervisor, while other parameters merely set defaults for the supervisor to follow when it is in main storage and functioning.

The SUPGEN parameters are divided into the following functional categories:

- Supervisor identification
- Timer service
- Logging
- Job management
- Task management
- Main storage management
- Shared code management
- Input/output
- Hardware

SUPGEN parameters, presented individually in 4.2, are discussed here from the point of view of how they affect the supervisor; after reading this discussion, along with Sections 2 and 3, you should know what parameter values you wish to specify to tailor a supervisor to your needs. The actual procedure for generating a supervisor is discussed in the system installation user guide/programmer reference, UP-8839 (current version).

All parameters shown in 4.2 are keyword parameters that you code exactly as shown if you perform system installation as a batch job. If you perform system installation interactively, each question in the dialog that corresponds to a keyword parameter will display that keyword so that you can refer back to it in this manual.

NOTE:

If you wish to configure spooling, the SUPGEN parameters that pertain to spooling are shown in the spooling and job accounting concepts and facilities manual, UP-9975 (current version).

4.2. SUPGEN PARAMETERS

This subsection discusses the supervisor parameters, each in the following format:

- a brief description of the parameter;
- the values you can specify by that parameter;
- an estimate of main storage requirements, if applicable; and
- other SUPGEN parameters needed to use this parameter.

4.2.1. Supervisor Identification Parameters

SUPMOD Keyword Parameter:

`SUPMOD=supvrnam`

Specifies the name of a supervisor configuration that has been defined by a previous SYSGEN and saved in the system source library file (YSRC) of your SYSRES volume or the OS/3 release volume (OS/3REL). The established supervisor may be requested through the use of this parameter. If used, SUPMOD must be the first keyword specified in this section.

When this keyword parameter is submitted, all other SUPGEN parameters, with the exception of SUPVRNAM, are ignored, and a diagnostic message is written on the output listing.

SUPVRNAM Keyword Parameter:

`SUPVRNAM={supervisor name / SY$STD}`

where:

`supervisor name`

Specifies the name of the supervisor. The name can be a maximum of six characters. When used with the SUPMOD parameter, it renames the supervisor configuration copied from your existing SYSRES volume.

4.2.2. Security Management Parameters

SECURITY Keyword Parameter:

SECURITY={NO / YES}

Specifies whether security other than security logon services (SLS) is to be used. This parameter allows security for IMS files, IMS programs, IMS transactions, execution profiles, user profiles, and terminal ids. Terminal ids are validated at logon if ISLOGONSC = YES.

SECLOG Keyword Parameter:

SECLOG={NO / YES}

This parameter permits you to specify whether the security log facilities are available. If YES is specified, the ISLOGONSC parameter must equal YES.

ISLOGONSC Keyword Parameter:

ISLOGONSC={NO / YES}

where:

YES

Indicates that your system is to provide security for interactive users by checking its security files when a user attempts to log on. If you specify ISLOGONSC=YES, the user's identification must be a valid name in your security file. If you specify ISLOGONSC=NO, any user can log onto your system with any identification. For more information on interactive security, refer to the current version of the security maintenance utility user guide/programmer reference, UP-8823, or the security management user guide/programmer reference, UP-9994.

ISADMID Keyword Parameter:

ISADMID=administrator-id

Specifies a 1- to 6-character id that identifies your system's security administrator. Your security administrator is the only user who can add or delete other user-ids from your security file. The first character of the administrator id must be an alphabetic character followed by up to five alphanumeric characters. If you don't specify this parameter, the system defaults to no id and you will not have a system security administrator.

When using the security maintenance utility (ISLOGONSC=YES), you must specify an administrator-id using the ISADMID parameter. If you don't, interactive security will be enforced and no one – not even the system administrator – will be able to log onto the system.

PASSWORD Keyword Parameter:

```
PASSWORD={YES}
         {NO }
```

Specifies whether passwords are required during logon. If YES is specified, a password is required during user profile creation and alteration. Also, if specifying YES for this parameter, then ISLOGONSC must equal YES.

TERMWAIT Keyword Parameter:

```
TERMWAIT={0-255}
```

This parameter allows terminals and user profiles to be waited (locked out) when an invalid logon id is detected.

where:

0-255

Specifies the number of entries in the terminal status table. An entry in this table is required for each workstation (local and remote) and remote terminal that uses information management system (IMS) or interactive services (IS). The terminal status table entry number determines the maximum number of workstations (local and remote) and remote terminals that can simultaneously be waited when an invalid logon id is encountered. Use of this parameter requires that ISLOGONSC equals YES.

If the parameter is omitted or zero is specified, the terminals and user profiles are not waited.

FILELOCK Keyword Parameter:

```
FILELOCK={YES  }
         {SHARE}
```

This parameter applies to model 8 only and is used to specify the type of file lock system to be generated.

where:

YES

Enables your supervisor to maintain file lock capabilities for system files (Y) and user files prefixed with $LOK.

SHARE

Enables your supervisor to maintain file lock capabilities for all files. This includes user files with or without the $LOK prefix, as well as system files (Y).

4.2.3. Timer Service Parameters

MAXTIME Keyword Parameter:

```
MAXTIME={1-999}
        {0    }
```

Specifies the default time (in minutes) for the maximum time parameter on the // JOB job control card.

To specify MAXTIME, you must also specify spooling and job accounting.

IPCTIMEOUT Keyword Parameter:

```
IPCTIMEOUT={10-3600}
           {120    }
```

Allows you to specify the amount of time that the system waits before it gives up on a response from another host. You can select any interval between 10 and 3600 seconds, or accept the system default of 120 seconds.

