IBM

Application Program

GH20-0786-3

CALL – OS

Executive and Utilities

Program Description Manual

Program Number 360A-CX-42X

This publication describes the facilities provided by CALL-OS to installation personnel who are responsible for the selection, evaluation, and implementation of the system. The intended audience includes systems engineers, installation programmers, marketing representatives, and customer systems personnel.

CALL-OS is a terminal-oriented, time sharing system designed to function under the control of the IBM System/360 Operating System with either of two options: Multiprogramming with a Fixed Number of Tasks (MFT), or Multiprogramming with a Variable Number of Tasks (MVT). From the terminal user standpoint, the CALL-OS service environment approximates that of a dedicated, in-house, data processing installation.

Note: The CALL/360-OS system has been renamed the CALL-OS system. Thus, documentation of Version 2 of the CALL/360-OS system refers to the system as CALL-OS.

Terminal Equivalency

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

Fourth Edition (March 1972)

This edition, GH20-0786-3, is a major revision obsoleting GH20-0786-2. Significant changes have been made throughout this edition and it should be reviewed in its entirety.

This edition reflects Version 2, Modification Level 0, of the CALL-OS time sharing system and all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are continually being made to the specifications contained herein. Therefore, before using this publication, consult the latest System/360 SRL Newsletter (GN20-0360) for the editions that are applicable and current.

Copies of this and other IBM publications can be obtained through IBM branch offices.

A form has been provided at the back of this publication for reader comments. If this form has been removed, address comments to IBM, Technical Publications Department, 1133 Westchester Avenue, White Plains, New York, 10604.

PREFACE

This publication describes the facilities provided by CALL-OS and discusses the concepts and techniques underlying their use. It is intended as a reference guide for systems engineers, installation programmers, marketing representatives, and customer systems personnel in the implementation of the CALL-OS time sharing system.

This publication contains nine major sections. The first section summarizes concepts needed for a better understanding of the system and describes the organization of the system; the next two sections describe the CALL-OS data base and the CALL-OS Batch Interface facility (COBI), respectively. The first two sections should be read by anyone not familiar with the CALL-OS system; the section on COBI should be read by anyone who desires to use this facility.

The next five sections contain the information necessary to design, build, and initialize a CALL-OS system, create and maintain the data base, and maintain the system itself. These sections are procedure-oriented and intended for use by experienced personnel.

The last section summarizes diagnostic aids available to the system programmer. An additional source of diagnostic information is the publication CALL-OS Operator's Manual, which contains all system messages and ABEND codes, as well as explanations. Finally, appendices provide additional information.

To derive maximum benefit from this manual, the user should have a working knowledge of the following support publications:

CALL-OS System Description Manual (GH20-0673)

CALL-OS Terminal Operations Manual (GH20-0787)

CALL-OS Operator's Manual (GH20-0788)

IBM System/360 Operating System: Concepts and Facilities (GC28-6535)

IBM System/360 Operating System: Job Control Language (GC28-6539)

IBM System/360 Operating System: System Generation (GC28-6554)

IBM System/360 Operating System: Linkage Editor and Loader (GC28-6538)

CONTENTS

FIGURES

TABLES

INTRODUCTION TO THE CALL-OS SYSTEM

CALL-OS is a terminal-oriented, time-sharing system which provides an individualized computing capability to a variety of users. It is designed to handle a high volume of traffic in a problem-solving environment, and to satisfy the needs of the experienced professional as well as the uninitiated computer user. From the individual terminal user viewpoint, the service environment approximates that of a dedicated data processing installation.

The system is interrupt-driven, which eliminates recurring interrogation of terminals by the computer center and reduces operating overhead. Most terminal-originated interrogations result in near-immediate responses. The facilities provided by CALL-OS constitute a complete in-house computing and data service capability. Some of these facilities are:

- Concurrent batch-processing capability under OS
- Highly responsive personal computing system under OS, designed for problem solving
- Extensive terminal command language, directed towards both the experienced and the inexperienced user
- Multiprogramming within a single task area
- Dynamic assignment of dispatching priorities to provide efficient use of CPU time
- Shared libraries which may contain source and/or object programs, and/or data files
- The ability to submit jobs to OS batch processing from the terminal with the CALL-OS Batch Interface (COBI) facility; COBI also allows the user to retrieve the output from the job at his terminal
- Three programming languages processed by three fast load-and-go compilers which generate dynamically relocatable code
- Terminal checkout of user programs
- Extensive edit capabilities for modification of line-numbered files
- Ability to link one user-written program to another via chaining facilities
- Ability to enter line-numbered information as a program-data file, either directly from a terminal or by means of a CALL-OS utility, and to read such a file as input during program execution
- Facility for entry of source programs, data, and terminal commands via paper tape
- Operator control of some system resources
- Security features which protect each user's data
- Offline utilities which provide facilities for building and maintaining the system and its data base

CALL-OS operates as a task under the control of the IBM Operating System (OS), Multiprogramming with a Fixed Number of Tasks (MFT) or Multiprogramming with a Variable Number of Tasks (MVT) control program. It occupies a single region or partition, referred to in this manual as a task area. Because it requires only one task area, background jobs may execute concurrently to capitalize on the remaining CPU and core storage capacity, thus providing the multiprogramming functions of the standard operating system, plus a compatible time-sharing capability. In this environment, each CALL-OS terminal user operates independently of every other user, and is usually unaware of other terminal users or other OS activity.

The CALL-OS system provides a personalized computing service to multiple terminal users. The user of this system is provided with an extensive terminal command language designed to facilitate communication between the terminal user and the computer. Several programming languages are provided to facilitate problem solving. CALL-OS operates under control of its own control program (executive), which performs all control functions and I/O operations, and provides a standard software interface for all compilers and user programs. The portion of the task area not used by the executive is called the user program area; this area is allocated to compilers and executing user programs.

The rest of this section defines system concepts required for better understanding of the system and the organization of the system.

SYSTEM CONCEPTS

CALL-OS employs numerous techniques and concepts woven together into a single functional package. Since a working familiarity with these elements is essential to a comprehensive understanding of the system, they are discussed in the following text.

PERSONAL COMPUTING

Personal computing can be defined as the ability of an individual to use the power of the computer to assist him directly in the solution of his daily problems. Key to the development of a personal computing tool is the development of the terminal-oriented, time sharing capability offered in CALL-OS. This capability permits individuals to obtain computing services when that capability is required. Such a computing capability is in essence transparent to the individual, thereby permitting him to concentrate on problem solutions without having to become further involved with computer/programming disciplines.

USER IDENTIFICATION AND SYSTEM SECURITY

When a user wants to use CALL-OS, he follows a specified procedure to sign onto the system. Part of this procedure involves identifying himself to the system. Identification consists of typing in a user number and a password. Only the user who gives the appropriate user number and password may access information previously retained under that identification.

A user number consists of six characters: the first three are alphabetic (A through Z) and the last three are numeric (0 through 9). User numbers are assigned and controlled by the installation.

A password consists of any combination of letters, numbers, and/or special characters, including embedded blanks, up to eight characters. The password is selected by the user, and, with his user number, provides his unique identification.

See the publication CALL-OS Terminal Operations Manual for a detailed description of the sign-on procedure and password assignment.

TIME SHARING AND TIME SLICING

In CALL-OS, time sharing is defined as the allocation of computer resources at a single facility, in a time-dependent manner, to several programs which are simultaneously core-resident. In this way, multiple users may occupy the same program task area at the same time, but actual program execution is effected on a scheduled basis for a single user.

Multiple residency is thus seen to be simultaneous, and rapid multiplexing from user to user provides the illusion of each user having the full resources of the system at his disposal.

In a time sharing environment, the technique employed in the allocation of system resources to operating programs is time slicing. A portion of the available central processing unit time is allocated to any compilation or execution task. This portion is called a time slice. By allocating a portion of the system resources to each user, a large number of users can be supplied with data processing services simultaneously.

Each terminal job is allotted a time slice as it enters the system. It is this time slice which determines the maximum length of time the job may process before it loses control to the next job in the job queue. The time slice allotted depends on the task to be performed. A basic design objective of the system is to ensure completion of most compilations and executions within the time slice.

JOB SWAPPING

Compilation is interrupted if a time slice is exceeded; execution is interrupted if the time slice is exceeded or if terminal input is requested. These interruptions cause the user program to be moved to a special area of disk storage (called the swap area) so that system resources can be allocated to other users. This process is called swapping. For example, upon expiration of a time slice, the currently active job is swapped out of the user program area onto a preassigned area called the disk work/swap area. A job is swapped back into the user program area when it again becomes eligible for execution.

TRIVIAL AND NONTRIVIAL RESPONSES

From the standpoint of the magnitude of system resources required, the functions and facilities of CALL-OS can be separated into trivial and nontrivial tasks. The amount of time taken by the system to respond to or accomplish one of these tasks is the response time.

For example, the acceptance of program statements keyed in by a user and the execution of certain system commands require minimal system resources; they are therefore carried out with essentially zero response times. This is possible because the required modules are permanently core-resident and execute to completion for each individual user.

If, on the other hand, a user requests execution of a program, storage must be allocated for this task, and significant amounts of system resources are required. To preclude one user from dominating the system, a time slice is allocated to the single user for such tasks. By allocating an increment of CPU time to each user who requires program execution, a large number of users can be accommodated with data processing services on a simultaneous basis. The result, however, is that these nontrivial tasks involve response times of the order of seconds, while near-instantaneous response is possible for trivial tasks.

DATA BASE

Information necessary for and created during operation of CALL-OS is kept in a collection of data sets known as the data base. This data base is used by CALL-OS for the storage and retrieval of system and user resources. Examples of these resources are programs, data files,

compilers, and overlay modules. The data base consists of three logical parts:

- The index, which identifies all permissible DD statements and data sets to be used by CALL-OS; the index is used primarily for data set identification during system initialization and offline data base manipulation.

- The system base, which fills the needs of the system for compiler storage, work/swap area, and nonresident modules.

- The user base, which contains all user number oriented data, such as passwords, programs, data files, and COBI job identification.

A detailed description of the data base is given in the section "CALL-OS Data Base".

LIBRARIES

CALL-OS provides two types of libraries: user libraries and system libraries. A user library contains information associated with a single user; a system library contains information shared by many users. Each type of library is described in more detail in the following text.

User Libraries

One user library is available to each terminal user. This library contains all programs and data files retained by the user, as well as any OS/360 jobs he intends for submission through COBI. The information in a user library is controlled only by the user associated with the library; however, the information may be made available to other users of the system.

System Libraries

The system libraries allow information to be shared among many users. The following system libraries are available:

- The *Library, accessible to only a specific group of users; that is, those users whose user numbers are identical for the first four characters

- The **Library and the ***Library, accessible to all users of the system

The information in the *Library is controlled by the terminal user. He specifies the name of a program or data file he wishes to share with other users. This name is entered in a special list, called a directory, which also indicates the sharing user; the program or data file itself remains in the appropriate user library. Only the user who shared (or pooled) the information may alter it or cause its name to be deleted from the *Library.

The information in the **Library is also controlled by the terminal user. This library operates in the same way as the *Library: that is, only the names are specified as being shared; the programs and data files remain in the sharing user's library. The difference between the two libraries is that programs and data files named in the **Library are available to all users of the system, while programs and data files named in the *Library are available only to the other users associated with that particular library.

The information in the ***Library is controlled only by the installation. This library contains a list of the programs and data files in it, as well as the programs and data files themselves. In this case, the list is called a catalog.

RESIDENT AND NONRESIDENT MODULES

The executive consists of two types of modules: permanently resident and potentially nonresident. During system build, all the permanently resident modules are link edited into a single load module called the base or nucleus, and each potentially nonresident module is link edited into the installation's job library as a separate module.

During system initialization, the installation specifies a list of potentially nonresident modules which are to be made resident. Modules named in the list are loaded with the base and become part of the resident system for this session of CALL-OS. Modules not named in the list are nonresident for this session; these modules are written into the overlay data set and are brought into storage when needed during system operation.

USE OF STORAGE WITHIN THE CALL-OS TASK AREA

In the storage available to OS/360, the nucleus occupies the low end addresses; the rest of core storage is divided into task areas as needed and assigned to jobs entered with the job control language. When CALL-OS is used, it typically occupies a task area at the high end of core storage. The reason for this depends on the system used: in an MFT system, the high-end task area receives the highest priority and it is desirable for CALL-OS to have a high priority within OS/360; in an MVT system, since CALL-OS is a long-running job, it is desirable to use the high-end task area to prevent unnecessary core fragmentation.

The storage between the CALL-OS task area and the OS/360 nucleus is divided into background task areas, which are given control by CALL-OS for a certain percentage of execution time or when CALL-OS has no work to be done. The task area for CALL-OS is divided into the executive area and the user program area, as shown in Figure 1.

Executive Area

The CALL-OS executive area contains the modules, subroutines, buffers, and control blocks necessary for execution. The base resides at the low end of the task area; during initialization, other modules may be made resident for the current execution of CALL-OS. Every user who signs on the system is given a control block called a user terminal table (UTT); this table contains information pertinent to the user. In addition, CALL-OS requires certain buffers, for example, the overlay buffer from which nonresident modules execute.

User Program Area

The user program area is used for the compilation and execution of user programs. This area is made up of three parts:

- Old job area
- New job area
- Compiler area

By definition, a job is considered a new job until its time slice has been exceeded. An old job is a job which has been compiled and has exceeded one time slice while in execution, and has not requested terminal I/O during its current execution. A compilation which exceeds a time slice is swapped out, but is placed at the end of the new job queue.

The compiler portion of the user program area is determined during system initialization, and is of sufficient size to contain the maximum compiler (or compiler phase) in the system. It is always occupied, either by the compiler in use or by the compiler that was last used.

The new job area is determined during system initialization based on the amount of core available. There is always one new job area, and there may be two new job areas. The boundary between the new job area and the old job area is adjustable; therefore, large old jobs can be executed by taking space from the new job area as required. Referring to Figure 1, new job area 1 is allocated upward from the compiler area, while new job area 2 is allocated downward from the boundary between the old and new job areas.

SYSTEM ORGANIZATION

The CALL-OS system can be considered to contain four parts: the executive, command languages, processing programs, and the CALL-OS Batch Interface (COBI) facility.

- The executive governs the order in which the processing programs are executed, and provides services that are required in common by the processing programs during their execution.
- Command languages serve as a means by which the terminal user and the system operator can communicate with the system.
- Processing programs consist of compilers and utility programs which are provided by IBM to assist the user, as well as user-written problem programs which are executed under CALL-OS. Both IBM and user-written programs have the same functional relationship to the executive.
- COBI is optional, and it permits the terminal user to submit jobs to OS/360 batch processing from his terminal and, if he desires, have the output printed at his terminal.

Figure 2 shows a simplified picture of some of the system components.

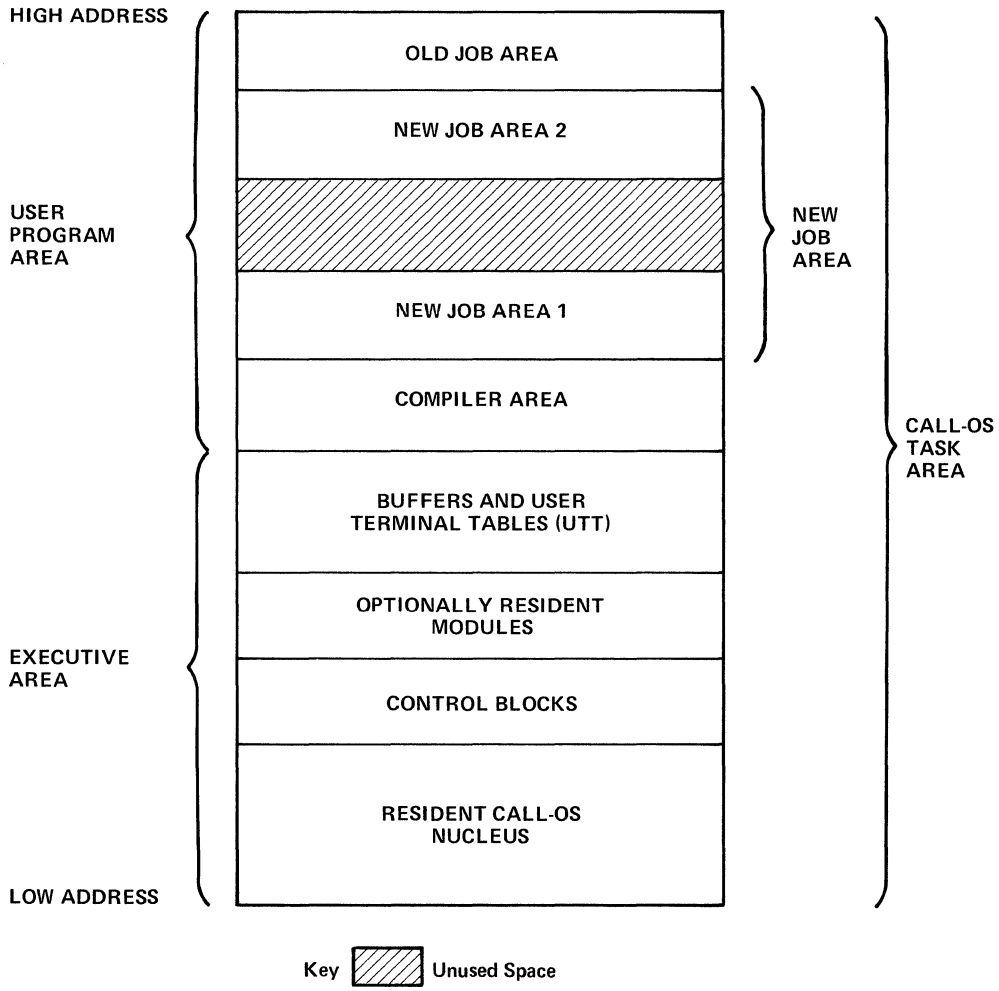


Figure 1. Use of storage in the CALL-OS task area

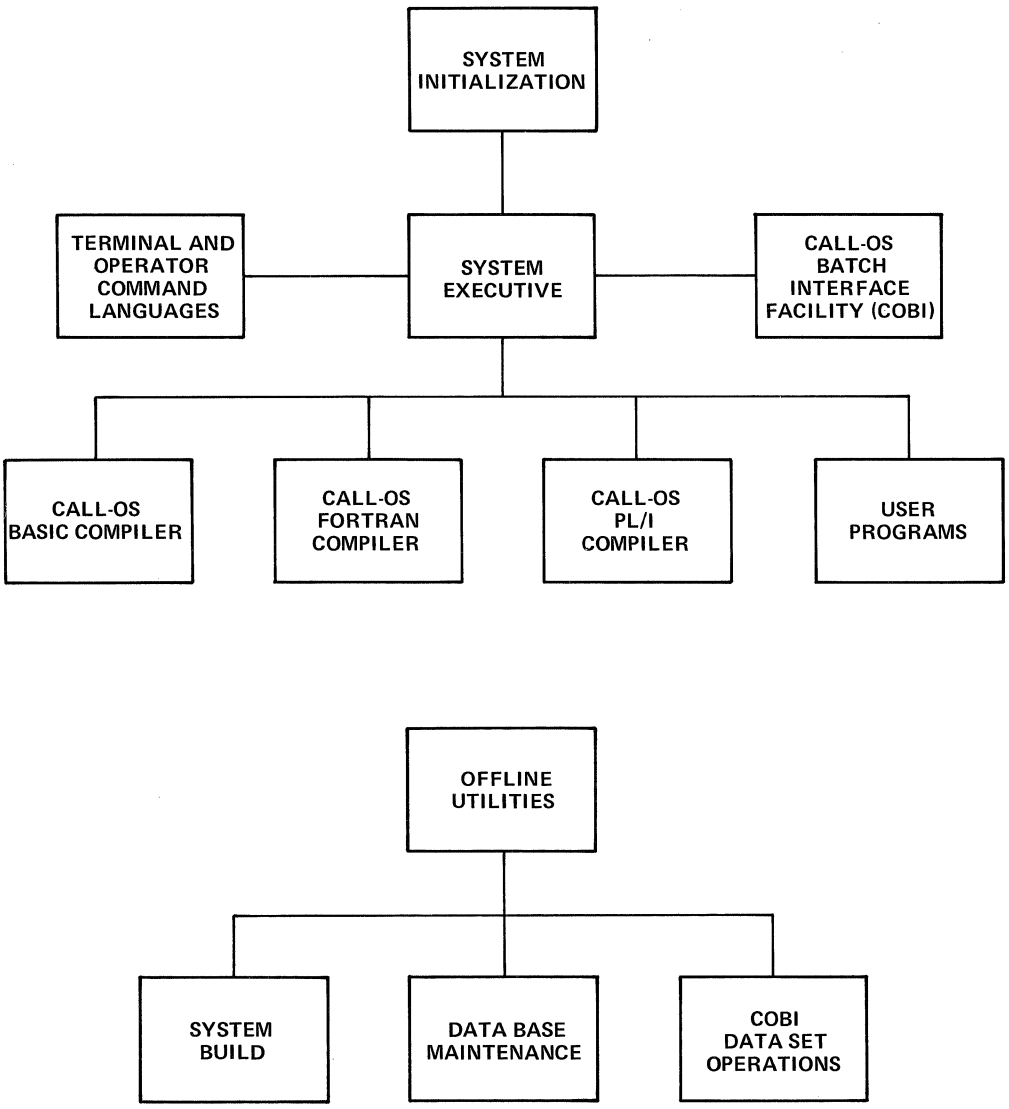


Figure 2. CALL-OS system components

EXECUTIVE

The executive serves as the control program for CALL-OS. It forms the heart of CALL-OS and controls all facets of process monitoring. It resides at all times in permanent areas of core storage which are storage-protected to ensure that they are not inadvertently modified. It is basically made up of the following control routines:

- Control program interrupt dispatcher (CPID)
- Resource managers
- User program area manager
- Terminal and disk I/O handlers

Their functions are summarized below.