UNATCONSOLE Keyword Parameter:

```
UNATCONSOLE={0-30}
            {0   }
```

Specifies whether hardware error messages that require an R/U (retry/unrecoverable error) or RC (retry/cancel) response are automatically answered by the operating system. Values from 1–30 specify time (in minutes) that elapses before the error message is answered automatically with either a C (cancel) or U (unrecoverable error) response. The default value 0 suppresses this feature.

MAXTYPE Keyword Parameter:

```
MAXTYPE={CPU  }
        {WALL }
        {NONE }
```

Specifies whether the max-time parameter on the JOB statement defines elapsed wall-clock time or job CPU time.

SYSTEMDATE Keyword Parameter:

```
SYSTEMDATE={DMY}
           {MDY}
           {YMD}
```

Specifies the system date format at IPL time.

where:

DMY
Defines dd/mm/yy as the date format.

MDY
Defines mm/dd/yy as the date format.

YMD
Defines yy/mm/dd as the date format.

4.2.4. Logging Parameters

ERRLOGBUF Keyword Parameter:

Models 3, 4, 5, and 6	Model 8
ERRLOGBUF={3-2400 / 3}	ERRLOGBUF={3-2400 / 6}

Specifies the number of resident, 80-byte buffers assigned for I/O error logging.

Error logging, which is always configured in your system, requires 1K bytes for its coding plus 280 bytes (minimum) for tables and either 3 error log buffers (models 3, 4, 5, and 6) or 6 error log buffers (model 8). Space requirements for additional buffers can be calculated with the formula (n x 80), where n is the number of additional buffers desired.

SAM Keyword Parameter:

SAM={NO / YES}

where

YES
Specifies that the system activity monitor is to be included in your system. Refer to system activity monitor user guide/programmer reference, UP-9983 (current version), for details.

4.2.5. System Management Parameters

DDP Keyword Parameter:

DDP={**NO** / YES}

Allows the selection of distributed data processing (DDP). YES must be selected if DDP is used.

DMGTMODE Keyword Parameter:

DMGTMODE={CDI / **MIXED**}

Allows you to specify the data management mode. This parameter applies to model 8 only.

where:

CDI

Specifies that your system uses only common data interfaces (CDI) for data management. CDI data management can access only SAT or MIRAM files.

MIXED

Specifies that your system operates in a mixed data management mode in which your programs can execute in either CDI or define-the-file (DTF) mode. Mixed data management can access all file types.

4.2.6. Job Management Parameters

JCREADWKS Keyword Parameter:

JCREADWKS={**NO** / YES}

Allows you to specify whether your workstation can initiate functions (RU, FI, and SI) that require a card reader.

WARNING

You must ensure that the cards you wish to read are actually in the card reader at the time you issue the command.

JOBACCT Keyword Parameter:

JOBACCT={**NO** / YES}

Specifies whether resident job accounting routines maintain a record of CPU time used by job and job step facilities, the number of I/O requests per device, the number of supervisor requests, main storage use, and transient function use. You may use this only if spooling is configured at SYSGEN. This parameter adds approximately 350 bytes to the resident supervisor in addition to 100 bytes in each job prologue.

JOBACCTREQ Keyword Parameter:

JOBACCTREQ={**NO** / YES}

Specifies whether a job must have an account number. It is recommended that YES be specified since users logging onto the system should log on with an account number. The account number can be specified as the eighth positional parameter on the JOB statement or by the OPTION ACCN job control statement. If YES is specified and an account number is not provided, the job is aborted and the user cannot successfully log on.

JOBSLOTS Keyword Parameter:

Models 3, 4, 5, and 6

JOBSLOTS={**1** / 1-14}

Model 8

JOBSLOTS={**1** / 1-48}

Specifies the number of jobs that are supported for execution in the system. The maximum number is 14 for models 3, 4, 5, and 6, and 48 for model 8. Users of model 8 should know that, although 48 job slots is the maximum for this system, the user has access to 47 with 1 job slot always reserved for interactive services. Once set by SUPGEN, this parameter cannot be changed or exceeded without generating a new supervisor. Thus, unless you wish to restrict the number of current running jobs, you should specify the maximum number.

MAXJOBS Keyword Parameter:

Models 3, 4, 5, and 6

MAXJOBS={Ø-14 / **14**}

Model 8

MAXJOBS={Ø-48 / **48**}

Specifies the maximum number of jobs that are scheduled in the system. The default values are 14 (for models 3, 4, 5, and 6) and 48 (for model 8) unless you have specified a different value for the number of job slots (JOBSLOTS=). In this case, the default value for this parameter will be the number of job slots previously specified.

NOTE:

JOBSLOTS is specified during system generation, while MAXJOBS is adjusted as the system runs.

MAXWSJOBS Keyword Parameter:

Models 3, 4, 5, and 6

```
MAXWSJOBS={0-14}
          {n   }
```

Model 8

```
MAXWSJOBS={0-48}
          {n   }
```

Specifies the maximum number of jobs initiated from workstations running concurrently in the system. The maximum values are 14 (for models 3, 4, 5, and 6) and 48 (for model 8) unless you have specified a different value for the number of job slots (JOBSLOTS=). The default value n is equal to one half of the number that was assigned to MAXJOBS.

MAXSWSJOBS Keyword Parameter:

```
MAXSWSJOBS={0-n}
           {1  }
```

Specifies the maximum number of jobs initiated from a single workstation running concurrently in the system where n is the value assigned to MAXWSJOBS.

JOBQUEREC Keyword Parameter:

```
JOBQUEREC={HOLD}
          {NO  }
          {YES }
```

where:

HOLD
: Specifies that at IPL time your system recovers all jobs in the job queue and puts them in hold status until you are ready to run them.

NO
: Specifies that at IPL time all jobs should be deleted from the job queue.

YES
: Specifies that at IPL time your system recovers all jobs that were scheduled and put in the job queue when the system last shut down.