Control Program Interrupt Dispatcher

CPID is interrupt-driven and operates disabled in the supervisor state. Except for machine checks, CPID intercepts all interrupts occurring during OS/360 and CALL-OS operation. Interrupts not related to CALL-OS operation are transferred directly to OS/360 for processing; interrupts related to CALL-OS operation are processed by CPID, and in some cases, transferred to OS/360 for further processing.

In addition to handling interrupts, CPID also dispatches all work to be done by CALL-OS, and supervises the amount of time allocated to the various tasks performed by CALL-OS, as well as to background processing of OS/360 jobs. During its processing, CPID interfaces with:

- OS/360 interrupt routines
- OS/360 task management thereby communicating with OS/360 when it is ready for operation
- OS/360 job management, if COBI is used, to process jobs submitted at the terminal and receive the output
- OS/360 timer supervisors because CALL-OS has both a real-time clock and a time-slice clock
- OS/360 input/output supervisor (IOS) routines by means of I/O appendages
- OS/360 trace routines, if desired, to insert specially formatted entries into the OS/360 trace table

Finally, CPID interfaces with the user program area manager to provide necessary services such as I/O operations for compilers and user programs. This interface is accomplished through a Type I supervisor call (SVC). This SVC is designed solely for use by CALL-OS and is included in an OS/360 nucleus during the process of building a CALL-OS system.

Dispatching of Work: CPID controls the dispatching of all work according to a five-level priority structure. Tasks requiring the fastest response time are given the highest priority. The five levels, in order by priority, are:

- Level 0, which consists of the CPID routines themselves, the I/O appendages, and those subroutines which operate with interrupts disabled

- Level 1, which consists of the modules which analyze commands, load the nonresident modules for execution, and execute from the overlay buffer, as well as those subroutines which operate with interrupts enabled

- Level 2, which consists of the modules required to process most commands

- Level 3, which consists of modules required to sort and/or perform editing functions on source programs

- Level 4, which consists of compilers and user programs; only one level 4 job is known to CPID at a time, as determined by the user program area manager

Each level has a queue of work to be done. These queues are maintained on a first-in, first-out basis and are used to determine the next task to be performed within each level.

Note: References to level 5 pertain to OS/360 background processing in conjunction with CALL-OS; level 5 is not, however, a level within the CALL-OS system.

Time Supervision: CPID supervises the allocation of time for the tasks required to service user requests. Three classes of time are kept. The first is a 6.7-second clock, which serves the needs of the executive system for user terminal control, time stamping, operator communication, etc.

The second clock is used for time-slicing control over user programs and compilers executing in the user program area. The amount of time allotted depends on the type of job. For example, new jobs, old jobs, and compilers have different time-slice values, as specified during system initialization. In addition, an old job which exceeds a time slice is allotted a larger time slice in order to increase the probability that the next time the job is dispatched it will complete execution.

The third clock is used to share the time available for executing user programs with lower priority jobs executing in OS/360 background. The time shared is allotted in increments; the size of the increment is specified during system initialization. The user program area manager determines who is to be given the next increment: more information can be found in a subsequent subsection.

Resource Managers

Resource managers control the allocation of the buffers used by CALL-OS. These buffers include:

- System buffer, used to hold system data such as data base records needed for processing user requests; there is only one system buffer and it is always present.

- Sort buffer, used to sort user source programs; there is only one sort buffer and it may be omitted with a system initialization option; if omitted, all sorting takes place in the user program area.

- Overlay buffer, used to hold nonresident modules for execution; there is only one overlay buffer and it is not present if the system is totally resident.

- 24-byte buffers, used to receive input from the terminal; these buffers, or pots, contain four bytes of control information and 20 bytes of data; when three pots are full, their data content is moved to a 60-byte buffer for output to disk; the number of pots for each line is specified during system initialization; the default is four pots per line with a minimum total of 60 pots in the system.

- 60-byte buffers, used to output source code to the work area on disk; there are ten 60-byte buffers in the system.

- 256-byte buffers, used for terminal output, such as listing of programs, execution output, and output from the scan function of COBI; the number of 256-byte buffers is specified during system initialization; the default is one buffer for every three lines with a minimum of five buffers.

A request for any type of buffer is entered in a queue for the buffer on a first-come, first-served basis. Activity on behalf of the user who needs the buffer is suspended until the request is filled.

User Program Area Manager

The user program area manager keeps track of the status of all jobs in the user program area, controls the swapping of jobs in and out of the area, and interfaces with CPID through the Type I SVC to provide executive services for executing compilers and user programs. The most important function of the user program area manager, however, is to inform CPID of the next job to receive control. Since the next job may be either in the user program area or in the background, the user program area manager is also responsible for sharing time with the background as well as determining the amount of time to be shared; this process is called background time slicing.

Scheduling Work in the User Program Area: The user program area manager determines which job (compilation, old job, or either of two new jobs) is to be dispatched next. To do this, it maintains two queues: the old job queue and the new job queue (for scheduling purposes only, a compilation is considered to be a new job; however, compilations take place under the control of the compiler time slice specified during system initialization).

Each queue is maintained on a first-in, first-out basis. A job is entered in the new job queue when the user initiates compilation or execution of a program, after completion of a terminal I/O operation requested by the job, or for redispatching if compilation time exceeds the compiler time slice. A job is entered in the old job queue when execution time exceeds the appropriate time slice (either old or new job time slice as specified during system initialization) or if the job requests more than a predetermined number of disk I/O operations (12 for new jobs, 30 for old jobs).

For a user program area job to be dispatched, it must be in either of two states:

- Ready to compute (if a compiler is needed, it has already been loaded)

- Actively computing (one or more increments of its time slice have already been used)

If more than one eligible job exists, the following priority scheme is used to determine which job is dispatched:

1. A new job that is actively computing

2. A new job that is ready to compute, or if both new jobs are ready to compute, the one which has been in core for the longest period of time

3. An old job that is actively computing or ready to compute

The user program area manager then determines whether background time slicing is to go into effect. If not, the user program area manager performs certain initialization steps to prepare the selected job for dispatching and informs CPID that a job is ready; CPID performs the actual dispatching.

Background Time Slicing: Time is shared with the background on a percentage basis, depending on the mode of allocation in effect. Three modes are available; the first is in effect when a CALL-OS system is initialized, the other two may be specified with the *BATCH operator command (described in the publication CALL-OS Operator's Manual). The three modes are:

- Mode 0, the automatic mode, indicates that a certain percentage of the time available to old jobs is given to the background; the percentage is based on the number of users on system and is altered automatically as users sign on and off the system; for example, for five users or less, 95 percent of the time available to old jobs is given to the background; this percentage decreases approximately 1.5 to 2 percent for each additional user.

- Mode 1, indicates that a specified percentage of the time available to old jobs is always to be given to the background, regardless of the number of users on the system; the percentage is specified in the *BATCH command and remains in effect either until altered with another *BATCH specification or until the automatic mode is restored, also with the *BATCH command.

- Mode 2, is identical to mode 1 except that the specified percentage is taken from the time available to old and new jobs.

Therefore, available old job (or old job and new job) time is shared on a rotating basis between the user program area and the background. The appropriate percentage of time is allotted on an incremental basis, determined by the size of the increment specified during system initialization.

The total number of increments allotted to an old job (or old and new jobs for mode 2) is always enough to fill the time slice specified for the job. However, the total amount of elapsed time that the user program spends in core is based on the percentage of time shared with the background, as well as the amount of shared time that the background actually used.

The total number of increments allotted to the background depends either on the number of users on the system or on the percentage of time to be given to the background. In practice, many background jobs require only a small amount of CPU time in which to initiate an I/O operation before being forced to wait for its completion.

Terminal and Disk I/O Handlers

Disk and terminal input/output operations are performed through I/O appendages; these appendages service the interrupts resulting from I/O activity. Special routines provide error handling services for disk and terminal operations.

Disk Operations: Disk input/output activity within the user program area is initiated by the use of the system Type I SVC in conjunction with an appropriate SVC code which specifies the exact nature of the I/O request. The servicing of requests for disk I/O, both for the system and the user, is accomplished by use of the OS/360 execute channel program (EXCP) facility. Examples of disk I/O activity are swapping operations, reading in nonresident modules, manipulating catalogs and directories, and handling user data files.

For disk operations, event control block posting is performed by the CALL-OS disk appendages. This precludes abnormal termination of the system due to errors that affect a minimal number of users. If a disk I/O error is user-related, the user job is aborted and a retry request is issued; if the error is recoverable, control is passed to normal OS/360 error recovery routines. Error recording in SYS1.LOGREC is also done by OS/360 routines.

Terminal Operations: As with disk I/O, user terminal I/O is performed through the OS/360 EXCP facility. Subsequent terminal I/O activity, however, is handled via the restart exit from the terminal I/O appendage. Examples of terminal I/O activity are entering source programs, using the terminal command language, printing job output, and paper tape operations.

For terminal operations, event control block posting and error recovery are performed by the CALL-OS terminal appendages. Recoverable terminal I/O errors are handled with a retry request. Unrecoverable errors are classified as such on two grounds: threshold exceeded, when the same error recurs for a specified number of times (the number, or threshold, depends on the type of error, but once established, exists until a successful operation occurs); ratio exceeded, when a specified number of errors of any type occur in a predetermined number of I/O operations on the same terminal.

Error recording in SYS1.LOGREC is done by OS/360 routines by means of a device-dependent error routine supplied during the building of the CALL-OS system. This error routine performs no error recovery, it merely signals IOS that an error should be recorded.

COMMAND LANGUAGES

One of the functions of the executive is the provision of certain terminal services to the user and the operator. To achieve this function, two command languages are provided:

- Terminal command language
- Operator command language

The modules which process the commands are part of the executive. However, the languages themselves form an integral part of the system and are therefore considered separately.

Terminal Command Language

The terminal command language is designed to facilitate communications between the terminal user and the system. These commands permit the terminal user to do the following:

- Sign on and off the system with password security
- Compile and execute source programs under CALL-OS
- Choose a compiler for compilations
- Store and execute object programs
- Create, modify, and save source-program files
- Check out user programs at the terminal
- Use routines saved in CALL-OS system libraries
- Obtain listings of line-numbered files
- Perform various edit operations on one or a number of line-numbered files
- Define and redefine data files
- Share programs and data files with other users by pooling them in system libraries
- Protect program and data files pooled by him into system libraries
- Purge program and data files from his library
- Submit jobs to OS batch processing and retrieve output at the terminal
- Enter programs and data files via paper tape
- Create paper tape output online
- Request status information

With these facilities, the terminal user can access system resources applicable to his particular problem. The terminal command language is structured so that the user can concentrate on problem solution rather than on hardware. For a detailed description of the terminal command language, the user is referred to the CALL-OS Terminal Operations Manual.

Operator Command Language

A set of commands is available for exclusive use by computer center operating personnel. Each command in this set is identified by an asterisk as the first character. The system honors these commands only if they are transmitted from a command console. If these commands are transmitted from any other terminal, they are rejected.

Through these commands, the operator can install and remove users, transmit messages to users, enable and disable lines, determine user status both for CALL-OS users and jobs submitted to OS through COBI, and terminate system operation.

For a detailed description of the operator command language, the user is referred to the CALL-OS Operator's Manual.

PROCESSING PROGRAMS

Processing programs consist of compilers, utility programs, and user programs; only compilers and utility programs are discussed here.

Compilers

Three unique compilers offering three applications-oriented, programming languages are available with the CALL-OS system. From the standpoint of the user, applications-oriented languages are vital in extending computer usage to the noncomputer professional, thus paving the way for the uninitiated person to avail himself of the computing or processing capability. In CALL-OS, the user may define his problem solution in a choice of languages:

- CALL-OS BASIC - simple to learn and use
- CALL-OS FORTRAN - scientific/engineering language
- CALL-OS PL/I - flexible power for any problem

Compilation of a user source program can be invoked by entering a RUN or STORE command at a terminal; the result of compilation is relocatable object code, which can be executed immediately and/or retained in the user's library for execution at later times. It is also possible for one user program to initiate the running of a succeeding program (compilation and execution, or only execution if the program has been compiled previously) via program chaining facilities. Communication between the compilers and the system executive is handled by the user program area manager through a Type I SVC.

The compilers are noninterpretive and reentrant. By definition, reentrant code does not alter itself during execution; thus, the same copy can be used over and over again and/or concurrently. The core and time savings due to reentrancy are very real and important. In addition, one compiler may be compiling several programs at the same time but be at a different point in the process for each program.

The object programs produced by these compilers are dynamically relocatable. This means an object program may be swapped out of one portion of the user program area during execution, and swapped into another portion whenever execution is resumed.

The compilers process the source code in one pass, permitting fast compilation speeds. CALL-OS BASIC and CALL-OS FORTRAN are single-phase compilers; CALL-OS PL/I is a two-phase compiler.

Two types of data files are created by user programs within CALL-OS: external (EBCDIC) and internal (binary). The former can be created by programs written in CALL-OS FORTRAN or CALL-OS PL/I, and can be used by programs written in the language that created them. The latter can be created and used interchangeably by programs written in CALL-OS FORTRAN, CALL-OS PL/I, and CALL-OS BASIC. The created data should, of course, be in a form supported within the structure of the language used. In addition, line-numbered files (also called program-data files) can be

read as input in the same manner as external data files. Such files cannot, however, be opened for output (that is, created or modified) by an executing program.

CALL-OS BASIC Compiler: CALL-OS BASIC provides an enhanced version of the BASIC time-sharing language originally developed at Dartmouth College, Hanover, New Hampshire. The language is easy to learn and simple to use. It is ideally suited as a first-entry language, as well as for the occasional user who need not be an experienced programmer. Some highlights of the CALL-OS BASIC language include the following:

- Ability to execute a CALL-OS BASIC program in either of two levels of precision, without modification of the source code
- Availability of an image-type output format, wherein the user explicitly lays out the format for his print line
- Multiple data file support
- Powerful facilities for matrix operations and input/output

CALL-OS FORTRAN Compiler: This is similar to the most widely used and known of all higher-level scientific languages. It is convenient and familiar to the scientific/engineering technical community. Some highlights of CALL-OS FORTRAN include the following:

- Statement entry format, free-form terminal oriented
- Free-form, list-directed input/output
- Multiple data file support
- Additional special characters supported

CALL-OS PL/I Compiler: CALL-OS PL/I is a multipurpose language designed to extend the range of applications that can be handled by a single high-level language. It allows the user to enter his statement in a free-form format. CALL-OS PL/I provides the user with many of the features of the PL/I language, using the facilities of CALL-OS via a remote terminal. Some of the many advantages to the user of a combination of language and system features include the following:

- Capability to handle a variety of data types, including character strings and complex numeric data
- Extended array facilities
- Flexible, stream-oriented, input/output facilities, including list-directed, data-directed, and edit-directed data specification
- A large number of built-in functions
- User-controlled interrupt processing
- Terminal checkout (debugging) of user programs
- Ease of modification of user programs

Utility Programs

Utility programs constitute those modules within CALL-OS which are used to build, initialize, and maintain a user system. Two types of utilities are provided: online, which constitute utilities which run in the CALL-OS task area, and offline, which run at a time when CALL-OS is not running. Initialization is an online utility; the offline utilities include system build, data base creation and maintenance, and initialization and maintenance of COBI data sets.

System Build: The system build process refers to the initial establishment of CALL-OS under OS/360. At this point in time, only a preliminary allocation of resources is involved. This essentially amounts to the setting of limits to various parameters, with a final allocation of resources, depending upon the specific installation's requirement, to be made later during system initialization.

System Initialization: System initialization modules are brought into the CALL-OS task area as soon as it is allocated by OS/360. These modules set up control information, allocate storage within the CALL-OS task area, and prepare the system for execution. When finished, control is given to the executive.

Data Base Operations: Data base operations involve the creation and manipulation of the data base. Creation consists of initializing the data base and creating entries for all the data sets in the index.

Manipulation includes the backup, maintenance, and updating of the user data base; the following functional capabilities are available (note that the terms "program" and "program file" imply either a saved source program file or a stored object code file):

- Reorganize a CALL-OS user group or system group from one data base cluster to another data base cluster.
- Validate a particular user or a range of users.
- Delete certain types of CALL-OS data base records and files.
- List programs, data files, and control information from the CALL-OS data base.
- Output a program or data file from the CALL-OS data base in OS/360 format.
- Write a program or data file to tape.
- Insert or replace a program or data file from card, disk, or tape input into the CALL-OS data base.
- Merge CALL-OS directory entries from tape input or other CALL-OS directories.
- Recreate part or all of a CALL-OS data base from a backup tape.
- Account for CPU time, disk space, and terminal connect time in the equivalency file of each user group.
- Update user catalog files with respect to current COBI job status.

In addition, facilities are provided for conversion of a backup tape from the CALL/360: Standalone system to a format suitable for use under CALL-OS.

COBI Data Set Operations: Another set of offline utilities is used to initialize and maintain the COBI data sets. The COBI data sets must be preformatted before they can be used. In addition, the data sets may have to be expanded, reinitialized, or cleaned up.

CALL-OS BATCH INTERFACE (COBI) FACILITY

The COBI facility is designed to give the terminal user the capability to create OS/360 jobs and submit those jobs for batch processing under OS/360. The JCL, source program, and data for the job is saved in his user library; from there, the job is submitted, upon command from the user, to OS/360 for processing. At the time the job is submitted, the user may specify that the JCL for the job and/or specified SYSOUT data sets produced by the job be retained for scanning at the terminal.

COBI is optional and if used, is included during CALL-OS system build. Once a part of the system, a further option during system initialization makes it possible to omit COBI from the current session. However, before COBI may be used, certain requirements must be met by the installation. These requirements, along with a more detailed description of COBI operation, are contained in the section on COBI.

CALL-OS DATA BASE

The CALL-OS data base is comprised of a collection of data sets which serves as a data storage and retrieval means for CALL-OS. The data base is generated during a separate step within the CALL-OS system build procedure. Backup and recovery of the data base is achieved through the use of either OS/360 utilities or the CALL-OS data base utility.

The data base consists of three logical parts: the system base, which serves the system's needs for storage areas; the user base, which serves as the user's program and data file storage areas; and the CALL-OS Index (hereafter referred to as the index), which serves to identify all data sets used by the system.

Each part of the data base is comprised of one or more data sets which are OS/360 BDAM-compatible data sets. All CALL-OS data sets reside on an IBM 2314 or 2319 direct access storage facility and are cataloged in the system catalog. Since the system and user bases may be comprised of several data sets, they may be allocated on several volumes.

SYSTEM BASE

System base data sets are comprised of data sets which are used for compiler storage, work/swap areas, and overlay module storage. These data sets are allocated during the data base build procedures, and are formatted or manipulated through the use of either OS/360 utilities or CALL-OS utilities.

COMPILER DATA SETS

Compiler data sets contain, in loadable form, a compiler or a compiler phase, each of which is contained on a single data set. A compiler data set varies from 1 to 20 tracks in size and must not cross cylinder boundaries. The data sets are allocated and formatted during the data base build procedures. The compilers are written as one contiguous record in track overflow mode, because of their high usage factor. Therefore, these data sets must not have multiple extents or contain alternate tracks.

Compiler data sets are OS/360 BDAM-compatible and may be backed up through the use of OS/360 or CALL-OS utilities. The ddnames used for these data sets must be the names of the compilers which are to be used. In the examples in this manual, the ddnames are given as BASIC, FORTRAN, PLI, and PL2.

WORK/SWAP DATA SETS

Work/swap data sets provide a work area for storing and swapping of user programs and data during online operations. These data sets are allocated during data base build. They are identified by unique ddnames of the form SWAP00 through SWAP19. Each user terminal is assigned one cylinder of work/swap space at each initialization of the online system. These data sets must therefore start and end at cylinder boundaries. Swapping is done in track overflow mode, so these data sets must not have multiple extents or contain alternate tracks.

Up to 20 different work/swap area data sets can be allocated and entered in the index. Since a great portion of all I/O involves these data sets, they should be spread across as many drives as possible. They should be placed on the volumes as close to the center of the data base as possible. During system initialization, any combination of the work/swap data sets available may be defined. However, the amount of space in all data sets must allow one cylinder for each line on the system.

A cylinder of work/swap space is divided into two major areas: the work area which is used to record incoming source data, and the swap area which is used for swapping of user areas (see Figure 3).

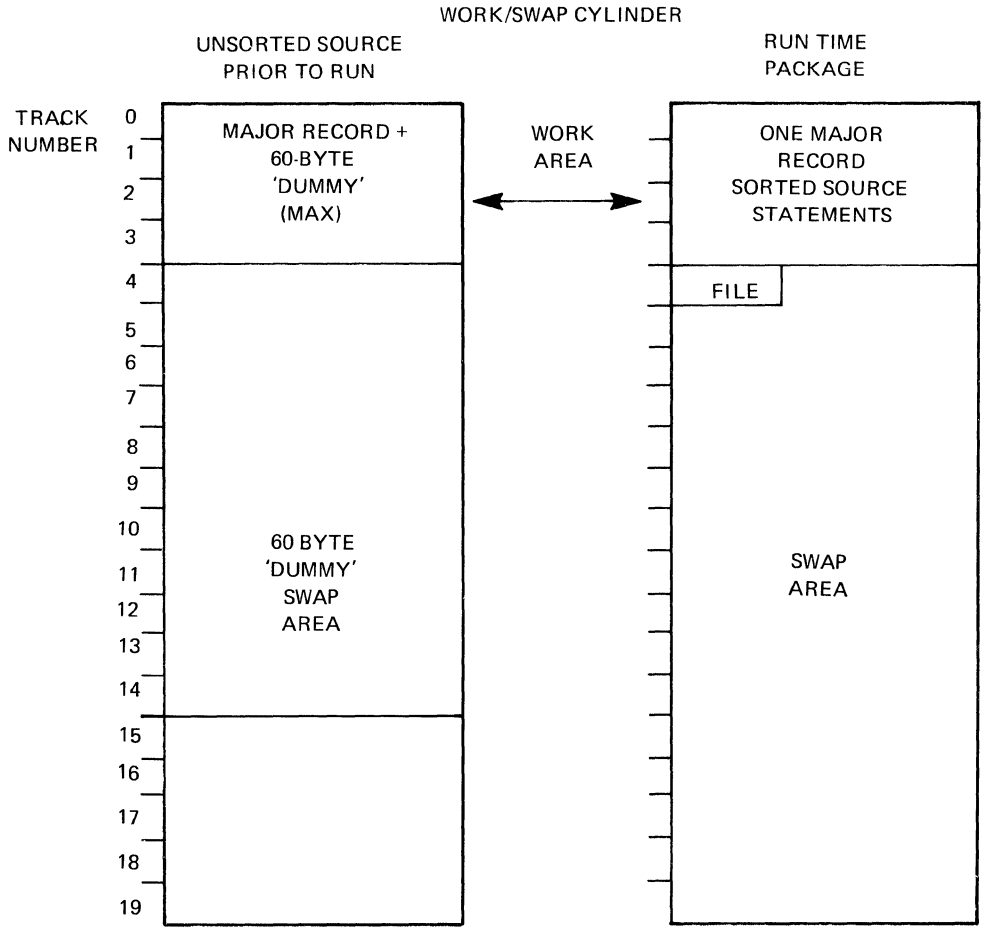


Figure 3. Disk work/swap area format

Work Area

When the CLEAR command is given or when the user signs on the system, the first track on the cylinder is formatted into 60-byte dummy records (containing no valid data) as shown in Figure 4.

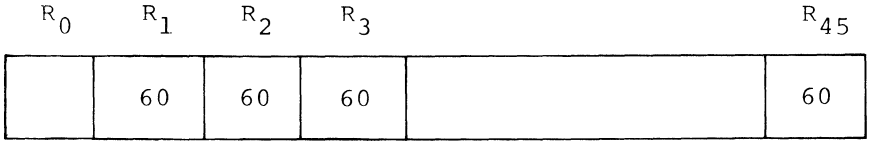


Figure 4. Initial format of the work area

As line-numbered input is entered at the terminal, the dummy records are replaced by 60-byte records containing source data from three 20-byte pots. Except for the first character in each terminal line, which is always a count byte (in binary), the contents of each 60-byte disk record are in EBCDIC code. The characters following the count are the line number.

A new line always begins at a 20-byte boundary. The end of a line is denoted by the EBCDIC character N/L (X'15'). Any space between the end-of-line character and the next 20-byte boundary will, in practice, be occupied by previous information (called a "fill"). Note that CALL-OS does not pad this space; the original information entered there is left intact. The source lines will have been edited, at this time, for character erasures. A logical record may cross a physical boundary.

Note that the user may build up to 15 tracks of 60-byte disk records. Whenever a sort takes place, however, the final packed format cannot exceed four tracks.

When a user gives a command which involves a sort (such as RUN, LIST, or SAVE), the 60-byte disk records containing the line-numbered entries are read into core and sorted. After the sort, the lines are written back to disk as one major record, while the remainder of the track is filled out with 60-byte dummy records as shown in Figure 5. This major record contains packed lines with no fills in the line. The first byte of the major record is the count byte of the first line. The last byte of the record is an EOF (end-of-file) character, which is an X'01'. Figure 6 shows a sample format of a major record.

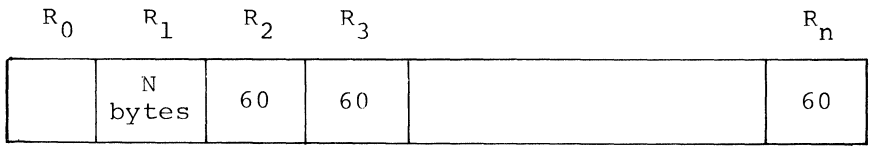


Figure 5. Work area following a sort

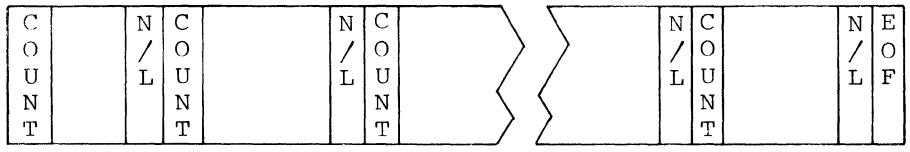


Figure 6. Format of a major record

When the user adds more source lines to a source program, they are written into the 60-byte dummy records. The source program entered at the terminal, as it resides in the work area at a particular time, may consist of a major record alone, a mixture of a major record and one or more new 60-byte records, or only new 60-byte records. Figure 7 shows a sample major record with three 60-byte add-on records.

In situations where a sort is not needed (for example, when a LIST is followed by a RUN), only the major record is read in, and the sort is not performed. When the major record requires an entire track, the second track is used to hold 60-byte records. If, after a sort, the major record is greater than one track, the remainder of the major record is written onto the second track, and the second track is filled out with 60-byte dummy records.

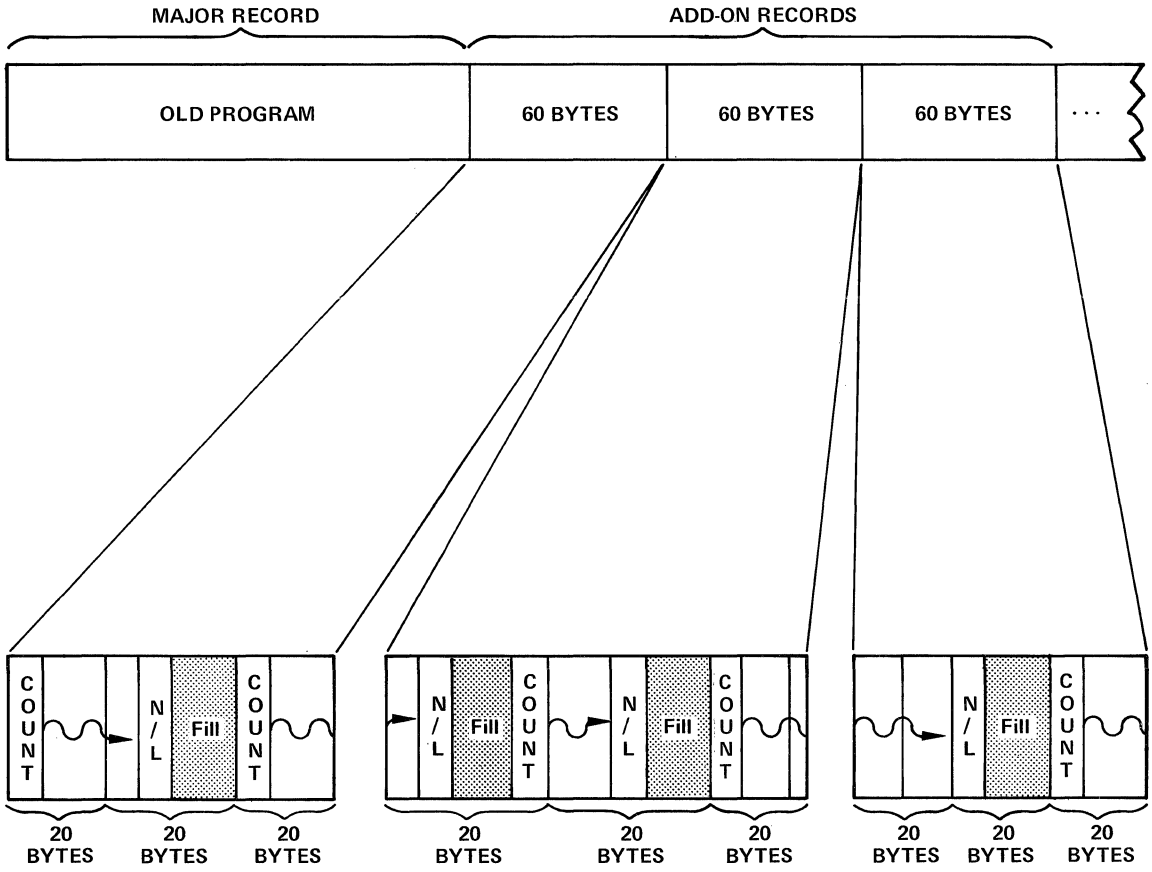


Figure 7. Major record with three add-on records

Swap Area

The swap area portion of a cylinder of work/swap space assigned to a user terminal occupies the last 16 tracks of the cylinder. This area is used for swapping the user's program area from the user's memory to disk. The swap area is divided into two parts. The first is a recording area used for disk address pointers to program files. The second is the actual swap area. These areas are defined as follows:

First Record of Swap Area — The first record of the swap area for each user terminal is a 2048-byte record with no key. This record is created and used by the executive to carry disk addresses for program files. Each logical data file is assigned 512 bytes. The first logical file is assigned the first 512-byte area, the second file is assigned the second area, and so on, up to the maximum of four data files.

When a data file is opened, the disk addresses for that data file are obtained from the appropriate catalog file. The addresses are placed in the proper 512-byte area, starting with the first byte. Each

disk address is in the relative-data-set-relative-track form (DTTR) and occupies four bytes. The disk address for the last record for each of the files is followed by X'FFFFFFFF'. The upper limit for the number of records in a data file is 100. (For further information, see the discussion of the DFLINK parameter in the description of the run-time options in the section, "Initializing the System." There can, therefore, be no more than 404 bytes consumed by disk addresses and a word of X'FFFFFFFF'.

User Program Swap Area

The balance of the first track (5054 bytes) and the 15 remaining tracks of the cylinder are used for swapping the run-time package, communications area, data area(s), and object code. The maximum size program that can be accommodated, therefore, is 5054+(15x7294)=114,464 bytes. Because user memory is allocated in 2048-byte blocks, this number must be rounded down to the nearest multiple of 2048, which is 112,640, or 55 blocks of 2048 bytes.

At swap time, only as much of the user program disk swap area is used as is necessary. The total number of bytes transferred will be a multiple of 2048 bytes (all of the user's allocated memory).

The records do not contain keys, and are written with the write special count key and data commands. This allows them to be read with one (if less than 65K bytes) or two (if greater than 65K bytes) CCWs.

OVERLAY DATA SET

The overlay data set, used for overlay modules, is allocated during data base build and occupies up to one cylinder disk storage space. At each online initialization of the system, those modules not specified in the RESMODS data set are written to the overlay data set in loadable form. Each overlay module is written at the EXCP level as one contiguous record and must not exceed 7294 bytes of data.

Backup and recovery of this data set is not required, since the data set is dynamically recreated each time the system is initialized. The ddname for this data set is OVLY. This data set need not exist in CALL-OS systems which always operate in the total-resident mode; in this case, the data set is not represented by an entry in the index.

USER BASE

The two classifications of user base data sets are system group data sets, which contain systemwide information, and user group data sets, which contain user-oriented information. Data sets within each classification may be divided into two clusters: cluster one is the primary cluster, cluster two is the alternate. Clusters permit a user installation to group its data sets in alternate ways, and/or allow a system or user group to be allocated on alternate data sets for backup purposes.

The system group and each user group in a cluster may have up to 40 data sets. The data sets are identified by group and relative data set number. The relative data set number is used to sequence data sets within the same group and to differentiate between the clusters. For example, the relative data set numbers range from 00 through 39 for the primary cluster and from 40 through 79 for the alternate cluster.

SYSTEM GROUP

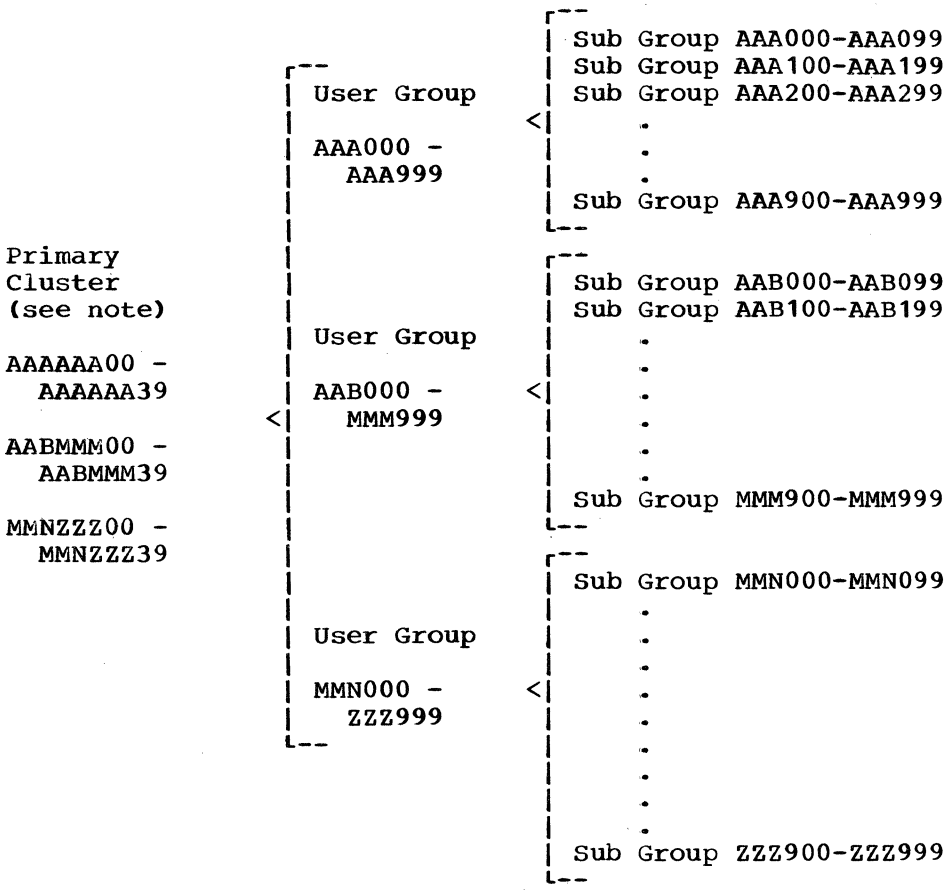
The system group contains the system (***) catalog and its associated programs and data files, as well as the system (**) directory of shared programs and data files. The system catalog is manipulated only from the command console or with the data base utility; programs and data files are entered from this console with the terminal command language. The system directory is manipulated from either the command console or a remote terminal; names of programs and data files to be shared with the rest of the system are added to the directory with the terminal command language. All of the information in the system group is available to the entire system.