This parameter determines which job queue recovery option becomes the default option when, at IPL time, the system requests operator action (RECOVER FILES?).

RECOVERDS Keyword Parameter:

```
RECOVERDS={NO  }
          {YES }
```

where:

YES
: Specifies that you have the option of either recovering disk space to retry a job or of terminating that job when the job step processor displays a warning that it cannot allocate sufficient disk space for a job. This option allows you to provide the extra disk space the job needs. If you take the default option, RECOVERDS=NO, the job step processor will simply terminate a job if sufficient disk space cannot be found.

RUNVSN Keyword Parameter:

```
RUNVSN={SYSRES}
       {SYSSPL}
       {vsn   }
```

Specifies the default placement of the system RUN device. You can override the default at IPL time regardless of what you specify here.

where:

SYSRES
: Specifies the use of the system resident volume as the RUN device.

SYSSPL
: Specifies the use of the first spooling volume as the RUN device (spooling must be configured).

vsn
: Specifies the serial number of the volume serving as the RUN device.

4.2.7. Log Record Parameters

SYSLOG Keyword Parameter:

```
SYSLOG={NO  }
       {YES }
```

Specifies whether job and workstation log files (L, A, and W records) accumulated in spool files will be transferred to user disk or tape files. If this parameter is not configured at SYSGEN, it can be requested by the operator from a console later. Refer to spooling and job accounting concepts and facilities, UP-9975 (current version). Remember, you must specify JOBACCT=YES to have A records.

CONSOLOG Keyword Parameter:

```
CONSOLOG={MIN  }
         {NORM }
         {MAX  }
         {NO   }
```

Specifies whether console log C records and workstation log W records are collected. The system collects a record of all communications between system, console (console log), and workstation (workstation log) in this buffer. Then the system copies it into the spool file when it fills the buffer area. This parameter also specifies the size of the main storage buffer area for records.

where:

MIN
Specifies a 304-byte buffer.

NORM
Specifies a 560-byte buffer.

MAX
Specifies a 1072-byte buffer.

NO
Is the default that specifies console and workstation logs are not to be recorded in the spool file.

CONPRINT Keyword Parameter:

```
CONPRINT={NO  }
         {YES }
```

Specifies whether console log (C records) are printed when a file is breakpointed.

RETAINLOG Keyword Parameter:

```
RETAINLOG={NO  }
          {YES }
```

Specifies whether console log (C records) are retained in a spool file after printing.

4.2.8. Task Management Parameters

TRANS Keyword Parameter:

TRANS={1-15 / **1**}

Specifies the number of transient areas that should be established in main storage. The more transient areas declared, the more jobs requiring transient areas can be operating concurrently. Each transient area is allocated 1200 bytes of main storage whether it is being used or not. For most multijobbing applications, three transient areas are sufficient; in virtually all instances, no more than four are warranted.

TRNWKAREA Keyword Parameter:

TRNWKAREA={**NO** / YES / 32-125}

Specifies whether your system generates a 65K byte (the standard size) or a 32K to 125K byte transient work area to keep the most recently used transient module in main storage. If specified, transients can be loaded into main storage directly from the work area instead of from your SYSRES volume. This reduces the number of I/O delays that occur, thereby improving performance. Specifying YES initializes the transient work area to 65K bytes at initial program loading (IPL).

RESMOD Keyword Parameter:

RESMOD=[SM$ASCKE][,SM$ATCH][,SM$DBS][,SM$GTPUT][,SM$LOCK][,SM$STXIT]
[,SM$TASK][NONE]

Specifies certain software modules that are to be included in main storage with the supervisor rather than called as transients. You may specify the subparameters in any order or in multiple calls of the keyword.

If, however, you are generating a supervisor for nine thousand remote (NTR), all the modules except the SM$GTPUT module (which is optional) are required.

where:

SM$ASCKE

Specifies that the assign key function used by information management system 90 (IMS 90) for secondary storage key assignment is resident. Making this function resident increases IMS 90 performance. This option adds approximately 700 bytes to the resident supervisor.

SM$ATCH
Specifies that transient routines are resident that create and activate a task requiring control of the processor. Used mainly for start-up and end task processing. Specifying this subparameter in a batch environment is not recommended unless heavy multitasking is used. This option adds approximately 1200 bytes to the resident supervisor.

SM$DBS
Specifies that the data base system (DBS) interface module is resident. This option adds approximately 300 bytes to the resident supervisor.

SM$GTPUT
Specifies that transient routines (GETCOM, PUTCOM, and GETINF) that handle job and system information are resident. This option adds approximately 700 bytes to the resident supervisor. Do not specify as resident unless a specific processing function requires it.

SM$LOCK
Specifies that the supervisor lock and unlock functions are resident. These functions provide the read and write lock capability for system resources. This option adds approximately 900 bytes to the resident supervisor.

SM$STXIT
Specifies that the transient routine is resident that creates, changes, or terminates linkage between the supervisor and island code subroutine. You should use this option only when running programs that purposely generate frequent program checks, such as emulators. This option adds approximately 1400 bytes to the resident supervisor.

SM$TASK
Specifies that multitasking transient routines are resident. These routines help gain efficiency in a multitasking environment. SM$TASK is automatically made a resident module if spooling is configured. This option adds approximately 1100 bytes to the resident supervisor.

If the parameter is omitted and spooling is not configured, the system automatically specifies a resident loader built into the system. If RESMOD=NONE and another subparameter is also specified or defaulted, NONE is ignored.

PRIORITY Keyword Parameter:

```
PRIORITY={1-60}
         {1   }
```

Establishes the number of task priority levels recognized by the supervisor and is used when the system has multijobbing capabilities. The maximum level is 60.

You can specify as little as one priority level, all jobs running at that level. Specifying more than one level, however, allows you to set primary tasks (and the jobs they control) to different priorities as the need arises.