The system group has a user number and password associated with it. The user number is SYSLIB and may not be changed. A newly created system group has a default password. This password should be changed periodically to a new installation-selected password by means of a PASSWORD command, issued from the command console. The system group password is not required to sign on as a command console user, but it is required to access the system group by means of the data base utility. (See "Ensuring File Security".)

To run CALL-OS, the system group must be present. The system group must have at least one and may have up to 40 data sets in any one cluster. System group data sets are defined by DD statements with names of the form SYSGRPnn, where nn is the relative data set number. The DD statements must have names from SYSGRP00 through SYSGRP39 in the primary cluster, and from SYSGRP40 through SYSGRP79 in the alternate cluster.

USER GROUP

The organization of user group data sets is based on three logical groupings: a set of 99 users (sub group), a set of sub groups (user group), and a set of user groups (cluster). Figure 8 illustrates this logical organization schematically.



Note: Corresponding names for the alternate cluster are:

AAAAAA40 - AAAAAA79
AABMMM40 - AABMMM79
MMNZZZ40 - MMNZZZ79

Figure 8. User group data set organization

A sub group is a set of 99 users whose user numbers have the same first four characters. For example, a sub group could consist of the set of users with the numbers AAA101 through AAA199. The sub group is the smallest set of users for which sharing of program and data files is allowed. File sharing is made possible through the use of the special user number which has zeros as the last two characters. When this user number is validated, the system creates a dummy user catalog (called a directory).

The directory for a sub group is the *Directory and it corresponds to the *Library. The *Directory serves the same purpose as the directory of a partitioned data set. It contains the names of each shared file for the sub group and the user number of the sharing user; the file itself remains in the library of the user who shared it. Thus, for example, if user number AAA100 is validated, a *Directory is created for the sub group of users whose numbers are within the range AAA101 through AAA199. Note that the *Directory for a sub group does not exist unless the proper number is validated.

A user group consists of a range of user numbers (such as AAA000 through CCC999). Only user numbers ending with 000 or 999 can serve as bounds. This range can include as few as 1000 user numbers (AAA000 through AAA999) or more than 17 million (AAA000 through ZZZ999). Each user group is defined by the user installation to include a set of related users. Each user group is assigned to a collection of single-volume OS/360 data sets (not all of which need reside on the same volume).

These data sets are defined by DD statements with unique names of the form aaazzz00-aaazzz39 in the primary cluster, and aaazzz40-aaazzz79 in the alternate cluster, where aaazzz is the name of the user group. To designate a valid user group, the first three letters of this name (aaa) must precede the last three letters of the name (zzz) in the collating sequence. Thus, RAATZZ is a valid user group name, and supports any user number from RAA000 through TZZ999.

User group data sets are used for the storage of catalogs, programs, and data for all users in the group. A given user is assigned disk storage space only from the data sets assigned to his group, thus ensuring data set integrity. User group data sets are governed by the following restrictions:

- Identical or overlapping user groups from different clusters must not be opened for the same session (see the discussion on how to use clusters later in this chapter).
- No multivolume data sets are allowed.
- Space is assigned on behalf of a group of user numbers; more than one data set can be assigned to the same group of users.
- Space is dynamically allocated from preformatted data sets.
- For program and data files, space is reallocated only to a user who has purged space, if the required amount is available.

STRUCTURE OF EACH USER BASE DATA SET

A system or user group's data storage is allocated from the data sets assigned exclusively to that group at data base build time. These data sets will normally contain the following:

- Allocation record, which contains a record of the amount of usable space left in the data set
- Equivalency file, which is a linked file of user numbers which have been validated for the group; each user number entry has a pointer to a user's catalog
- Catalog, which is a linked file which contains all the program and data file names saved by the user; the location and file length are included
- Directory file, which is a linked file which contains a list of file names and user numbers for pooled program and data files
- Source program file, which contains user source programs; these files are organized in several sizes and are stored in the smallest size available which can contain the entire program
- Object program file, which contains user object programs; object program files are stored in half tracks

- Data file, which contains input and output data for running programs; data files are stored in half tracks
- Program-data file, which contains user line-numbered information; these files are organized in several sizes and are stored in the smallest available size that can contain the entire file
- File descriptor record, which is used for every object program file and for any data file longer than four records

Note: Program-data files are distinguished from source-program files only by usage. No structural differences exist between the two types of files. In this manual, unless otherwise specified, the term program file should be assumed to refer to both source-program files and program-data files.

The first data set in each system group must have at least three tracks, while the first data set in each user group must have at least two tracks. All other user data sets may be as small as one track, although larger data sets (one cylinder or more) are recommended for optimum performance. These data sets may have multiple extents if they are obtained in the initial allocation; CALL-OS does not attempt to use a secondary allocation.

Allocation Record

One allocation record exists for each system group and user group data set. By convention, it occupies the first record of the first track of the data set. Its function is to provide a starting point for the CALL-OS initialization routine. The allocation control table associated with each user base data set can then be initialized.

These tables are updated in core while CALL-OS is in operation, and reflect the next available tenth, fifth, half, and full track in the data set. When a user purges a program or data file, this space remains assigned to him and the allocation table does not reflect the purged space. During normal system termination processing, the most recent allocation information is written back into the allocation record for the data set.

Equivalency File

An equivalency file is a linked file of records which contain all the user numbers validated for a group. Each group has one equivalency file which begins in the second track of the first data set in the group. Up to 200 user numbers can be indicated before this record is chained to another full-track record somewhere within the group.

When a user signs on the system, the equivalency file for his group is checked to confirm that his user number is validated. The entry associated with a validated user number also provides the beginning address of the user's catalog. In addition, the equivalency file entry for a user contains a total space allocation record and total terminal and execution times.

Each physical record of this file occupies one full track. A record consists of a 36-byte key and a 7212-byte data field that can be forward-linked to the next record of the file. The first physical record of the file, by convention, resides on the second track of the first data set assigned to the group. Each record entry is 36 bytes long. There are twelve bytes at the end of each physical record. The bytes are always read or written, but are ignored for searching

purposes. The EOF indicator for the file is a full word containing X'01010101'.

Catalog

The catalog file is a linked file which contains information about each program and data file retained by the user. In addition, when COBI is present in the system, the catalog contains information about each job submitted by the user to be run under OS. Finally, the catalog defines space that is reusable because program and/or data files have been purged from the data base. One catalog file exists for each user number validated onto the system.

The catalog for each user consists of one or more records, each of which can be forward-linked to the next record in the catalog. Each record occupies a half track and consists of a 36-byte key field and a 3440-byte data field. The data field of each record can contain up to 95 entries, each of which is 36 bytes long. One entry is provided for each program and data file kept by the user, as well as each COBI job submitted by him. The information in each entry depends on the type of entry.

For each program and/or data file, the catalog entry contains the name of the file, its length, and up to four disk addresses which give the location of the file. The catalog entry for a source program always contains the entire set of disk addresses for the storage areas occupied by the program (the maximum program size is four tracks). The catalog entry for an object program always contains the address of a record called the file descriptor record; this record in turn points to the storage area(s) for the object program. If a data file is four or less records long, the catalog entry contains all the disk addresses for the file; if a data file is longer than four records, the catalog entry contains the address of a file descriptor record. Source program, object program, and data files are discussed later in this section along with file descriptor records.

For each COBI job, the catalog entry contains the COBI job number (in the form #nnnnn) and the job name under which the job is known to OS/360. This is the only information about COBI jobs that appears in the data base. The jobs themselves reside in OS/360 data sets.

Note: The catalog for the system group has the same physical characteristics as other user catalogs and serves as the ***Catalog for the entire system.

Directory File

The directory file is a linked file which contains a list of file names and user numbers for shared program and data files. The function of this file is to allow contributing users to share (pool) programs and data files. These files are shared either with other members of the same sub group (*Directory) or with all validated users on the system (**Directory). The *Directory resides in the user group data sets; the **Directory resides in the system group data sets.

The *Directory file facilitates sharing of programs and data between members of a sub group. There may be one such directory for each sub group. The directory is created by validating a user number with its last two numeric digits 00. This directory is then available for use by all members of the same sub group. An entry in the equivalency file is created, which points to a newly-created directory file assigned to the appropriate user range.

The **Directory file is much the same as the *Directory. That is, it contains the names of shared files and the user number of the sharing user. However, files named in the **Directory are available to the entire system. There is only one **Directory for the system.

Note: The *Directory for the system group corresponds to the **Directory for the rest of the system; that is, if the name of a file is entered at the command console for sharing in the *Directory, the name is placed in the **Directory, thereby making the file available to the entire system.

Each physical record of a directory file occupies one half track; it has a 36-byte key field and a 3440-byte data field, and can be forward-linked to the next record of the file. Each record can contain as many as 143 entries, and each entry is 24 bytes. The entry contains the name of the file being shared, the user number of the owner, the date on which the file was shared, and the address of the owner's catalog.

Program and Data Files

These files constitute the user's data base. The physical format of source program, object program, and data files is described.

Source-Program File. This file contains a user source program. It occupies a tenth, fifth, half, or full track; if more than a full track is required, additional full tracks are assigned, up to a total of four tracks. In each case there is a 36-byte key field. The actual data field sizes are 534 bytes, 1252 bytes, 3440 bytes, and 7212 bytes, respectively. The data field content is user-dependent.

Object Program File. This file contains a user object program. Each physical record of this file resides on a half track, consisting of a 36-byte key field and a 3440-byte data field. Each program may occupy up to 33 half tracks. A file descriptor record is supplied for every object program.

Data File. This file contains data used for input to and output from running programs. Each physical record of this file resides on a half track, and consists of a 36-byte key field and a 3440-byte data field. There may be up to 100 such records for each data file. If a file is longer than four records, a file descriptor record is supplied.

The data field is entirely filled with data, even though it may not all be valid data. The internal format of the data field is determined by the compiler which creates the file and is not referenced by the executive.

Program-Data File. This file is a program file, created as line-numbered input directly from the terminal, or entered into a user's library by means of a CALL-OS utility. The file can be opened for input (and included in the count of simultaneously open files) by an executing program. The line number preceding a line is not considered as part of the input; the first character following one blank (or following the line number if no blank is included) is treated as the first data character of the line. Program-data files cannot be opened for output, but other rules that apply to data files apply also to program-data files. The physical organization of a program-data file is the same as that of a source-program file.

File Descriptor Record

The catalog entry points to one file descriptor record for each data file with more than four records and for each object program. This record occupies a tenth track and consists of a 36-byte key field and a 534-byte data field. The key field contains a copy of the first 20 bytes of the catalog entry. The data field contains up to 100 disk address links for the appropriate data or object program file. File descriptor records are assigned on an as-needed basis.

Summary

The files and records within the user base data sets are logically related as follows (see also Figure 9):

- One allocation record exists for each user base data set; it is used to record total usable space in the data set.
- One equivalency record exists for each group; it begins at a known relative track location within each group. The equivalency record is accessed on validation and contains the disk address of all catalogs in this group.
- One catalog exists for every validated user number; a catalog record has one entry for every data and program file belonging to that user and, when COBI is present, one entry for every job submitted by that user.
- Source programs (and program-data files) may occupy up to four full tracks. The list of disk addresses in the catalog entry provides for sequential accessing of programs.
- Data files may occupy up to 100 half tracks. The addresses of these half tracks, if they exceed the four available in the catalog entry, are stored in a file descriptor record pointed to by the catalog entry.
- Object programs may occupy up to 33 half tracks. However, every object program, regardless of its length, has a file descriptor record associated with it.
- For user groups, directory records are treated as artificial users. The records contain the addresses of programs and data files which may be shared among users in a related range of real user numbers. The user number which ends in 00 is a dummy number. It provides the *Directory for the remaining 99 user numbers in the range indicated by the first four characters in the user number.
- For the system group, the first ***Catalog record is contained in the first record of the third track of the first data set. The **Directory records begin at the second record (half track) of the third track. Typically, the *** files are maintained by the computer center, while ** files are contributed by users of the system to be shared among all other users.

Note that none of the components of a group need be contiguous. In all likelihood, they are scattered across the full spectrum of data sets for the group.

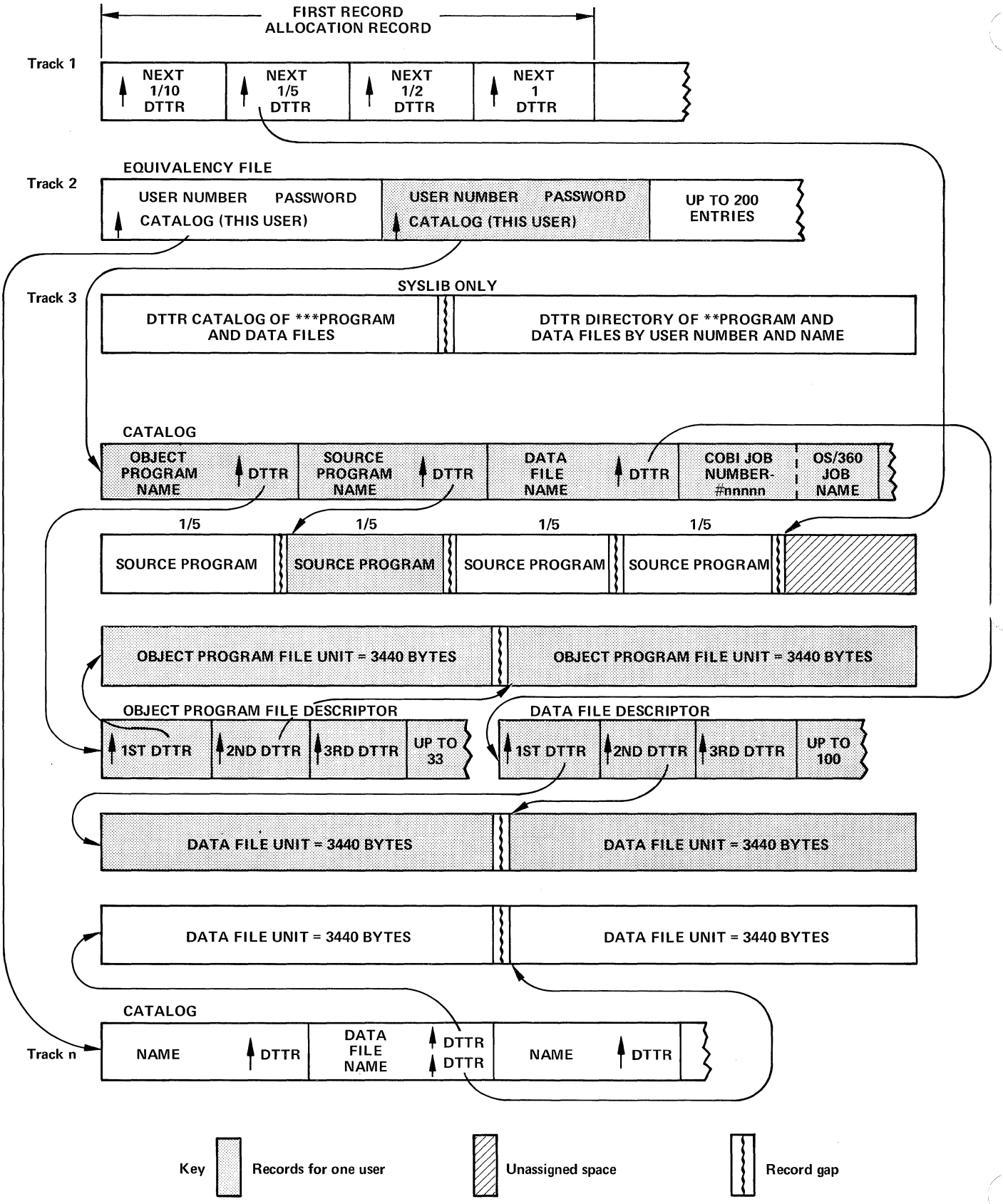


Figure 9. Relationships of user base data set records

ASSIGNING USER NUMBERS

User numbers, and how they are assigned, affect the system in many ways. The entire user number range allowable on the system is any six-character identification, starting with three alphabetic characters and ending with three numerics. This range forms an alphabetic sequence from AAA000 to ZZZ999. User storage space is allocated within a user group which consists of a minimum of one data set and a maximum of 40. A group of users is defined by a user number range based on the first three characters of the user number through some alphabetically continuous series. For instance, all user numbers starting with AAA through DZZ could be designated as group 1, and all user numbers between EAA and ZZZ could be designated as group 2, for a two-group system.

In addition to the user groups, one group must be provided for the system and must contain at least one data set. This system group is created at system generation time and is available to the command console for entry of files. The group contains the systemwide (***) catalog and files, and the systemwide (**) directory of shared files. This initial data set can be supplemented (as can all user groups) by allocation of additional data sets for this group, up to a total of 40 in each cluster.

CALL-OS can be started for any subset of groups at job initialization time. These data sets contain all catalogs, equivalency records, directories, programs and data files. Various subset sequences of this entire range may be assigned in a particular way in order to accomplish the following:

1. Ability of a small group of related users to share programs while restricting all other system users.

   Each user number that is validated, ending with the last two numeric digits of 00, creates a dummy user directory. The directory is available for the pooling and use of shared files by all users whose user numbers are identical for the first four characters. The next level of directory sharing is systemwide. This means, for instance, that all users desiring subsystem proprietary sharing must have user numbers assigned in the same 99-number band. For example, user numbers ICX701 through ICX799 have a unique directory created by validating dummy user ICX700.

2. Data set integrity.

   When disk data sets are allocated, they are attached to a cluster of data sets in behalf of some user number group. The number of groups and assignment of people to user numbers within the groups have a great deal of influence on the use of system resources. A given user is assigned space only from the data sets assigned to his group.

CALL-OS can be initiated to bring up any number of mutually exclusive groups up to a maximum of 113 index entries. Any user who attempts to sign on and whose group was not specified at startup time, is notified and disconnected.

USING CLUSTERS

CALL-OS permits the use of two clusters: the primary cluster and the alternate cluster. The system group and each user group may have up to 40 data sets in either cluster. When a group is used in a CALL-OS session, all the data sets associated with that group must be present. For example, if a user group has data sets IBMIBM00, IBMIBM01, and IBMIBM02 in the primary cluster, all three data sets must be present.