SYMBPRI Keyword Parameter:

```
SYMBPRI={0   }
        {0-59}
```

Specifies the priority level at which system symbionts are to run (including the job scheduler, but excluding ICAM). If omitted, priority 0 is assigned.

You can use this keyword parameter to assign the execution priority of system symbionts relative to user jobs. For example, specifying SYMBPRI=3 results in user job steps at priority 1 and 2 (specified on the EXEC job control statement) executing at a higher priority than system symbionts. However, this technique should be used only in rare situations for user job steps that are extremely time critical.

MAXRUNSYMBS Keyword Parameter:

```
MAXRUNSYMBS={1-10}
            {2   }
```

Specifies the maximum number of run symbionts that can be executed concurrently in the system. You can select any value from 1 through 10, or let the system default to 2.

4.2.9. Main Storage Management Parameters

SUPVPROTECT Keyword Parameter:

```
SUPVPROTECT={NO }
            {YES}
```

Specifies whether the supervisor memory is read protected. If YES is specified, all supervisor memory areas and dynamic buffers are protected from read access except for those areas requiring direct access by a user. Attempts by a user job to access any protected areas results in failure due to protection exception. This facility, if selected, causes some memory waste (from 0 to 4095 bytes) in the resident supervisor area and likewise some waste in the dynamic buffers.

VOLTABLE Keyword Parameter:

VOLTABLE={NO / YES}

Specifies whether the system's volume table is resident in main storage. When YES is selected (indicating residence), this feature allows faster scheduling of jobs, thereby improving performance. The volume table requires 512 bytes of main storage. If your system can accommodate this feature, it is recommended that you specify YES.

SYMBMEM Keyword Parameter:

SYMBMEM={5-100 / NLMT}

Allows you to specify the percentage of available memory allocated for symbiont use. NLMT indicates no limit to the amount of memory available for symbiont use.

INTMEM Keyword Parameter:

INTMEM={5-100 / NLMT}

Specifies the percentage of available memory allocated for interactive services use. NLMT indicates no limit to the amount of memory available for interactive services use.

JOBMEM Keyword Parameter:

JOBMEM={0-100 / NLMT}

Specifies the percentage of available memory allocated for job use. NLMT indicates no limit to the amount of memory available for job use.

ROLLOUT Keyword Parameter:

```
ROLLOUT={NO  }
        {YES }
```

where:

YES

Specifies the capability of the supervisor to recognize jobs with preemptive priority. The supervisor can then roll lower priority jobs out of main storage for higher priority jobs and later roll those lower priority jobs back into main storage. If the ROLLOUT keyword parameter is not specified, the supervisor automatically equates the preemptive priority jobs to high priority jobs; thus, no rollin and rollout are performed.

EXPREGION Keyword Parameter:

```
EXPREGION={4096      }
          {0-999999  }
```

Specifies the size of your system's dynamic buffer pools within main storage from which your system can dynamically allocate buffers to your job.

If you wish to avoid dynamic allocation of buffer pools altogether, specify EXPREGION=0. This option forces dynamic buffer management to allocate all its needed buffers from the resident buffer pool, eliminating the possibility of fragmentation from the presence of buffer pools scattered throughout main storage. If you choose this parameter, be sure to specify a large enough resident buffer pool for your system's needs with the RESBUFSIZE parameter.

You can change the size of, or eliminate, your system's expansion region at system IPL time. For more information see the operations handbook operator reference, UP-8859 (current version).

RESBUFSIZE Keyword Parameter:

```
RESBUFSIZE={1-999999}
           {500     }
```

Specifies the size of the resident buffer pool within the supervisor. This pool serves one of two related functions, depending on what you specify with the EXPREGION keyword parameter:

If you specify a nonzero value (with EXPREGION), the resident buffer pool will be used only when dynamic buffer management cannot obtain a buffer in an expansion region. In this case the default size of 500 bytes is probably sufficient for all systems using workstations.

If you specify EXPREGION=0, making expansion regions unavailable to the supervisor, the resident buffer pool becomes the only buffer pool available for dynamic allocation of system buffers such as those used for workstations. If you are a moderate to heavy user of workstations, you should specify a large value, for example 200000 bytes, then raise or lower that size depending on your experience.

You can change the size of your system's resident buffer pool at system IPL time. For more information, see the operations handbook operator reference, UP-8859 (current version).

IPCBUFSZ Keyword Parameter:

IPCBUFSZ={256-4096 / 1024}

Specifies the maximum sized buffer used for transmission to or from your system host. The default is 1024 bytes (standard value).

DLOADBUFR Keyword Parameter:

DLOADBUFR={ / 1-32767}

Specifies the maximum number of 256-byte blocks of main storage that can be dynamically allocated (expanded) for a program compiled under ANSI 1974 COBOL. If you want COBOL CALL statements to dynamically expand your job regions, specify the maximum value of 32,767 unless you limit the expansion by specifying a smaller value.

DLOADTABLE Keyword Parameter:

DLOADTABLE={ / 1-255}

Specifies the number of entries per job in the job DLOAD table. This option applies only to programs compiled under ANSI 1974 COBOL. Choose a value equal to the maximum number of COBOL CALL statements in any one COBOL job. For example, if JOB A contains 4 CALL statements and JOB B contains 10, specify DLOADTABLE=10.

4.2.10. Shared Code Management Parameters

SCDINDEX Keyword Parameter:

```
SCDINDEX={NO }
         {YES}
```

where:

YES
: Allows faster loading of shared code modules into main storage and faster processing between two or more shared code modules. Commonly used shared code modules include the OS/3 general editor (EDT), interactive services, screen format services, data management, and ESCORT. This feature uses approximately 4000 bytes of main storage.

SHAREDMGT Keyword Parameter:

```
SHAREDMGT={1-2000}
          {40    }
```

Specifies the total number of slots to be reserved in a resident table that controls shared code modules. Each shared code module in main storage at a given time requires one table entry, or slot. For most multijobbing applications, the default value of 40 slots is sufficient. If your system uses unusually heavy multijobbing (and, especially, if you often get system messages warning of a full shared code table), you should specify more than the default number of slots. If you have fewer than three job slots and the conditions above do not apply, you can decrease the value.

Each slot you specify is 40 bytes long.

RESHARE Keyword Parameter:

```
RESHARE=[{shared-load-module-1      }] [,...,{shared-load-module-n      }]
        [{shared-load-module-group-1}]      {shared-load-module-group-n}]
```

Specifies a list of shared load modules or groups of modules, each name being one to eight characters in length, that are to reside in main storage at IPL. All shared load modules reside in system library YSCLOD. For a list of module group names, refer to the system installation user guide/programmer reference, UP-8839 (current version). To get a printed list of the individual modules in YSCLOD, key in the following console command:

```
RV SCLIST
```

The OS/3 supervisor can handle shared code modules efficiently without the use of the RESHARE parameter. For example, modules that are frequently used are not deleted from main storage between calls unless there is a critical need for that main storage area; this reduces the time spent loading shared code. As a module's frequency of use drops, however, it becomes more likely to be deleted from main storage between calls; this helps to reduce the main storage fragmentation that can occur when shared code regions are left allocated in small pieces throughout main storage.

In some environments, it might be possible to further improve the efficiency of shared code handling by using the RESHARE parameter. Before using it, however, you should have a good understanding of which shared code modules are most heavily used in your environment. Sometimes, it's obvious: if, for example, you are a heavy user of the dialog processor, it's probably desirable to specify the shared-load-module-group name "DP" with the RESHARE parameter. Refer to the system installation user guide/programmer reference, UP-8839 (current version), for a list of eligible group names. But there may also be some individual shared code modules that would be good candidates for inclusion in RESHARE. You can identify these by entering the MI SC console command from time to time and comparing the lists for shared code modules present each time. Or you can analyze system dumps taken repeatedly over a period of time. Remember, though, that these resident shared code modules take up space in main storage even when they are not in use so you should be very selective about which modules or module groups you specify with RESHARE.

IGNORESFT Keyword Parameter:

```
IGNORESFT={NO }
          {YES}
```

where:

NO

Specifies that the system processes all // SFT statements, loading the data management modules these statements identify before the job is scheduled.

YES

Specifies that the system ignores all // SFT statements, loading the data management modules these statements identify only when the job needs them during execution. This option allows you to use dynamic shared code without having to remove // SFT statements from all of your existing control streams.

4.2.11. Spooling Parameters

SPOOLING Keyword Parameter:

```
SPOOLING=(INPUT  )
         {NO     }
         {OUTPUT }
         {REMOTE }
         {DDP    }
```

Establishes spooling and specifies the spooling level. If no spooling capability is specified, all other spooling parameters are ignored. Also, no virtual devices are allocated by input/output generation (I/OGEN).

where:

INPUT
Specifies that both input readers and output writers are used for spooling.

NO
Specifies no spooling capability.

OUTPUT
Specifies that only output writers are used for spooling.

REMOTE
Specifies that remote batch processors, input readers, and output writers are used for spooling. If you specify SPOOLING=REMOTE, you must also specify whether the spool file output is compressed by using the SPOOLICAM parameter.

DDP
Specifies that distributed data processing (DDP), remote batch processors, input readers, and output writers are used for spooling. If you specify SPOOLING=DDP, you must also specify whether the spool file output is compressed by using by using the SPOOLICAM parameter.

SPOOLVSN Keyword Parameter:

```
SPOOLVSN={SYSRES}
         {vsn   }
```

Allows you to specify the volume serial number of the primary disk that the supervisor uses as a spool file.

SPOOLVSNn Keyword Parameter:

SPOOLVSNn=(SYSRES / * / vsn)

Supports multivolume spooling. You may allocate up to eight disk volumes for spooling. Identify the nth sequential volume of the spool file, where n is a decimal number from 2 to 8. You may identify each spool volume through multiple calls of a keyword (e.g., SPOOLVSN2=vsn, SPOOLVSN3=vsn, etc), or you may identify only the last sequential volume. If you identify only the last volume, all other volumes default to *, permitting you to specify the device address of a disk unit containing spooling volumes at initial program loading (IPL).

SPOOLCYL Keyword Parameter:

SPOOLCYL=(ALL / 1–1000 / 50)

Specifies the number of cylinders initially allocated for the spool file on the primary spooling volume. It is recommended that you use the default 50 unless your request exceeds the value of virtual devices allocated by I/OGEN parameters. (Refer to system installation user guide/programmer reference, UP-8839, for more details.) In this case, you may require more than 50 spooled cylinders. If you select ALL, this allocates all available cylinders on the primary spooling volume. This option should be used only for a dedicated spooling volume.

SPOOLCYLn Keyword Parameter:

SPOOLCYLn=(ALL / 1–1000 / 50)

Supports multivolume spooling. It enables you to specify the number of cylinders initially allocated for a spool file on the nth sequential spooling volume, where n is a decimal number from 2 to 8. It is recommended that you use the default 50 unless your request exceeds the default value of virtual devices allocated by I/OGEN parameters. Refer to system installation user guide/programmer reference, UP-8839 (current version), for more details. If this is the case, you may require more than the default number of spooled cylinders. Selecting ALL allocates all available cylinders on the primary spooling volume. This option should be used only for a dedicated spooling volume.