Data sets from both types of cluster may be used simultaneously if the following conditions are met:

- The data sets present for a group must belong to the same cluster. For example, SYSGRP00 and SYSGRP41 may not be present in the same session; nor may IBMIBM00, IBMIBM41, and IBMIBM42.
- The user number ranges for the specified user groups must not overlap. For example, AAADDD00 and BBBFFF40 may not be present in the same session because the user numbers from BBB001 through DDD999 belong to both user groups.

In the following example, assume that the primary cluster has three groups and the alternate cluster has four groups, as follows:

PRIMARY CLUSTER

Group User Range	Data Sets	DD Names
System Group	1	SYSGRP00
AAA000 through IJK999	20	AAAIJK00-AAAIJK19
IJL000 through ZZZ999	19	IJLZZZ00-IJLZZZ18

ALTERNATE CLUSTER

Group User Range	Data Sets	DD Names
System Group	1	SYSGRP40
AAA000 through GZZ999	10	AAAGZZ40-AAAGZZ49
HAA000 through RZZ999	10	HAARZZ40-HAARZZ49
SAA000 through ZZZ999	10	SAAZZZ40-SAAZZZ49

A CALL-OS session could use the system group from either cluster, and with certain restrictions, user groups from both clusters. For example, the SAAZZZ user group from the alternate cluster may be used only with the AAAIJK user group from the primary cluster; use of the primary cluster IJLZZZ user group would result in overlapping user numbers.

If the user group configurations in both clusters are alike, clusters may be used effectively through reorganization. For example, if one user group in a system is more active than the others, it may become necessary to reorganize this group earlier to eliminate purged space. The REORGANIZE function of the data base utility may be used to reorganize the user group into the other cluster. This reorganized user group may then be brought up with the user groups from the original cluster, thereby providing the full range of user group availability.

Another use of the REORGANIZE function with primary and alternate clusters of data sets is to keep two different configurations of the same groups. For example, take the two following configurations:

Cluster	Group User Range	Number of Data Sets
Primary	System	2
	AAA through GGG	5
	III through PPP	3
	YYY through YYY	2
Alternate	System	1
	AAA through JJJ	3
	KAA through YYY	4

The REORGANIZE function edits and reorganizes the primary into the alternate cluster in three steps:

	Primary			Alternate	
STEP A:	SYSTEM	(2)	to	SYSTEM	(1)
STEP B:	AAA - GGG	(5)	to	AAA - JJJ	(3)
	III - PPP	(3)			
STEP C:	III - PPP	(3)	to	KAA - YYY	(4)
	YYY - YYY	(4)			

The ability to accomplish this rebuild of the groupings and reallocation of number of data sets depends on the following:

1. Enough disk drives are available to mount all (from and to) data sets at one time for each run.

2. Enough space is available in the alternate data sets to contain all records in the primary.

INDEX DATA SET

The index is a one-track OS/360 BSAM data set which contains an entry for each data set associated with CALL-OS. It identifies all data sets used by the CALL-OS online system and those operated upon by the data base utilities, and describes their logical groupings. The entire track for the index is allocated and initialized to X'FF's (hexadecimal) during data base build by U#UTIL3. Data set entries are added to or deleted from the index during data base build or data base manipulation with CALL-OS offline utilities. Each index entry is 64 bytes in length and identifies one data set.

The entry for a data set begins with a unique identification code that indicates the general use of the data set, for example, compiler or system group. The next field contains the relative data set number of the data set; this number is in the range from 00 through 79. (For compiler and overlay entries, this field is always zero.) The third field indicates the specific group to which the data set belongs (for identification purposes, compiler, overlay, and work/swap data sets are also considered to be "groups"); the groups and their associated identifications are:

- Compiler, identified by the first six letters of the compiler or compiler phase name (for example, BASIC, FORTRAN, PLI, or PL2)

- Work/swap, identified by SWAP; up to 20 such entries may appear in the index

- Overlay, identified by OVLY

- System group, identified by SYSLIB; up to 40 such entries may appear in each cluster

- User group, identified by the user number range of the group (for example, AAACZZ indicates that the user number range is AAA000 through CZZ999); up to 40 such entries may appear for each user group in each cluster

The next two fields of the index entry contain the name of the DD statement which defines the data set and the full data set name, with all qualifiers. The last two fields contain status information for the initialization and formatting utilities, and for compilers, the size of the compiler.

The index contains an entry for each DD statement and associated data set known to both the online and offline systems. The maximum number of index entries is 113; any or all data sets associated with these entries may be defined during system initialization. That definition determines which data sets are to be available during a particular CALL-OS session. This includes any combination of compiler, overlay, work/swap, system group, and user group data sets. For the system group and each user group, although the index itself may have entries for the group in both clusters, only one cluster may be represented for each group during online operation. Figure 10 shows the overall organization of the CALL-OS data base.

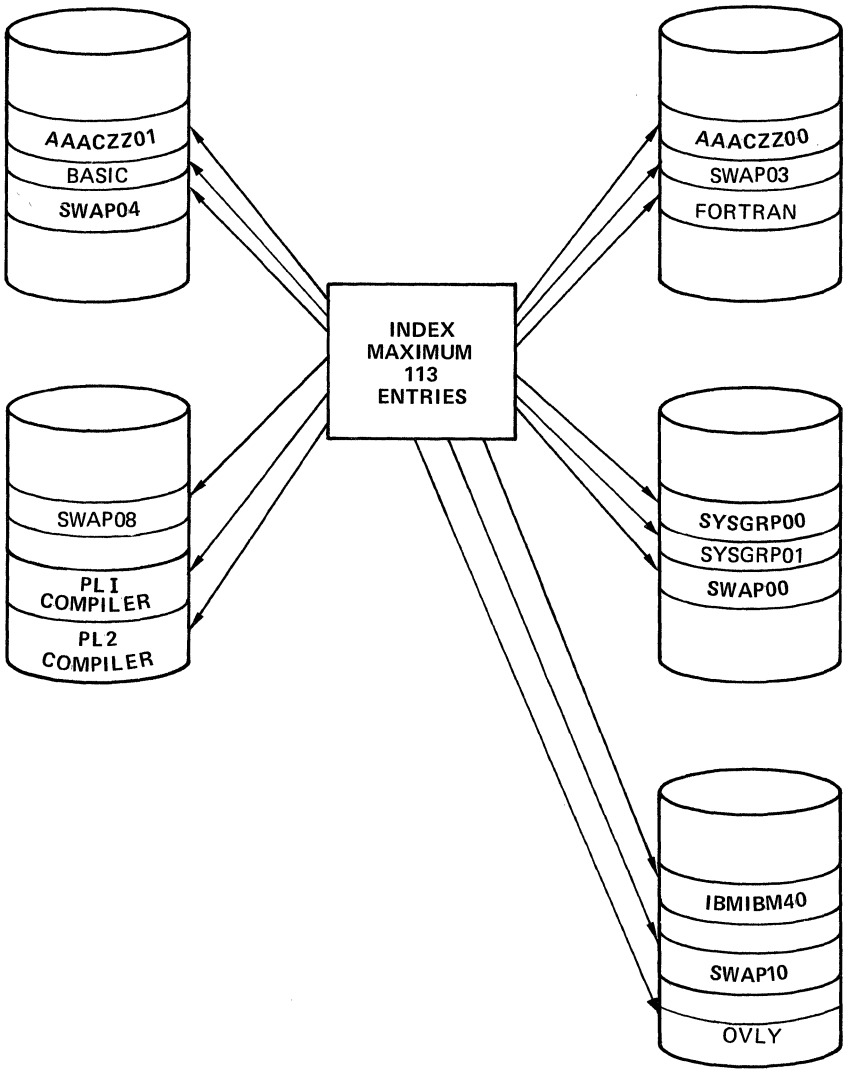


Figure 10. Organization of CALL-OS data base

DATA BASE AND SYSTEM PERFORMANCE

In the planning of system resources, the user should remember that since CALL-OS runs as a high-priority job in a multiprogramming system, the way to increase the throughput of the background activity is to restrain CALL-OS from developing too large a load. This can also be achieved by reducing the number of lines enabled and by adjusting the time slice allocated to background work.

It is also possible to assign availability priorities to groups of users and thus limit the use of system resources. The only problem created by bringing up less than the whole system is that programs available through the **Directory are not available if the user who owns the program is not brought up. The process of bringing groups up and down is not dynamic. The system must be shut down and be restarted to change user configurations.

LIMITATION OF DISK SPACE

Planned availability and priority of the uses of limited amounts of disk space may be desired for some logical or accounting breakdown. User numbers can then be assigned to guarantee a given amount to a particular group. Otherwise, within a group, allocation of space is on a first-come, first-served basis. A given user retains exclusive use of his purged space and, where feasible, this space is reused. Any purged space is available for use by the user until an offline reorganization run is made to return space to the entire group; the purged space is then made available to all users in the group.

PLANNED EFFICIENCY OF DISK ARM USE

For accounting convenience, it may be advantageous to have one data set allocated to each user group; however, disk arm movement on high-usage user groups may impair performance. A substantial performance improvement may be realized for such user groups by allocating them to multiple data sets. For example, one high-usage user group may be allocated to eight data sets on eight different disk modules. The system rotates the allocation through all data sets, filling them all up together and spreading online arm usage across eight disk modules. This assumes that all data sets are initially allocated with an equal amount of space. Lower usage groups, however, can be allocated wherever it is convenient to locate them on disk volumes.

BACKUP OF THE DATA BASE

Backup of the data base data sets can be achieved through OS/360 utilities as well as CALL-OS offline utilities. In general, system base data sets, which are easily created and do not often change during a CALL-OS session, do not call for maintenance in the normal sense of the word. User base data sets, on the other hand, require frequent backup, which can be accomplished in any of the following ways:

1. Through the use of the OS/360 DUMP/RESTORE utility on all packs which contain data base data sets. The backup copy can be on tape or on a 2314 or 2319 direct access storage facility. Note that the format of the restored pack(s) is such that the backup copy can be used to bring the system online.

2. Through the use of the OS/360 IEHMOVE utility to copy all data sets for a user group. The format of the copied data set is compatible with CALL-OS format and structure and can be used to bring the system online.

3. Through the use of the data base utility REORGANIZE function to provide a compressed copy of a user group situated in one cluster of a data base to a single, unused, preinitialized user group in an alternate cluster, deleting purged or unused disk space in the process. This is a disk-to-disk operation.

4. Through the use of the data base utility TAPE function to provide a backup tape. This tape may then be used with the data base utility RECONSTRUCT or INSERT/REPLACE funtion to recreate the data sets.

The use of OS/360 utilities is described in the publication IBM System/360 Operating System: Utilities. The data base utility functions are discussed in detail in "Creating and Maintaining the Data Base".

REMOVING A USER FROM THE DATA BASE

It may become necessary to remove the files associated with a particular user from the data base. The *CANCEL command is not sufficient for this purpose because the files for a cancelled user remain in the data base. One method of removing such files is to manually purge all the files with the PURGE command before issuing the *CANCEL command. A second, and better, method is to use the automatic purge facilities of the DELETE function of the data base utility. The DELETE function also produces an archive tape of the deleted files and removes * and ** directory entries pooled by the cancelled user.

Even if all of the user's files have been purged, the cancelled equivalency entry and one empty catalog half track remain in the data base after a reorganization. A dormant user number such as this may be reissued to a new user at a later time by revalidating the user number with the *VALIDATE command. If it is desired to remove all dormant user numbers from the data base, the RECON function of the data base utility can be used. When /$ range cards are used during a reconstruction of the data base from a backup tape, those users which are no longer required may be omitted from the cards. This completely eliminates these users from the system, including the files, the equivalency entry, and the half-track catalog record associated with each user.

The data base utility functions are described in detail in "Creating and Maintaining the Data Base".

CALL-OS BATCH INTERFACE FACILITY

This section describes the CALL-OS Batch Interface (COBI) facility, which permits CALL-OS terminal users to generate OS/360 jobs and submit those jobs to OS/360 batch processing. The introduction to this section describes general concepts important for an understanding of COBI and follows a sample job through COBI and OS/360 processing. The rest of the section describes the COBI data set requirements, the preparatory steps necessary to use COBI, and the maintenance utilities for the COBI data sets.

COBI is optional, and is included in the system during CALL-OS system build. In addition, an initialization option permits a COBI-built CALL-OS system to be initialized without the COBI facility. Additional information on COBI is found in other sections of this publication, as follows:

- COBI storage requirements are given in "Designing the System".
- The process of building a CALL-OS system with COBI is described in "Building the System".
- The initialization options and DD statement requirements applicable to COBI are described in "Initializing the System".

Before using the additional information on COBI, the user should be familiar with the information in the following text.

INTRODUCTION

The COBI facility provides the CALL-OS terminal user with the means to submit a job for OS/360 batch processing. The user may request that JCL statements, system messages, and SYSOUT data sets created during execution of the job be saved for printing (scanning) at the terminal. COBI saves the JCL and system messages by copying them from the appropriate output queue to the COBI JCL data set. COBI saves the requested SYSOUT data sets by providing DSNAME, SPACE, DISP, and UNIT parameters for specified DD statements according to user-specified options when the job is submitted; those SYSOUT data sets not saved by the user are assigned to an output class for processing by OS/360. After a job has been submitted, the user may inquire into the status of his job and the scannable data sets.

At the time the job is submitted, a 48-byte record in the COBI index data set is assigned to that job. A job number of the form #nnnnn is assigned to the job, where nnnnn corresponds to the record number of the COBI index record. The job name and identifiers of scannable SYSOUT data sets are recorded in the COBI index record. Upon completion of the OS/360 processing of the job, additional information is placed in the COBI index record.

In order to determine when a submitted job has completed execution, save its JCL, and locate scannable SYSOUT data sets, an OS/360 output class must be assigned solely for COBI submitted jobs. A COBI writer program periodically examines the output queue of that class for submitted jobs which have completed execution. For each completed job found in the queue, the COBI writer determines if the job is a COBI-submitted job. If not, the job output is reset to an output queue for processing by an OS/360 writer.

A COBI-submitted job is recognized by scanning the output queue for a JCL comment statement immediately following the JOB statement. (This comment statement is inserted by COBI when the job is submitted; the statement contains the job number, the job name, date of submittal, and other control information.) The COBI writer extracts the job number and retrieves the related COBI index record.

If either the JCL was requested to be saved or a JCL error was detected by OS/360, the JCL and system messages are copied into the COBI JCL data set and a pointer to the JCL is placed in the COBI index record. If SYSOUT data sets were requested to be saved and the data sets were created and kept by OS/360, the identification of the volumes containing the data sets are stored in the COBI index record. User and/or system completion codes for the job, if any, are also stored in the COBI index record. Finally, the job's output in the COBI class queue is reset to an output queue for processing by an OS/360 writer. The JCL, system messages, and SYSOUT data sets not saved for scanning would then normally be listed on a printer.

After a job has been processed by the COBI writer, the COBI index record contains sufficient information to respond to user inquiries for the status of his job, and the status and scanning of scannable data sets. The user may also request that the job, the JCL, or SYSOUT data sets be scratched. The scratching of a job is accomplished by clearing the space occupied by its JCL in the COBI JCL data set, scratching the DSCBs of the scannable SYSOUT data sets, and, finally, clearing the space occupied by the job's COBI index record.

COBI CONCEPTS

After a COBI job has been read into an OS/360 job queue, it is executed in the same way any non-COBI job is executed. The major difference in the overall processing of a COBI job is that the user may request scanning of data sets associated with his job. The way in which scanning is made possible involves several important concepts; an understanding of these concepts is necessary to understand COBI operation as a whole.

Output Classes

When CALL-OS is initialized, an output class is designated for the routing of output from COBI-submitted jobs. It is assumed that only these jobs will have output in the COBI class output queue. The CBCLASS initialization parameter is used to assign this class to CALL-OS. If the parameter is omitted, the default is class Z; this class is used in subsequent examples.

The COBI class output queue contains the JCL and system messages for all COBI-submitted jobs, as well as pointers to any temporary SYSOUT data sets not saved for scanning. This output queue is monitored by the COBI writer for COBI-submitted jobs which have completed execution. When such a job is detected, the JCL and system messages are saved, if necessary; pertinent information regarding job execution and the location of scannable SYSOUT data sets is also recorded. The job output in the COBI output class queue is then reset to an output queue to be processed by an OS/360 writer.

The output queue containing the reset output is processed in the normal manner by an OS/360 output writer started by the operator for that output class. This class is designated by the OSCLASS initialization parameter. If the parameter is omitted, the default is class A; this class is used in subsequent examples. The OSCLASS and CBCLASS parameters must not specify the same class.

Submittal of OS/360 Jobs

The terminal user must enter the JCL, source program, and data for an OS/360 job as source lines from the terminal; he then saves these source lines as a program file in his library. He issues a SUBMIT command to direct COBI to format and copy the OS/360 job from the specified program file(s) to the COBI SYSIN (input) data set. This data set can then be processed by a COBI reader program which reads the jobs into the OS/360 job queue.

When the SUBMIT command is issued, the user may specify that JCL and SYSOUT data sets be saved for scanning. The SYSOUT data sets are designated by data set identifiers and may be of either of two types: user-defined data sets and procedure-defined data sets. User-defined SYSOUT data sets are identified by a parameter of the form Unnn, where nnn is a number chosen by the user ranging from 1 through 127. Procedure-defined SYSOUT data sets are identified by a parameter of the form nPmm, where: the value of n may range from 1 through 7 and indicates which procedure within the job contains the desired SYSOUT data set; the value of mm may range from 1 through 15 and indicates which SYSOUT data set within the procedure is to be saved. For example,

```
SUBMIT  PGMA,(JCL,U1,2P1)
```

This SUBMIT command specifies a CALL-OS program file named PGMA which contains an OS/360 job. The JCL is to be saved for scanning, as well as two SYSOUT data sets: user-defined data set U1 and procedure-defined data set 2P1 (the first SYSOUT data set in the second procedure executed by the job).

Identifying COBI Jobs and Data Sets

COBI assigns a job number of the form #nnnnn to every submitted job. This is true for all submitted jobs with one exception: if the user did not specify any data sets to be scanned when he submitted the job and if the JOB statement contains a MSGCLASS parameter referring to other than the COBI output class; in this case, no record of the job is kept by COBI.

If desired, COBI will also assign a job name to every job. The job name consists of the user number (of the form aaannn, also referred to as userid) of the submitting user and a two-character identifier obtained by hashing the COBI-assigned job number.

The user number, the job number, and the scannable data set identifier described previously are used to ensure unique data set names for scannable data sets. COBI uses these three pieces of information to create a qualified data set name of the form:

```
DSN=DIB.userid.job-number.data-set-identifier
```

where

userid	is the user number, of the form aaannn, of the submitting user
job-number	is the COBI assigned job number, of the form #nnnnn
data-set-identifier	is the parameter specified in the SUBMIT command, of the form Pnmm or Unnn. Note that the user-specified identifier nPmm is used in the form Pnmm because data set names may not begin with a numeric character.

Therefore, the user need only refer to a data set by its data-set-identifier and COBI will generate a data set name to be used by OS/360 for space allocation and output processing.