SPOOLMAP Keyword Parameter:

SPOOLMAP={1-32767 / **64**}

Specifies the number of full words (four bytes) of main storage reserved for the resident spool file bit map. Spool file suballocation is controlled by the bit map. If multivolume spooling is being used, the spool file bit map size should be calculated on the total number of cylinders being reserved for the spool file. Refer to system installation user guide/programmer reference, UP-8839 (current version), for the spool file bit map calculation table.

SPOOLBUFR Keyword Parameter:

SPOOLBUFR={1 / **2** / 4 / 8 / 16 / 32}

Specifies the size of the spool buffer allocated to each job preamble in terms of the number of 256-byte, main storage blocks. storage. Specifying a larger value for this keyword parameter increases system performance by reducing input/output to the spool buffer. You can specify only the parameters shown in the format.

SPOOLOWBUFR Keyword Parameter:

SPOOLOWBUFR={**2** / 4 / 8 / 16 / 32}

Lets you specify the number of 256-byte blocks of main storage allocated to spool buffers for print and punch output writers. Specifying larger values for this keyword parameter increases system performance by reducing the number of disk accesses to the spool file. You can specify only the parameters shown in the format.

SPOOLBURST Keyword Parameter:

SPOOLBURST={**NO** / YES}

Specifies whether output spooling functions in burst mode enable output files to be written before termination of an associated job. Operating in burst mode requires output processing criterion configured to control output writer's mode of processing for available output files.

where:

YES
Specifies output spooling functions in burst mode of operation. If omitted, output file processing by output writers is not done until the job terminates (nonburst mode).

SPOOLHDR Keyword Parameter:

SPOOLHDR={NO / YES}
Specifies whether spool files are printed with headers.

where:

NO
Specifies that spooled output files are printed without headers.

YES
Is the default that specifies a 3-page header precedes the printing of each spool file.

SPOOLPRT Keyword Parameter:

SPOOLPRT={ACT / ALL / LOG / NO}
Specifies the log printing criteria for a terminated job. This includes log types for printing (L and A records) and whether to print log records. You must specify JOBACCT=YES to have A records.

where:

ACT
Specifies that job accounting records are printed when job terminates.

ALL
Specifies that both job log and accounting records are printed when job terminates.

LOG
Specifies that only log records are printed when job terminates.

NO
Specifies that both job log and accounting records are not printed when job terminates.

SPOOLCOMP Keyword Parameter:

SPOOLCOMP={NO / YES}

Specifies whether spool file output is compressed where three or more consecutive blanks are present.

where:

NO

Specifies that the output image in the spool file should not be compressed. It is recommended that you do not specify SPOOLCOMP=NO when most files contain large amounts of blanks or if you use block sizes greater than 120.

If omitted, the spool file output image is compressed.

SPOOLUPDATE Keyword Parameter:

SPOOLUPDATE={NO / YES}

Specifies whether the spool directory entry is updated when a logical track is crossed or when the file is closed.

where:

NO

Indicates that the entry is updated only when the file is closed. Should the program cancel, you lose any output that the program generated before cancellation.

If omitted, the spool directory is updated each time a logical track is crossed. Should a program cancel, you are able to print any output that the program generated before cancellation.

DDPSPOOL Keyword Parameter:

DDPSPOOL={1-128 / 10}

Specifies the number of concurrently running tasks that returns spooled output for distributed data processing (DDP) jobs. Each task that returns DDP spooled output takes a task control block and at least 8K of dynamic buffer space. In most cases, the default value of 10 tasks is sufficient. If however, your system takes too much time to return DDP spooled output, it is recommended that you assign a higher value. If the default value of 10 tasks degrades your system's overall performance, lower the value.

SPOOLMAXLINE Keyword Parameter:

SPOOLMAXLINE={1-255 / 5}

Specifies the maximum number of printer input/output records, in thousands, to be processed before the system halts the job and sends a warning message to the operator.

SPOOLICAM Keyword Parameter:

SPOOLICAM={C1 / Cn / Mn / C?}

Specifies the name of the integrated communications access method (ICAM) symbiont load module (C1-C9, M1-M9, or C?) called by the spooler to service remote batch and distributed data processing (DDP) spooling. If C? is specified, the operator is asked at load time for a valid ICAM name. You must specify the parameter SPOOLING=REMOTE or SPOOLING=DDP and must configure the ICAM symbiont as described in the COMMCT section of system generation. Refer to system installation user guide/programmer reference, UP-8839 (current version).

If omitted, jobs creating remote batch or DDP output encounter an error at the end of the job and require ICAM (load module C1) to be manually loaded to process output.

SPOOLRECV Keyword Parameter:

SPOOLRECV={ALL / CLOSED / LOG / NONE}

Specifies the level of recovery for the spool file when the supervisor is initialized during initial program loading (IPL) procedure.

where:

ALL
Specifies that all complete and incomplete spoolfiles are recovered (warm start).

CLOSED
Specifies that only completed spool files are recovered (warm start).

LOG
Specifies that only completed log files are recovered (warm start).

NONE
Specifies that the spool file is not to be recovered (cold start).

SPOOLTEST Keyword Parameter:

SPOOLTEST={NO / YES}

Specifies whether your supervisor suppresses console messages requesting an operator decision to print a test line when a change of form is required.

where:

NO

Indicates that test line messages should not be displayed.

If omitted, test line messages are displayed.

SPOOLFARSI Keyword Parameter:

SPOOLFARSI={NO / YES}

Specifies whether FARSI (Iranian Language) translation is executed for all spooled files containing *FARSI or @FARSI as the first six characters of the // LBL job control statement.

where:

YES

Indicates that FARSI translation is to be used; in which case, the output writer must be loaded with an additional 400_{16} bytes of main storage, and the input reader must be loaded with an additional 300_{16} bytes of main storage.

If omitted, FARSI translation is not executed.

SPOOLNOINPUT Keyword Parameter:

SPOOLNOINPUT={NO / YES}

Specifies whether your supervisor suppresses input spooling for remote batch. Use this parameter when you specify SPOOLING = REMOTE or SPOOLING = DDP and you want to suppress the input spooling capability throughout your spooling system. When you suppress input capability, you cannot run the input reader.

If omitted, remote batch spooling is not suppressed.

SPOOLMODE Keyword Parameter:

```
SPOOLMODE=[ACCTNO,account-number][CARTNAME,cartridge-name]
[DEVICE,device-type-code][FILE,filename][FORM,form-name][JOB,jobname][PRI]
```

Establishes output file processing criterion for spooler when operating in burst mode. The output files are processed according to type-code specified, with processing ending when type-code is satisfied. If operating in the burst mode and no criterion is specified, SPOOLMODE equals PRI and output files are processed under job priority basis. If you are uncertain of the criterion to be implemented, specification changes can be made from a console later on. Refer to the procedure outlined in the handbook for operators, UP-8859 (current version).

where:

`ACCTNO,account-number`

Is a field containing one to four alphanumeric characters, indicating that account-number is the criterion used for determining file processing. All files created by jobs whose account number is the same as the account-number configured (you specify this on the job control statement) are processed without operator intervention.

`CARTNAME,cartridge-name`

Is a one to eight alphanumeric character field indicating that cartridge-name is the criterion used for determining file processing. All files whose cartridge name is the same as the cartridge-name configured (you specify this on the LCB job control statement) are processed without operator intervention.

`DEVICE,device-type-code`

You can specify device type codes as follows:

770	Model 8 only
776	Models 3, 4, 5, 6, and 8
789	Models 3, 4, 5, 6, and 8
798	Model 8 only

Indicates to the output writer that device-type number is the criterion for determining file processing. All files available for processing whose device type is the same as the criterion configured are processed without operator intervention.

FILE,filename
Is a one to eight alphanumeric character field indicating that filename is the criterion used for determining file processing. All files available for processing whose file name is the same as the filename configured are processed without operator intervention.

FORM,form-name
Is a one to eight alphanumeric character field indicating that form-name is the criterion used for determining file processing. All files whose form name is the same as the form-name configured (on VFB or SPL job control statements) are processed without operator intervention.

JOB,jobname
Is a one to eight alphanumeric character field indicating that jobname is the criterion used for determining file processing. All files whose job name is the same as the jobname configured (you specify this on the job control statement) are processed without operator intervention.

PRI
Specifies that job priority is the criterion for determining file processing. Files are processed on a first-in, first-out basis without operator intervention.

4.2.12. Input/Output Parameters

NOTE:

The parameters contained in this subsection define I/O software functions that complement the hardware your system uses. To configure the hardware itself, you must include I/OGEN parameters, one set for each supervisor configured by SUPGEN, as described in the system installation user guide, UP-8839 (current version).

TAPEBLKNO Keyword Parameter:

TAPEBLKNO={**NO** / YES}

where:

YES
Indicates that the supervisor supports tapes written with block numbers, as well as tapes without block numbers. If omitted, tapes written with block numbers are not supported.

COMM Keyword Parameter:

Models 3,4,5, and 6	Model 8
COMM={1-8 / YES / **NO**}	COMM={1-14 / YES / **NO**}

Specifies the extent to which your supervisor supports integrated communications access method (ICAM). This parameter must be specified if ICAM is used.

where:

1-8 (models 3,4,5, and 6) or 1-14 (model 8)
Specifies the number of single line communication adapters (SLCAs) in your system. This must be specified if your ICAM network interfaces with communication terminals.

YES
Specifies that your ICAM network interfaces with directly connected local workstations only. Do not specify YES if you have both local workstations and communication terminals.

NO
Specifies that communications are not supported by the system.

If the entire parameter is omitted, it is assumed that communication is not supported by your supervisor's configuration.

COMM1 Keyword Parameter:

COMM1={1-14 / YES / **NO**}

This parameter applies to model 8 only and specifies whether the system supports integrated communications access methods (ICAM) and two input/output microprocessor (IOMP) channels.

where:

1-14
Specifies the number of single line communication adapters (SLCAs) in your system. This must be specified if your ICAM network interfaces with communication terminals.

YES
Specifies that your ICAM network interfaces with directly connected local workstations only. Do not specify YES if you have both local workstations and communication terminals.

NO

Specifies that communications are not supported by the system.

If the entire parameter is omitted, it is assumed that communication is not supported by your supervisor's configuration.

CHAN Keyword Parameter:

CHAN={**13** / 15}

This parameter applies to model 8 only and specifies the input/output microprocessor (IOMP) channel number.

CHAN1 Keyword Parameter:

CHAN1={13 / **15**}

This parameter applies to model 8 only and specifies the second input/output microprocessor (IOMP) channel number.

IORB Keyword Parameter:

IORB={5-150 / **5 X COMM**}

Specifies the number of input/output resource blocks (IORBs) to be generated for the channel. The default value is equal to five times the number of communication lines specified in the COMM parameter.

IORB1 Keyword Parameter:

IORB1={5-150 / **5 X COMM1**}

This parameter applies to model 8 only and specifies the number of input/output resource blocks (IORBs) to be generated for the second channel. The default value is equal to five times the number of lines specified in the COMM1 parameter.

ISINTLMT Keyword Parameter:

ISINTLMT={1-255 / **0**}

Specifies the maximum number of interactive users that your system recognizes at any one time. Interactive sessions use only workstation or interactive communications terminal input.

DDPMAXUSERS Keyword Parameter:

DDPMAXUSERS={1-256 / **16**}

Specifies the maximum number of users allowed to access distributed data processing (DDP).