Definition of SYSOUT Data Sets

As mentioned briefly before, a scannable data set is defined by a COBI-supplied DD statement that has DSNAME, SPACE, DISP, and UNIT parameters. These parameters identify the data set as an OS/360 data set which is to be created and kept for scanning. The DD statement contains only a SYSOUT parameter if the data set is not to be saved for scanning. Scannable data sets may be defined in cataloged procedures or in user-provided DD statements. The technique used by COBI to supply DD statement parameters depends on the way in which the data set is defined.

User-Defined SYSOUT Data Sets: To save a user-defined SYSOUT data set for scanning, the user specifies a parameter of the form Unnn in the SUBMIT command. For every Unnn parameter specified, COBI scans the submitted job for a DD statement which contains the parameter &Unnn. For example, if the user had specified the parameter U1 on a SUBMIT command, COBI expects to find a DD statement in the job of the following form:

```
//ddname        DD       &U1
```

If COBI finds a DD statement parameter to match the parameter on the SUBMIT command, it replaces the DD statement parameter with the following information:

```
DSN=DIB.userid.job-number.Unnn,SPACE=allocation,
DISP=(NEW,KEEP),UNIT=device-class
```

where

userid, job-number, and nnn are as described previously

allocation — is a space allocation for the data set of the form (TRK,(primary,secondary),RLSE) where primary and secondary are either the allocations specified in the DSPACE initialization parameter or defaults of ten tracks each

device-class — is either the class of devices specified in the UNITNM initialization parameter or a default of 2314; for more information on device class and its use with COBI, see the subsection "COBI Device Class"

While scanning a submitted job, if COBI encounters a DD statement parameter with no matching SUBMIT parameter, the DD statement parameter is replaced with SYSOUT=Z. This data set is then routed to the COBI output class, from which it is reset to the OS/360 output class for processing.

In the following example, assume U2 was specified on the SUBMIT command:

```
//LIST     JOB     (accounting),'PROG. NAME',MSGLEVEL=1
//STEP1    EXEC    PGM=MYPROG
//SYSPRINT DD      &U1
//SYSUT1   DD      DSN=TESTDATA,DISP=(OLD,KEEP),VOL=SER=MYPACK
//SYSUT2   DD      &U2
```

COBI replaces the &U1 parameter on the SYSPRINT DD statement with SYSOUT=Z; it replaces the &U2 parameter on the SYSUT2 DD statement with the DSNAME, SPACE, DISP, and UNIT parameters described previously.

Procedure-Defined SYSOUT Data Sets: Since COBI does not have access to the procedure library, a different technique is used to provide DSNAME, SPACE, DISP, and UNIT parameters for a scannable procedure-defined SYSOUT data set. To save such a data set for scanning, the user specifies a parameter of the form nPmm in the SUBMIT command. For every nPmm parameter specified, COBI scans the job for the required (nth) procedure and supplies a symbolic parameter default of the form Pmm on the EXEC statement of the procedure. COBI assumes that the procedure will have a corresponding parameter of the form &Pmm.

However, this technique obviously requires some changes in the cataloged procedure itself. Therefore, before COBI-submitted jobs can execute cataloged procedures and save procedure created data sets for scanning, the cataloged procedures must be converted to a form suitable for use by COBI.

In effect, the conversion process requires that symbolic parameters of the form &Pmm be provided for every DD statement which contains a SYSOUT parameter for a specified class. In addition, a default for the symbolic parameters is provided to assign the SYSOUT data sets to the COBI output class. Therefore, if the user does not request scanning of a SYSOUT data set in a procedure, the data set is eventually routed to an OS/360 output class for processing. The conversion process also supplies a space allocation of SPACE=(TRK,(10,10),RLSE) for any SYSOUT data set which does not already have one. Finally, sequence numbers are added to each statement in the procedure.

Figure 11 shows an example of a procedure, before and after its conversion. Assume that only those data sets assigned to class A are to be converted. The DIBCONPR utility is provided to perform this conversion and is described in greater detail in "Preparing to Use COBI".

PROCA Before Conversion:

```
//MYSTEP    EXEC    PGM=MYPROG
//SYSUDUMP  DD      SYSOUT=A
//SYSPRINT  DD      SYSOUT=A
//PRINTOUT  DD      SYSOUT=A,SPACE=(CYL,(20,5))
//SYSPUNCH  DD      SYSOUT=B
```

PROCA After Conversion:

```
//PROCA     PROC    P1='SYSOUT=Z',  MYSTEP  SYSUDUMP    00000100
//                  P2='SYSOUT=Z',  MYSTEP  SYSPRINT    00000200
//                  P3='SYSOUT=Z'   MYSTEP  PRINTOUT    00000300
//MYSTEP    EXEC    PGM=MYPROG                          00000400
//SYSUDUMP  DD      &P1,SPACE=(TRK,(10,10),RLSE)        00000500
//SYSPRINT  DD      &P2,SPACE=(TRK,(10,10),RLSE)        00000600
//PRINTOUT  DD      &P3,SPACE=(CYL,(20,5))              00000700
//SYSPUNCH  DD      SYSOUT=B                            00000800
```

Figure 11. Sample cataloged procedure conversion for COBI

COBI Device Class

It is not possible for COBI to be aware of either the space available on direct access volumes or the current usage of the volumes. Therefore, to enable OS/360 to allocate space for scannable SYSOUT data sets, a nonspecific volume request is made in the DD statements for these data sets. To permit more than one volume to be used for scannable SYSOUT data sets, a class of devices is assigned for COBI use. This is achieved by specifying either a device type or a group (generic) name in the UNIT parameter of the DD statement.

Since the request for space is a nonspecific volume request and the data set is not temporary, OS/360 allocates space on a volume called a storage volume. OS/360 allocates space for the data set either on the storage volumes with the device type specified or on storage volumes having the specified generic name. The UNITNM initialization parameter may be used to specify a device type, a generic name, or a specific unit address. If the parameter is omitted, the default is a device type of 2314.

It is recommended that a unique generic name be used to identify those volumes on which scannable SYSOUT data sets may reside. This generic name is specified during OS/360 system generation (see "Building the System") and restricts the number of storage volumes that may be used in allocating space for scannable SYSOUT data sets. If the units given a generic name are not defined by any other group name (for example, SYSDA or SYSSQ), then only scannable data sets will reside on storage volumes having the COBI generic name.

SAMPLE COBI JOB AND ITS PROCESSING

This section illustrates the processing of a single COBI job, from its initial creation and submittal by the terminal user, through its processing by COBI and OS/360, as well as the handling of the output produced by the job. Assume that the job to be submitted executes the converted cataloged procedure PROCA, as shown previously in Figure 11, and that the user number of the submitting user is IBM408.

Creating and Submitting the Job

Before an OS/360 job may be submitted through COBI, the JCL, source statements, and/or data for that job must be present in a user's library. One way to do this is to create an OS/360 job at the terminal in the same way a CALL-OS source program file is created. For example, the user enters the JCL, and in this case, the data for the job, as numbered statements; he then saves this input in his library under the name PGMA, as follows:

```
001    //MYJOB          JOB     accounting-information
002    //STEP1          EXEC    PROCA (as shown in Figure 11)
003    //MYSTEP.SYSIN   DD      *
004    (data for MYPROG, the program executed by PROCA)
 .
 .
 .
nnn    /*
SAVE   PGMA
READY
```

He may then submit PGMA for OS/360 batch processing by issuing the following command:

```
SUBMIT  PGMA,(1P2)
```

The reply from COBI indicates the job number and the job name, as follows

```
#14 SUBMITTED AS IBM4080E
```

The notation 1P2 indicates that, in the first procedure in the job, the second SYSOUT data set (SYSPRINT) is to be saved for later scanning at the user's terminal. Since the user did not specify that the JCL be saved for scanning, the JCL is not saved unless a JCL error is detected.

For a detailed description of job submittal with COBI, see the publication CALL-OS Terminal Operations Manual.

COBI Processing After Submittal

After a job is submitted, COBI makes certain modifications to the job input before it is passed to OS/360 for execution. The CALL-OS line numbers are removed and certain parameters are added to the JOB and EXEC statements. In addition, a JCL comment is inserted after the JOB statement. Figure 12 shows the input for PGMA after the modifications have been made; the underlined portions indicate additional parameters.

```
//IBM4080E     JOB     accounting-information,MSGCLASS=Z
//* #14    additional-information
//STEP1        EXEC    PROCA,
//   P2='DSN=DIB.IBM408.#00014.P102,DISP=(NEW,KEEP),UNIT=CBSCAN'
//MYSTEP.SYSIN DD      *
 .
 .     (data for MYPROG)
 .
/*
```

Figure 12. PGMA input after COBI processing

The job name and a message class assignment (MSGCLASS parameter) are added to the JOB statement. The job name consists of the user number (IBM408) of the submitting user and a job-name identifier (E), obtained by hashing the COBI-assigned job number. (Note that if the ANYJNAME initialization parameter is specified, COBI will not assign a name to the job.) The message class is the COBI output class assigned in the CBCLASS initialization parameter; this same class must be used when converting cataloged procedures for use with COBI.

Following the JOB statement, COBI inserts a JCL comment statement beginning with //* #nnnnn, where #nnnnn is the job number (#14). The statement also contains additional information, such as the job number in binary, the name of the job, the CALL-OS coded date, and control information used by COBI.

For each procedure-defined SYSOUT data set designated as scannable, an override parameter is added to the EXEC statement. In this case, the overriding parameter is P2, because the second data set is to be scanned. The parameter value contains the data set name, a disposition, and a unit assignment. The data set name is the format described previously: index qualifiers of DIB, the user number (IBM408), and the job number (#00014), followed by a specific data set identifier (P102, the second data set of the first procedure). The disposition in all cases is DISP=(NEW,KEEP). The unit assignment is taken from the UNITNM initialization parameter; this parameter is used to indicate a nonspecific volume request for space allocation.

After these modifications are made to the program input, the job is written into a COBI input data set, which becomes the input data set for

the COBI reader program. This program attaches the OS/360 reader-interpreter, which reads the submitted jobs into an OS/360 job queue. (See the description of the COBI input data sets in "COBI Data Sets".)

OS/360 Processing

Symbolic parameter substitution made by the OS/360 reader interpreter results in the JCL and cataloged procedure statements shown in Figure 13. OS/360 device allocation assigns the SYSPRINT data set to one of the volumes mounted on the devices in the CBSCAN device class. Up to the point where an OS/360 output writer would normally process the job output, the execution and associated OS/360 processing is identical to that for any OS/360 job.

```
//             EXEC    PROCA
// P2='DSN=DIB.IBM408.#00014.P102,DISP=(NEW,KEEP),UNIT=CBSCAN'
//PROCA        PROC    P1='SYSOUT=Z',  MYSTEP  SYSUDUMP
//                     P2='SYSOUT=Z',  MYSTEP  SYSPRINT
//                     P3='SYSOUT=Z'   MYSTEP  PRINTOUT
XXMYSTEP       EXEC    PGM=MYPROG
XXSYSUDUMP     DD      &P1,SPACE=(TRK,(10,10),RLSE)
IEF653I  SUBSTITUTION JCL - SYSOUT=Z,SPACE=(TRK,(10,10),RLSE)
XXSYSPRINT     DD      &P2,SPACE=(TRK,(10,10),RLSE)
IEF653I  SUBSTITUTION JCL - DSN=DIB.IBM408.#00014.P102,DISP=(NEW,KEEP),
UNIT=CBSCAN,SPACE=(TRK,(10,10),RLSE)
XXPRINTOUT     DD      &P3,SPACE=(CYL,(20,5))
IEF653I  SUBSTITUTION JCL - SYSOUT=Z,SPACE=(CYL,(20,5))
```

Figure 13. PROCA after symbolic parameter substitution

Output Destinations and Final COBI Processing

The data set to be scanned (SYSPRINT) is written on a storage volume, as determined by OS/360 device allocation routines. Pointers to the other two data sets produced by the job (SYSUDUMP and PRINTOUT) are placed in the COBI class output queue, along with the JCL and resulting messages associated with the job. This output class is monitored by the COBI writer and information on execution status is retained. Finally, the output from the job is reset from the COBI class output queue to an output queue for processing by an OS/360 output writer.

COBI DATA SETS

The COBI facility requires data sets in addition to those required for execution of the CALL-OS system. These additional data sets, while not part of the data base, must be allocated to CALL-OS before COBI can be used and must be defined by DD statements when the system is initialized. The COBI data sets are used to retain information about the jobs submitted through COBI, to transmit the jobs to OS/360 batch processing, and to make information available to the terminal user. The data sets are:

- COBI index data set, which provides space for the index records assigned to submitted jobs
- COBI JCL data set, which provides space for storing the JCL and system messages for completed COBI-submitted jobs prior to printing at the user's terminal
- Two COBI input data sets, which are used alternately in the copying and formatting of the submitted jobs from the terminal users' library and as input data sets for the COBI reader program

The COBI index, JCL, and input data sets must be allocated and preformatted with the U#5INIT utility program before they can be used by CALL-OS system; these data sets may also be maintained with the U#5RINIT and U#5PURGE utility programs, used to reinitialize and clean up the data sets, respectively. In addition, the COBI index and JCL data sets may be expanded with the U#5CBXPN utility. (The U#5INIT utility program is described in "Initializing the COBI Data Sets"; the other utilities are described in "Maintaining the COBI Data Sets".)

In addition to the COBI index, JCL, and input data sets, additional data sets are required when COBI is initialized. Each volume which is to contain scannable data sets must be defined by a DD statement. The scannable data sets do not have to be initialized. Finally, the OS/360 job queue data set (SYS1.SYSJOBQE) must be defined. This data set is used to locate and access the COBI class output queue. The system job queue data set does not have to be initialized by the U#5INIT utility, nor does it have to be defined during reinitialization or purging of the COBI data sets.

The following text describes the use and characteristics of the COBI index, JCL, input, and scannable data sets and assumes that, when necessary, the data sets are in their preformatted state.

INDEX DATA SET

The COBI index data set provides space for the index records assigned to jobs submitted from a user's terminal. Every record in use except the first contains all the information pertaining to one job submitted to OS/360 batch processing; the first record contains data set identification information. Each record in the data set is 48 bytes long and the data set may contain up to a maximum of 32,000 records. The actual number of records in the data set is determined when the data set is initialized. This data set is defined with a CBNDX DD statement, where CBNDX is the ddname.

During CALL-OS system initialization, the records in the COBI index are scanned to determine which records are available for use. A bit string corresponding to the records is built in core: the bits for the records in use are set to one, the bits for the records available for use are set to zero.

When a new job is submitted, the bit string is scanned and the first available record in the COBI index is assigned to the new job. The bit corresponding to that record is then set to one. The number of the COBI index record assigned to the job is used as the COBI-assigned job number. For example, the job assigned to the fourth record is given job number #4; the job assigned to the one-hundredth record is given job number #100. Because the first record is always used for data set identification, it is impossible for a job to be assigned job number #1.

After a job has been submitted to OS/360 for processing, its index record contains most of the information about the job. For example, the record contains the user number of the user who submitted the job, the job name, and the location of the job in a COBI input data set, in case the user decides to cancel the job. During processing, the index record is used to maintain status information regarding the execution of the job.

When execution of a job is completed, additional information is put into the COBI index record for the job. If the user specified that an output data set be saved for scanning at his terminal, the relative volume identification of the device containing the data set is added to the record. In addition, system and/or user completion codes are

inserted in the record. The relative volume identification and the completion codes are written by the COBI writer.

The index record for a job is deleted when the job is scratched or cancelled. Under certain circumstances, the scratching of a data set may also cause the COBI index record for the job to be deleted. When an index record is deleted, the bit corresponding to it in the bit string is set to zero, thus making an index record available for use by a new job.

The original size of the COBI index is estimated according to the maximum number of jobs that may be processed at any one time. After a COBI system has been in operation for a while, the COBI index may become full during a session. In this case, no more jobs may be submitted until either some index records become free or the index itself is expanded. The utility program U#5CBXPN is used to increase the size of the COBI index.

Two other utilities are available for maintaining the COBI index: U#5RINIT, which reinitializes the data set, and U#5PURGE, which cleans up the data set. All three utilities are described in "Maintaining the COBI Data Sets".

JCL DATA SET

The COBI JCL data set contains the JCL for submitted jobs under two conditions: either the user specified that the JCL for the job was to be saved for scanning when he submitted the job, or a JCL error was detected when the job was processed under OS/360 but the user did not specify that the JCL be saved. In either case, the COBI writer copies the JCL into the COBI JCL data set. The actual module used depends on whether CALL-OS is in operation or not: if CALL-OS is in use, the module is M#JCL; if CALL-OS is not in use, the module is DIBWTR.

Space in the JCL data set is allocated in record sets: a JCL record set contains from one through four 3440-byte records. Every set except the first is used for JCL storage; the first set is reserved for the volume identification table. This table contains the volume serial numbers of all volumes used to store output data sets saved for scanning at the terminal. The rest of the JCL record sets are allocated as needed for the storage of the JCL for a job. The total number of records in the JCL data set and the number of records in each record set is determined when the data set is initialized. The JCL data set is defined with a CBJCL DD statement, where CBJCL is the ddname.

During CALL-OS system initialization, the volume identification table is built from the DD statements supplied for volumes containing scannable data sets. In addition, the JCL record sets are scanned to determine which are available for use. A bit string corresponding to the sets is built in core: the bits for the sets in use are set to one, the bits for the sets available for use are zero.

After a submitted job has been processed by OS/360, the COBI writer processes the job's output in the COBI class output queue. If the JCL for the job is to be saved, the COBI writer allocates the first available JCL record set to the job, enters a pointer to the JCL in the index record for the job, and copies the JCL. An identification header is written at the beginning of each 3440-byte record used for JCL storage.

As the JCL is copied, it is compressed to eliminate blanks and reformatted. The new format of a line of JCL in the JCL data set is identical to the format of a CALL-OS source statement in a program file. (The first byte contains the number of characters in the line and the

last byte contains X'15', which indicates the end of the line.) JCL lines are not split across records. The minimum amount of space used for the JCL for a job is one set; the maximum is three sets. If the JCL for a job exceeds three sets, the rest of the JCL is not saved for scanning; however, the complete JCL is printed on the high-speed printer with the rest of the output for the job. A bit in the identification header of the first record in the first set indicates that the JCL has been truncated.

The original size of the COBI JCL data set is estimated according to the size of the COBI index and the number of records in a record set. After a COBI system has been in operation for a while, it may become necessary to adjust the size of the JCL data set. The utility program U#5CBXPN is used to expand or contract the JCL data set.

Two other utilities are available for maintaining the COBI JCL data set: U#5RINIT, which reinitializes the data set, and U#5PURGE, which cleans up the data set. All three utilities are described in "Maintaining the COBI Data Sets".

INPUT DATA SETS

COBI uses two input data sets so that, ideally, one is attached to COBI to receive jobs from terminal users while the other is being read into OS/360 batch processing by the COBI reader program. The data sets are defined with CBSYSINA and CBSYSINB DD statements, where CBSYSINA and CBSYSINB are ddnames; the COBI reader program is DIBRDR and is executed by either of two cataloged procedures, DIBRDRA and DIBRDRB. One cataloged procedure is associated with each data set and reads only that data set. For example, DIBRDRB reads only the CBSYSINB data set. The format of the two data sets and the space allocated to them must be identical. The amount of space is determined when the data sets are initialized.