DDPMAXHOSTS Keyword Parameter:

DDPMAXHOSTS={1-256 / 16}

Specifies the maximum number of hosts in the DDP network. You can choose any valid number between 1 and 256, or let the system default to 16.

IPCMAXSESS Keyword Parameter:

IPCMAXSESS={1-256 / 16}

Specifies the maximum number of connections to each remote host for your system. You can specify any value from 1 to 256, or let the system default to 16 connections.

ISBATCHLMT Keyword Parameter:

ISBATCHLMT={1-255 / 8}

Specifies the maximum number of batch sessions that can execute concurrently in your system. Batch sessions use only card image, tape, or disk input.

ISINTPRI Keyword Parameter:

ISINTPRI={1-60 / 1}

Specifies the priority level of interactive commands entered from either a workstation or an interactive terminal.

ISNETNAME Keyword Parameter:

ISNETNAME=network-name

Specifies a 4-character communications network name that tells interactive services on which network to operate. If your system uses a communications network, you must specify one network name. If you do not specify a name, your system defaults to no name and your communications network will not support interactivity. This parameter does not affect the operation of local workstations, but it is required if you want your remote terminals to act as workstations.

ISLOCAPID Keyword Parameter:

ISLOCAPID={locap-name / OS3}

Specifies a 1- to 4-character name that you request when you want to use interactive services. This name must match the LOCAP name for interactive services in the system's COMMCT network definition. (See system installation user guide /programmer reference, UP-8839 (current version).) If the default OS3 is used, ICAM supplies the locap name. Once connected to interactive services, a terminal can accomplish the same tasks as a workstation.

If you omit this parameter, you will not be able to connect to interactive services from a communications terminal. In the COMMCT phase of system installation, you must specify demand mode interface if you want your communications network to support interactivity.

4.2.13. Floating-Point Parameter

FLOATPT Keyword Parameter:

```
FLOATPT={NO  }
        {YES }
```

where:

NO
Indicates that the supervisor does not support floating point.

You must include floating point capability in your supervisor if you intend to run FORTRAN programs.

Specifying FLOATPT=YES increases each TCB in your system by 32 bytes.

Appendix A. Statement Conventions

The conventions used to delineate the configuration parameter values in this manual are:

- Parameter definition is by keyword association. Keyword parameters consist of a word or a code immediately followed by an equal sign, which is followed by a specification. Keyword parameters can be written in any order except where restrictions are noted. Keyword parameters may be written in columns 1 through 71. More than one keyword parameter may be included on the same card, but they must be separated by at least one blank character. Keywords and their specifications must be contained on the same card.

  Examples:

```
PRIORITY=5
PRIORITY=5    JOBSLOTS=3    TRANS=3
PRIORITY=5    TRANS=3       JOBSLOTS=3
```

- Capital letters, commas, equal signs, and parentheses must be coded exactly as shown. The exceptions are acronyms that are part of the generic terms representing information to be supplied by the programmer.

  Examples:

```
SUPMOD=supvrnam
COMM=2,18
CACH=(nn,CCA-name,line-number)
```

- Lowercase letters and words are generic terms representing information that must be supplied by the user. Such lowercase terms may contain hyphens and acronyms (for readability).

  Examples:

```
channel
supervisor-name
vsn
```

- Information contained within braces represents alternate choices of which only one may be coded.

  Examples:

  ```
  {MAX }
  {MIN }
  {NO  }
  {NORM}
  ```

- Information contained within brackets represents optional entries that (depending upon program requirements) are included or omitted. Braces within brackets signify that one of the specified entries must be chosen if that parameter is to be included.

  Examples:

  ```
  [SUPMOD=supvrnam]
  [SUPVRNAM={supervisor-name}]
            {SY$STD         }
  ```

- An optional parameter that has a list of alternate entries may have a default specification that is supplied by the operating system when the parameter is not specified by the user. Although the default may be specified by the user with no adverse effect, it is considered inefficient to do so. For easy reference, when a default specification occurs in the format delineation, it is printed on a shaded background. If, by parameter omission, the operating system performs some complex processing other than parameter insertion, it is explained in an "if omitted" sentence in the parameter description.

  Examples:

  ```
  [SUPVRNAM={supervisor-name}]
            {SY$STD         }
  ```

- Keyword parameters may contain sublists called subparameters. Subparameters may be positional or nonpositional, as indicated in the text. Subparameters must not be separated by blanks.

  – Positional subparameters must be coded in the order shown, and commas must be retained for any that are omitted, with the exception of trailing commas.

    Examples:

    ```
    SYSOUT=DSP00,8417,PARTIAL
    SYSOUT=DSP00,8417
    SYSOUT=vsn,,NOPREP
    SYSOUT,,PARTIAL
    ```

- Nonpositional subparameters may be coded in any order on a single card, separated by commas, or they may be coded on separate cards, repeating the keyword parameter. If all subparameters will not fit on one card, the keyword parameter must be repeated on the second card.

  Examples:

```
RESMOD=SM$ASCKE,SM$ATCH,SM$DBS,SM$GTPUT,SM$LOCK
RESMOD=SM$ASCKE,SM$ATCH,SM$DBS,SM$GTPUT
RESMOD=SM$LOCK
```

- The label entry must begin in column 1.

  Example:

```
Label     △Operation△     Operand
1         1Ø    16                                          72
SUPGEN
END
```

- The assembler coding form (UD1–1548) is recommended for coding of the SYSGEN keyword parameters, since the LABEL, OPERATION, and OPERAND field limits are illustrated.

- Keyword parameters must not appear on the SUPGEN label parameter cards.

CUT

SPERRY

USER COMMENT SHEET

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

_____

(Document Title)

_____ _____ _____

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

From:

_____

(Name of User)

_____

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

CUT

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY CORPORATION

ATTN.: SOFTWARE SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

FOLD