Each COBI input data set consists of two parts. The first part contains the jobs submitted from the terminal; the second part contains the write-to-operator messages associated with these jobs. The first record in the data set contains a pointer to the beginning of the part which contains the write-to-operator messages.

Initially, the first record in each data set has the characters EOF starting in column 31; this indicates that the data set is available for receiving COBI jobs. Thereafter, the reader program writes the EOF record at the beginning of the data set when all the jobs have been read into OS/360 batch processing. This ensures that the jobs will not be processed twice.

When CALL-OS with COBI has been initialized, one input data set is attached to COBI and the other is in available status. When a job is submitted from a terminal, the job is placed in the data set attached to COBI. At certain points in the processing of submitted jobs, COBI attempts to switch the two data sets, thereby making the data set currently attached to COBI available to the COBI reader. The switching of data sets occurs under the following conditions:

- When the current input data set is full
- When a certain number of jobs have been read into the data set (this number is specified when the system is initialized)
- When a certain number of minutes have elapsed since the data set was attached to COBI (this number is specified when the system is initialized) and there is at least one job in the data set

- When the operator requests that any jobs ready for OS/360 processing be made available to a COBI reader

If any one of the preceding conditions is met, COBI checks the status of the alternate input data set. If the data set has the EOF record as the first record, the data set is ready to receive new jobs. If the EOF record is not present and the current input data set is not full, COBI continues to write jobs on the current data set until the alternate data set is available. When the alternate data set becomes available, the current input data set is given to a COBI reader for processing and the alternate data set is attached to COBI to receive new jobs. If both data sets are full or unavailable, no more jobs may be submitted until an input data set becomes available.

The initialization parameters are described in the section "Initializing the System"; the operator commands are described in the publication CALL-OS Operator's Manual.

SCANNABLE DATA SETS

Scannable data sets contain information which may be printed (or scanned) on a user terminal. The user may scan three types of data sets:

- Output (SYSOUT) data sets associated with one or more of the jobs submitted by him; he must request that the data set be saved for scanning when he submits the job.
- Output data sets which have his user number as one of the index qualifiers in a data set name. Usually, the data set name was not created by COBI.
- System data sets whose high level index qualifier was designated as scannable during system initialization of CALL-OS.

The volumes which are to contain or do contain such data sets are defined during system initialization. The DD statements for these volumes have a name of the form SCANxx, where xx is an identifier which serves to make the name unique.

The volume identification table in the JCL data set contains the volume serial numbers of all scannable data set volumes defined during the current or any previous session of CALL-OS with COBI. However, not all of these volumes need be mounted during a session and any session may define new volumes by adding volume serial numbers during system initialization. The only way the volume serial numbers are deleted from the volume identification table is by reinitializing the JCL data set.

Scannable Output Data Sets

During OS/360 system generation, it is recommended that the installation assign a specific class of devices to be allocated for scannable output from COBI jobs. During CALL-OS system initialization, specific volumes are mounted, identified by volume serial number, and assigned to the COBI device class. When a user submits a job, he designates which SYSOUT data sets he wishes to scan by specifying the appropriate data set identifiers (of the form Unnn or nPmm); finally, during normal OS/360 device allocation for the job, the data set is allocated on one of the storage volumes assigned to the COBI device class. A SCANxx DD statement must be included in the CALL-OS startup deck for each volume mounted for the COBI device class and which contains scannable data sets. The blocksize for these data sets must

not exceed 127 physical blocks per track; the data sets must reside on 2314 or 2319 disk storage.

When a data set is specified as scannable, COBI assigns a retention period of seven days; at the end of this time, if the user has not scanned the data set, the operator may scratch the data set. When a user scans a data set, he specifies whether the data set is to be kept or scratched. If it is to be kept, the retention period is reset for another seven days.

Scannable System Data Sets

With the SCANDS initialization parameter, the installation may specify two high level index qualifiers, thereby making two groups of system data sets available for scanning at a user terminal. For example, if SYS1 is specified, the terminal user may scan SYS1.PROCLIB or SYS1.MACLIB or any other data set with SYS1 as the high level index qualifier. The volumes on which these data sets reside must also be defined by SCANxx DD statements during CALL-OS initialization. The COBI command facilities do not permit the scratching or modification of these system data sets.

PREPARING TO USE COBI

Before COBI may be used, certain preparatory steps must be taken to provide the necessary operating environment. These steps are:

- Modify the IEEVLNKT control section by adding the names of the COBI reader and writer programs
- Convert the installation cataloged procedures with the DIBCONPR utility to enable the procedures to be used by COBI jobs
- Add cataloged procedures for the COBI reader and writer programs to SYS1.PROCLIB
- Initialize the COBI index, JCL data set, and the two input data sets
- Link edit the COBI reader and writer programs into SYS1.LINKLIB

The last two steps may be performed together as part of an IBM supplied cataloged procedure (COBIBLD). Before COBIBLD may be used, the first step of the CALL-OS system build process must have been completed; this step puts the CALL-OS procedures into SYS1.PROCLIB. The system build process is described in the section "Building the System".

MODIFYING THE IEEVLNKT CONTROL SECTION

The OS/360 control section IEEVLNKT contains the names of all load modules which may be executed by an OS/360 operator start command. The operator executes these load modules from the OS/360 system operator's console by issuing a start command which refers to a cataloged procedure. The procedure then initiates execution of the appropriate load module. When COBI is used, the operator must issue start commands which refer to COBI cataloged procedures. These procedures execute the COBI reader and writer modules, DIBRDR and DIBWTR, respectively.

Therefore, before COBI can be used, the names DIBRDR and DIBWTR must be added to the system. An IBM-supplied module named DIBNAMES contains the COBI names in load module format. (This load module is copied from OSRTS.EXEC.JOBLIB into qualifier.JOBLIB during Step I of the CALL-OS system build process; see "Building the System.) The linkage editor is

used to combine these names with the names in the IEEVLNKT control section and to replace references to IEEVLNKT with references to DIBNAMES. The actual linkage editor control statements used depend on whether MFT or MVT is used.

With an MFT System

When an MFT system is used, the IEEVLNKT, IEEVACTL, and IEEVRCTL load modules must all be modified to include the COBI names. The JCL required to execute the linkage editor and the linkage editor control statements necessary to modify the modules are as follows:

```
//LKED          EXEC    PGM=IEWL,PARM='NCAL,LIST,XREF,LET,RENT,REFR'
//SYSUT1        DD      UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSLIB        DD      DSN=qualifier.JOBLIB,DISP=SHR
//SYSLMOD       DD      DSN=SYS1.LINKLIB,DISP=OLD
//SYSPRINT      DD      SYSOUT=A
//SYSLIN        DD      *
  INCLUDE  SYSLIB(DIBNAMES)
  INCLUDE  SYSLMOD(IEEVLNKT)
  ENTRY  DIBNAMES
  NAME  IEEVLNKT(R)
  REPLACE  IEEVLNKT(DIBNAMES)
  INCLUDE  SYSLMOD(IEEVRJCL)
  INCLUDE  SYSLIB(DIBNAMES)
  INCLUDE  SYSLMOD(IEEVRJCL)
  ENTRY IEEVRJCL
  ALIAS IEEPSN
  NAME  IEEVRJCL(R)
```

where

qualifier — is the high level index qualifier assigned to CALL-OS data sets during system build.

With an MVT System

When an MVT system is used, only the IEEVRJCL load module must be modified to include the COBI names. The JCL required to execute the linkage editor and the linkage editor control statements necessary to modify the module are as follows:

```
//LKED          EXEC    PGM=IEWL,PARM='NCAL,LIST,XREF,LET,RENT,REFR'
//SYSUT1        DD      UNIT=SYSDA,SPACE=(TRK,(100,10))
//SYSLIB        DD      DSN=qualifier.JOBLIB,DISP=SHR
//SYSLMOD       DD      DSN=SYS1.LINKLIB,DISP=OLD
//SYSPRINT      DD      SYSOUT=A
//SYSLIN        DD      *
  REPLACE  IEEVLNKT(DIBNAMES)
  INCLUDE  SYSLMOD(IEEVRJCL)
  INCLUDE  SYSLIB(DIBNAMES)
  INCLUDE  SYSLMOD(IEEVRJCL)
  ENTRY  IEEVRJCL
  ALIAS  IEEPSN
  NAME  IEEVRJCL(R)
```

where

qualifier — is the high level index qualifier assigned to CALL-OS data sets during system build.

CONVERTING CATALOGED PROCEDURES

If SYSOUT data sets produced by a cataloged procedure are to be scanned at the terminal, the procedure should be converted to a form suitable for use by COBI with the DIBCONPR utility program. (If a procedure is not converted, a &Unnn parameter must be used to override the procedure JCL.) It is recommended that the converted procedures be placed in a procedure library unique to COBI and available for use only by COBI users. If the converted procedures are returned to the procedure library they were in before conversion, the names of the converted procedures must be altered.

The SYSPRINT data set in the GO step of certain cataloged procedures defaults to undefined record format. Since undefined records may not be used with the COBI SCAN option, the default must be overridden if the data set is to be scanned. Overriding DCB information may be added either before the cataloged procedures are converted or at the time the procedure is executed. The DIBCONPR utility cannot be used to add this information to a DD statement.

The rest of this subsection describes the JCL required to execute the utility, the conversion process, an example of procedure conversion, and the use of the COBI procedure library.

JCL Requirements for DIBCONPR

When executing the DIBCONPR utility, the user may specify the class of SYSOUT data sets to be converted, the class to which the converted data sets are to be assigned, and the space allocation to be given to any data set which does not already have one specified in the procedure. (The space allocation is necessary because if any of the data sets are to be scanned, they become sequential disk data sets; as such, they do not receive the SYSOUT default space allocation assigned by the reader.) All three parameters are optional and appropriate defaults are supplied if the parameter is omitted.

In addition, DD statements must be supplied which define the procedure library which contains the procedures to be converted, the sequential data set in which the converted procedures are to be written, a listing data set, and a control card data set.

The complete JCL required to execute DIBCONPR is as follows:

```
//ANYNAME       JOB     ---
//JOBLIB        DD      DSN=qualifier1.JOBLIB,DISP=SHR
//STEPNAME      EXEC    PGM=DIBCONPR,PARM=('OSCLASS=a','CBCLASS=z',
//                      'SPACE=parms'),REGION=60K
//SYSLIB        DD      DSN=qualifier2.PROCLIB,DISP=SHR
//SYSPRINT      DD      SYSOUT=A
//SYSPUNCH      DD      data-set-definition
//SYSIN         DD      *
control-cards
/*
```

where

qualifier1 — is the high level index qualifier assigned to CALL-OS data sets during system build

a — specifies the class of SYSOUT data sets to be converted and must be a valid SYSOUT class. The default is class A. This class is the OS/360 output class.

z — specifies the class to which the appropriate SYSOUT data sets are to be converted and must be a valid SYSOUT class. The default is class Z. This class is the COBI output class; the CBCLASS system initialization parameter must match the CBCLASS class specified for the conversion process.

parms — is any valid space allocation of 50 characters or less. All data sets in the converted class are assigned this space allocation unless a SPACE parameter is already present in the unconverted procedure. In most cases, cylinder allocation rather than track allocation should be employed, because of its superior performance characteristics. The default is (CYL,(1,1)). (DIBCONPR performs only minimal syntax checking on this parameter.)

qualifier2 — is the high level index qualifier of the procedure library which contains the procedures to be converted. In most cases, this will be SYS1. Note that additional procedure libraries may be concatenated to the library defined on the SYSLIB DD statement.

data-set-definition — defines a sequential data set which is to contain the converted procedures. For example, SYSOUT=B is a valid definition, as well as any tape or sequential disk data set definition.

control-cards — specify which procedures in the SYSLIB data set(s) are to be converted; each control card contains the member name of one procedure to be converted in columns 1 through 8.

Table 1 shows the defaults for those parameters which may be omitted.

Based on the parameters, DIBCONPR converts the procedures as described in the following text.

Table 1. Parameter defaults for the DIBCONPR utility

Parameter	Use	Default
CBCLASS	Specifies the class to which SYSOUT data sets are to be assigned during conversion	Class Z
OSCLASS	Specifies the class of SYSOUT data sets to be converted	Class A
SPACE	Specifies the space allocation to be used for converted data sets which do not already have one	(CYL,(1,1))

Conversion Process

For each SYSOUT data set in the class to be converted, DIBCONPR replaces the SYSOUT specification on the appropriate DD statement with the following:

```
&Pn[,SPACE=parms]
```

where

n	is the number of the data set within the procedure and may range from 1 through 15; if a procedure has more than 15 data sets to be converted, the rest are not converted.
parms	is either the space allocation specified in the SPACE parameter of the utility EXEC statement or the default if the SPACE parameter was omitted; if a data set to be converted already has a SPACE parameter, the parameter is not changed.

For each converted DD statement, the utility adds to the PROC statement a symbolic parameter with a default value; a comment field is added after the parameter. The parameter and the comment field have the following format:

```
Pn='SYSOUT=z'[,]    step-name    ddname
```

where

n	is the number of the data set within the procedure, as defined previously
z	is either the class specified in the CBCLASS parameter on the utility EXEC statement or a default of Z if the parameter was omitted; the comma follows the parameter if more symbolic parameters are required.
step-name	is the name of the step within the procedure to which the parameter applies
ddname	is the name of the DD statement which was converted

As a procedure is converted, the statements in it are given sequence numbers, starting with 100 and incremented by 100. (If other sequence numbers are desired, they can be specified as the procedure is written into the COBI procedure library by adding additional control statements.)

When a procedure is converted, an IEBUPDTE utility control statement is written in the SYSPUNCH data set; this control statement has the following format:

```
ADD     NAME=member-name
```

where

member-name	is the name of the converted procedure as specified in the control card input for DIBCONPR

The converted procedure is then written in the SYSPUNCH data set and a listing of the procedure is printed on the SYSPRINT data set. The entire process is repeated for each member named in the SYSIN data set.

Conversion Example

The following is an example of the JCL required to execute the DIBCONPR utility program:

```
//CONVERT     JOB     1,'PGMR, C',MSGLEVEL=1
//JOBLIB      DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//CON         EXEC    PGM=DIBCONPR,PARM='CBCLASS=J,SPACE=(CYL,(2,2))'
//                    REGION=60K
//SYSLIB      DD      DSNAME=SYS1.PROCLIB,DISP=SHR
//              DD    DSNAME=USER.PROCLIB,DISP=SHR
//SYSPRINT    DD      SYSOUT=A
//SYSPUNCH    DD      SYSOUT=B
//SYSIN       DD      *
SAMPLE
ASMFC
ASMFCL
ASMFCLG
 .
 .
 .
/*
```

Since the OSCLASS parameter was not specified on the utility EXEC statement, all SYSOUT data sets in output class A are to be converted.

Figure 14 shows the procedure SAMPLE before its conversion; Figure 15 shows the same procedure after its conversion. Note that the space allocation for PRINT1 was part of the unconverted procedure and was not altered by DIBCONPR.

```
//P1          PROC    LIB=TEST
//STEP1       EXEC    PGM=PROG1
//SYSLIB      DD      DSN=&LIB..JOBLIB,DISP=SHR
//SYSPRINT    DD      SYSOUT=A
//SYSPUNCH    DD      SYSOUT=B
//PRINT1      DD      SYSOUT=A,SPACE=(CYL,(8,4))
//STEP2       EXEC    PGM=PROG2
//SYSPRINT    DD      SYSOUT=A
```

Figure 14. Sample procedure before conversion by DIBCONPR

```
./          ADD NAME=SAMPLE
//P1            PROC  LIB=TEST,                                    00000100
//              P1='SYSOUT=J',          STEP1   SYSPRINT           00000200
//              P2='SYSOUT=J',          STEP1   PRINT1             00000300
//              P3='SYSOUT=J'           STEP2   SYSPRINT           00000400
//STEP1         EXEC PGM=PROG1                                     00000500
//SYSLIB        DD   DSN=&LIB..JOBLIB,DISP=SHR                     00000600
//SYSPRINT      DD   &P1,SPACE=(CYL,(2,2))                         00000700
//SYSPUNCH      DD   SYSOUT=B                                      00000800
//PRINT1        DD   &P2,SPACE=(CYL,(8,4))                         00000900
//STEP2         EXEC PGM=PROG2                                     00001000
//SYSPRINT      DD   &P3,SPACE=(CYL,(2,2))                         00001100
```

Figure 15. Sample procedure after conversion by DIBCONPR

Using the COBI Procedure Library

The converted procedures should be placed in a procedure library other than the one they were in before conversion; this library should be reserved for COBI use only. When the COBI procedure library is created initially, the entire SYSPUNCH data set may be used as the input for the IEBUPDTE utility program.

This utility may also be used to maintain the COBI procedure library; however, in this case, the ADD control statement for each procedure which already exists in the library must be replaced by a REPL control statement. For example, if an installation changes the assignment of its COBI output class, then the cataloged procedures must be reconverted. To reconvert procedures, DIBCONPR utility must be executed against the original unconverted procedures; procedures which are already converted may not be reprocessed by DIBCONPR.

The following example shows the JCL necessary to create a COBI procedure library:

```
//COBIPROC      JOB     ---
//UPDTE         EXEC    PGM=IEBUPDTE,PARM=NEW
//SYSUT2        DD      DSN=OSRTS.COBIPROC,UNIT=2314,VOL=SER=COBI01,
//                      SPACE=(CYL,(2,1,8)),DISP=(NEW,CATLG),
//                      DCB=SYS1.PROCLIB
//SYSPRINT      DD      SYSOUT=A
//SYSIN         DD      DATA
(SYSPUNCH output from the execution of the DIBCONPR utility)
/*
```

For detailed information on the use of the IEBUPDTE utility, see the publication IBM System/360 Operating System: Utilities.

SUPPLYING COBI READER AND WRITER PROCEDURES

The installation must provide three procedures to be used with COBI: two reader procedures and one writer procedure. The requirements for the procedures as well as how to add these procedures to the system are described in the following text.

COBI Reader (DIBRDR) Procedures

Two reader procedures, named DIBRDRA and DIBRDRB, must be supplied for use by the COBI reader program. Since this program attaches the OS/360 reader/interpreter IEFIRC, it is possible to modify an existing OS/360 reader procedure for use with COBI. Whether this is done or a new procedure is written, the following requirements must be met for both COBI reader procedures:

- PGM=DIBRDR must be specified on the EXEC statement
- The parameter field should be coded as shown with one exception: the last character (Z in the example) is the default output class for jobs read by the reader; this must be identical to the COBI output class specified in the CBCLASS initialization parameter.
- The recommended region size is 68K
- The IEFRDER DD statement specifies the data set name of the appropriate input data set; this name is of the form

  DSN=qualifier.CBSYSINx

  where

  qualifier is the high level index qualifier assigned to CALL-OS data sets during system build.

  x specifies the input data set, and is A in the DIBRDRA procedure, B in the DIBRDRB procedure.

The disposition on this DD statement is DISP=SHR. No other information need be supplied because the input data sets are initialized and cataloged before COBI is used.

- If the procedures were converted for COBI use, the IEFPDSI DD statement must contain the name of the procedure library in which the converted procedures were placed. If necessary, the SYS1.PROCLIB procedure library may be concatenated to the COBI procedure library. (This is desirable when not all the procedures were converted for COBI use.)

- The DIBRDRA and DIBRDRB procedures must be identical, with the exception of the data set name on the IEFRDER DD statement.

- The IEFDATA DD statement should contain a blocksize specification of at least 400 for efficient execution; however, if a program cannot handle input with this blocksize, a DCB parameter with the required blocksize must be added to the DD * statement for the program.

The following is an example of the DIBRDRA procedure:

```
//STEPNAME     EXEC    PGM=DIBRDR,
//                     PARM='8010300100102490503OSYSDA   000011Z',
//                     REGION=68K
//IEFRDER      DD      DSN=OSRTS.CBSYSINA,DISP=SHR
//IEFPDSI      DD      DSN=OSRTS.COBIPROC,DISP=SHR
//             DD      DSN=SYS1.PROCLIB,DISP=SHR
//IEFDATA      DD      UNIT=SYSDA,SPACE=(80,(500,500),RLSE,CONTIG),
//                     DCB=(BUFNO=2,LRECL=80,BLKSIZE=400,RECFM=FB)
```

In this example, a separate procedure library was used for the converted procedures, and SYS1.PROCLIB is concatenated to it. The corresponding DIBRDRB procedure would be identical with one exception: the data set name DSN=OSRTS.CBSYSINB must be used on the IEFRDER DD statement.

The PARM field of DIBRDRA and DIBRDRB procedures should contain a track allocation size compatible with the CALL-OS startup deck. (See ALOCTYPE and DSPACE under "Additional COBI Options".) This ensures that if a SYSOUT data set is retained for scanning on one execution, but not another, the same amount of space is allocated. (That is, the user-specified allocation for a data set not retained for scanning should be such that it has the same effect as the COBI-generated SPACE parameter for a retained data set.) For example, if the ALOCTYPE and DSPACE parameters are not specified, or if ALOCTYPE=CYL and DSPACE is not used, then the system generates a SPACE parameter CYL,(001,001). The DIBRDRA and DIBRDRB procedures should contain 020 020.

A detailed description of the PARM field and the IEFDATA DD statement for the OS/360 reader interpreter procedure is found in the publication IBM System/360 Operating System: System Programmer's Guide.

COBI Writer (DIBWTR) Procedure

One procedure named DIBWTR must be supplied for the COBI writer program, also called DIBWTR; this program is used when CALL-OS is not being used but COBI jobs remain to be processed. DIBWTR is not to be confused with an OS/360 output writer. DIBWTR monitors the CBCLASS output queue for output from COBI-submitted jobs. If either the JCL is to be saved for scanning or a JCL error was detected, DIBWTR copies the JCL into the JCL data set; the COBI job output is then reset to the OSCLASS output queue for processing by an OS/360 output writer. Whether the JCL is saved or not, DIBWTR also stores status information about the job in the COBI index record for the job. DIBWTR processes one job each

time it receives control; the time interval which elapses before it regains control is determined by the installation.

The EXEC statement in the DIBWTR procedure must contain the following parameters:

- CBCLASS, which specifies the COBI output class; this class is processed by DIBWTR
- OSCLASS, which specifies the OS/360 output class to which the COBI class is reset after processing
- ITIME, which specifies the number of seconds to elapse before DIBWTR processes another job

The procedure must be named DIBWTR and have DIBWTR as the name on the PROC card. This is necessary so that when a STOP command is issued by the OS operator, the correct CSCB can be located, and the system posted for termination.

The procedure must contain DD statements for the COBI index and the COBI JCL data set. In addition, the OS/360 system job queue must be defined because DIBWTR uses this data set to locate and access the CBCLASS output queue for COBI jobs. The following is an example of the JCL required in a DIBWTR procedure:

```
//STEPNAME     EXEC    PGM=DIBWTR,PARM=('CBCLASS=z','OSCLASS=a',
//                     'ITIME=nnnn'),REGION=28K
//CBNDX        DD      DSN=qualifier.CBNDX,DISP=SHR
//CBJCL        DD      DSN=qualifier.CBJCL,DISP=SHR
//CBJOBQ       DD      DSN=SYS1.SYSJOBQE,DISP=SHR
```

where

z	is the COBI output class; it must be identical to the CBCLASS specification given in the startup deck and used for DIBCONPR when converting cataloged procedures
a	is the OS/360 output class to which the COBI class is reset after processing; it must not be the same as the class specified for the CBCLASS parameter
nnnn	is the time, in seconds, to elapse between the time DIBWTR finishes one job and receives control to process the next job; the time interval must be within the range from 1 through 9999
qualifier	is the high level index qualifier assigned to CALL-OS data sets during system build.

Adding the COBI Procedures to the System

When the three procedures have been prepared as described in the preceding text, they must be added to SYS1.PROCLIB with the IEBUPDTE utility program. The following example shows the use of this utility to add the DIBRDRA, DIBRDRB, and DIBWTR procedures to the system:

```
//PROCUPDT     JOB      ---
//UPDTE        EXEC     PGM=IEBUPDTE,PARM=MOD
//SYSUT1       DD       DSN=SYS1.PROCLIB,DISP=OLD
//SYSUT2       DD       DSN=SYS1.PROCLIB,DISP=OLD
//SYSPRINT     DD       SYSOUT=A
//SYSIN        DD       DATA
./     ADD      NAME=DIBRDRA
   (JCL Statements for DIBRDRA)
./     ADD      NAME=DIBRDRB
   (JCL Statements for DIBRDRB)
./     ADD      NAME=DIBWTR
   (JCL Statements for DIBWTR)
/*
```

A detailed description of the IEBUPDTE utility is found in the publication IBM System/360 Operating System: Utilities.

INITIALIZING THE COBI DATA SETS

Before COBI can be used, the COBI index, COBI JCL data set, and the two COBI input data sets must be initialized by utility U#5INIT. The utility may be executed as part of a cataloged procedure supplied by IBM or it may be executed separately. In either case, the data set requirements and the initialization process are identical. The rest of this section describes the initialization process for each data set, the use of the cataloged procedure, and the use of the utility as a separate program. For a description of the data sets and their use, refer to "COBI Data Sets" presented earlier in this section.

Initialization Process

The four COBI data sets which must be initialized must be processed by U#5INIT at the same time. The utility formats the records in each data set and, when necessary, places required information in the records. The process for each data set is described in more detail in the following text.

COBI Index Data Set: The user specifies the number of 48-byte records to be allocated. The utility initializes and writes the specified number of records into the data set. When U#5INIT has finished processing, the first record of the COBI index identifies the data set and contains the following information:

- The total number of records in the data set
- The total number of records in the JCL data set
- The number of records in each JCL record set
- The number of cylinders in each COBI input data set.

The second and all subsequent records in the COBI index are identical in format: the first byte of each record contains zeros, indicating that the record is unused; the second byte contains the record number relative to the beginning of the data set (for example, 2 for the second record, etc.); the rest of each record contains zeros.

COBI JCL Data Set: The user specifies the total number of 3440-byte records to be allocated as well as the number of records in each JCL record set. The utility stores this information in the first record of the COBI index and writes the specified number of JCL record sets into

the data set. When U#5INIT has finished processing, the first record of the first record set contains an identifier and skeleton format for the volume identification table; the rest of the first set is not used. The remaining record sets in the data set contain zeros.

COBI Input Data Sets: The user specifies the number of cylinders to be allocated to each input data set. The utility stores this information in the first record of the COBI index and writes the first and only record into each input data set. This record begins with the identification //* and indicates the data set (SYSINA or SYSINB). Finally, the record contains the characters EOF starting in column 31. The rest of the record contains zeros; the rest of the data set is not altered.

Using the Cataloged Procedure

The cataloged procedure COBIBLD may be used to initialize the COBI data sets and to link-edit the COBI reader and writer modules into SYS1.LINKLIB. The first step of this procedure executes the U#5INIT utility. This utility allocates and preformats the COBI data sets according to information supplied either by the user or as defaults. User-supplied information is specified in the parameter field of the EXEC statement. The JCL required to use the cataloged procedure is as follows:

```
//INIT     JOB     ---
//JOBLIB   DD      DSN=OSRTS.JOBLIB,DISP=SHR
//DSINIT   EXEC    COBIBLD,CBINDEX=aaa,CBJCLRD=bbb,CBRDSET=mm,
//                 SYSIN=nn,QA=qualifier,VOLID=volid
```

where

aaa — is the number of 48-byte records to be allocated in the COBI index. This number is installation dependent and is based on the number of submitted jobs that might be present in the system at any one time. The number specified must be within the range from 2 through 32,000. If CBINDEX is omitted, the default is 150 records.

bbb — is the number of 3440-byte records to be allocated in the JCL data set. This number must be at least twice the number (mm) of records specified for a record set. In most cases, this number should be equal to the number of index records (aaa) times the number of records per set (mm). If CBJCLRD is omitted, the default is 300 records.

mm — is the number of 3440-byte records in each JCL record set and may be from one through four. If CBRDSET is omitted, the default is two records per set.

nn — is the number of cylinders to be allocated for the COBI input data sets. If SYSIN is omitted, the default is ten cylinders for each data set.

qualifier — is the high level index qualifier (QA) assigned to CALL-OS data sets during system build. If QA is omitted, the default is OSRTS.

volid — is the volume serial number of the volume which is to contain the COBI data sets. No default is provided, therefore VOLID must always be specified. Note that this causes the assignment of all four data sets to the same volume. By overriding one or more DD statements in the

cataloged procedure, the associated data sets may be assigned to different volumes.

Table 2 shows the defaults for those parameters which may be omitted. Figure 16 shows the statements in the first step of the COBIBLD cataloged procedure.

The user may override certain specifications in the DD statements of the procedure. The usual rules for overriding must be followed. For example, the overriding DD statements must be in the same sequence as the DD statements in the procedure. The DD statements which can be overridden are:

CBNDX Defines the COBI index data set

CBJCL Defines the COBI JCL data set

CBSYSINA Defines the SYSINA input data set

CBSYSINB Defines the SYSINB input data set

The VOL=SER parameter may be overridden on all four DD statements; the DCB parameter may be added to the CBSYSINA and CBSYSINB DD statements to provide a blocksize specification other than 3200 bytes for the input data sets. None of the other information on the DD statements should be overridden.

Table 2. Parameter defaults for the COBIBLD procedure

Parameter	Use	Default
CBINDEX	Specifies the number of 48-byte records to be allocated to the COBI index data set	150 records
CBJCLRD	Specifies the number of 3440-byte records to be allocated to the COBI JCL data set	300 records
CBRDSET	Specifies the number of 3440-byte records in each JCL record set	Two records per set
QA	Specifies the high level index qualifier assigned to CALL-OS data sets during system build	OSRTS
SYSIN	Specifies the number of cylinders to be allocated to each of the COBI input data sets	Ten cylinders for each data set

```
//COBIBLD     PROC   CBINDEX=150,CBJCLRD=300,SYSIN=10,QA=OSRTS,
//                   CBRDSET=2
//CB1         EXEC   PGM=U#5INIT,PARM=('CBINDEX=&CBINDEX',
//                   'CBJCLRD=&CBJCLRD','CBRDSET=&CBRDSET')
//CBNDX       DD     DSN=&QA..CBNDX,SPACE=(48,(&CBINDEX),,CONTIG),
//                   VOL=SER=&VOLID,DISP=(NEW,CATLG,DELETE),
//                   UNIT=2314
//CBJCL       DD     DSN=&QA..CBJCL,VOL=SER=&VOLID,UNIT=2314,
//                   SPACE=(3440,(&CBJCLRD),RLSE,CONTIG,ROUND),
//                   DISP=(NEW,CATLG,DELETE)
//CBSYSINA    DD     DSN=&QA..CBSYSINA,SPACE=(CYL,(&SYSIN),,CONTIG),
//                   VOL=SER=&VOLID,DISP=(NEW,CATLG,DELETE),
//                   UNIT=2314
//CBSYSINB    DD     DSN=&QA..CBSYSINB,SPACE=(CYL,(&SYSIN),,CONTIG),
//                   VOL=SER=&VOLID,DISP=(NEW,CATLG,DELETE),
//                   UNIT=2314
//SYSPRINT    DD     SYSOUT=A
```

Figure 16. JCL statements in the first step of the COBIBLD procedure

Overriding the Volume Serial Number: Since the VOLID parameter causes all four data sets to be allocated on the same volume, the user may want to override this allocation. The following example shows the use of the COBIBLD procedure while overriding the volume specification:

```
//DSINIT      JOB    ---
//JOBLIB      DD     DSN=OSRTS.JOBLIB,DISP=SHR
//INIT        EXEC   COBIBLD,VOLID=COBI01
//CBSYSINA    DD     VOL=SER=COBI02
//CBSYSINB    DD     VOL=SER=COBI02
```

Since the CBNDX and CBJCL DD statements were not overridden, the COBI index and the COBI JCL data set are allocated on volume COBI01. The two COBI input data sets are allocated on volume COBI02.

Note: Even when overriding, the VOLID parameter must be specified to provide a value for the corresponding symbolic parameter in the procedure.

Overriding the Blocksize for the Input Data Sets: The optimum and default blocksize for the COBI input data sets is 3200 bytes. However, the user may specify a smaller blocksize by overriding both DD statements and specifying the DCB parameter with the BLKSIZE subparameter. The blocksize specified must be the same for both input data sets, a multiple of 80, and within the range 400 through 3200. If any of these conditions are not met, a message is issued and initialization terminates.

The following example shows the use of the COBIBLD procedure while specifying a blocksize of 2400 bytes for the input data set:

```
//DSINIT      JOB    ---
//JOBLIB      DD     DSN=OSRTS.JOBLIB,DISP=SHR
//INITSTP     EXEC   COBIBLD,VOLID=COBI01
//CBSYSINA    DD     DCB=BLKSIZE=2400
//CBSYSINB    DD     DCB=BLKSIZE=2400
```

All four data sets are allocated on volume COBI01 and a blocksize of 2400 is used for the two input data sets.

Note: The use of a smaller blocksize for the input data sets may cause increased CALL-OS processing and OS/360 reader overhead.

Executing U#5INIT as a Separate Program

If the COBIBLD procedure is not used, the U#5INIT utility must be executed to initialize the COBI index, COBI JCL data set, and the two input data sets. All four data sets must be initialized at the same time and must meet the following requirements:

- Each must reside on a 2314 or 2319 storage device.
- The space allocated for each data set must be contiguous and must have only one extent.
- Secondary allocation may not be requested for any data set.

The JCL required to execute the U#5INIT utility is as follows:

```
//ANYNAME  JOB      ---
//JOBLIB   DD       DSN=OSRTS.JOBLIB,DISP=SHR
//STEPNAME EXEC     PGM=U#5INIT,PARM=('CBINDEX=aaa','CBJCLRD=bbb',
//                  'CBRDSET=mm')
//CBNDX    DD       DSN=qualifier.CBNDX,SPACE=(48,(aaa),,CONTIG),
//                  VOL=SER=volid1,DISP=(list),UNIT=2314
//CBJCL    DD       DSN=qualifier.CBJCL,SPACE=(3440,(bbb),RLSE,CONTIG,ROUND),
//                  VOL=SER=volid2,DISP=(list),UNIT=2314
//CBSYSINA DD       DSN=qualifier.CBSYSINA,SPACE=(CYL,(nn),,CONTIG),
//                  VOL=SER=volid3,DISP=(list),UNIT=2314[,
//                  DCB=BLKSIZE=yyyy]
//CBSYSINB DD       DSN=qualifier.CBSYSINB,SPACE=(CYL,(nn),,CONTIG),
//                  VOL=SER=volid4,DISP=(list),UNIT=2314[,
//                  DCB=BLKSIZE=yyyy]
//SYSPRINT DD       SYSOUT=A
```

where

aaa — is the number of 48-byte records to be allocated in the COBI index. This number is installation dependent and is based on the number of submitted jobs that might be present in the system at any one time. The number must be within the range from 2 through 32,000. If omitted, the default is 150. If present, this number must be specified in the CBINDEX parameter of the EXEC statement and in the SPACE parameter of the CBNDX DD statement; the number should be the same in both places.

bbb — is the number of 3440-byte records to be allocated in the JCL data set. This number should be twice the number of COBI index records specified and must be at least twice the number of records specified for a record set. If omitted, the default is 300. If present, this number must be specified in the CBJCLRD parameter of the EXEC statement and in the SPACE parameter of the CBJCL DD statement; the number should be the same in both places. In addition, the space allocated for the JCL data set must begin on a cylinder boundary. Since the specification is not made in cylinders, ROUND is required to force cylinder boundary alignment and RLSE is required to ensure that unformatted tracks are made available.

mm — is the number of JCL records in each JCL record set and is specified only in the CBRDSET parameter of the EXEC statement. The number must be within the range from 1 through 4. If omitted, the default is 2.

qualifier — is the high level index qualifier assigned to CALL-OS data sets during system build. If the usual default

value OSRTS is desired, then OSRTS must be specified as the qualifier.

volid1
volid2
volid3
volid4 — indicate the volume serial number of the volume to which the associated data set is to be assigned. The same or different volumes may be used for all four data sets.

list — specifies the disposition of the data set. The recommended disposition is DISP=(NEW,CATLG,DELETE). This ensures that the data sets are cataloged if the processing is successful, and are deleted if processing terminates abnormally. If an old data set is used and DISP=(OLD) is specified, the data set must be completely empty, as if newly created.

nn — is the number of cylinders to be allocated for each input data set. This number must be identical for both data sets.

yyyy — is optional and if used specifies the blocksize for the input data sets. The blocksize must be the same for both data sets, a multiple of 80, and within the range 400 to 3200; if any of these conditions are not met, initialization terminates. If the DCB parameter is not specified, a default of 3200 is used.

Table 3 shows the defaults for those parameters which may be omitted.

Table 3. Parameter defaults for the U#5INIT utility

Parameter	Use	Default
BLKSIZE on the CYSYSINA and CYSYSINB DD statements	Specifies the maximum blocksize allowed for the COBI input data sets	3200 bytes
CBINDEX	Specifies the number of 48-byte records to be allocated to the COBI index data set	150 records
CBJCLRD	Specifies the number of 3440-byte records to be allocated to the COBI JCL data set	300 records
CBRDSET	Specifies the number of 3440-byte records in each JCL record set	Two records per set

LINK EDITING THE COBI READER AND WRITER LOAD MODULES

The load modules for the COBI reader and writer programs (DIBRDR and DIBWTR, respectively) must be link edited into SYS1.LINKLIB. This link edit is performed as the second step of the COBIBLD cataloged procedure. The COBIBLD procedure also initializes and preformats the COBI index, JCL data set, and the COBI input data sets. The JCL required to execute COBIBLD is explained in detail in "Initializing the COBI Data Sets".

Figure 17 shows the JCL statements contained in the second step of the COBIBLD procedure. Note that the linkage editor control statements are contained in another cataloged procedure, DIBCBINC; Figure 18 shows the statements in this cataloged procedure. If the COBIBLD procedure is not used, the user must execute the linkage editor separately, using the same JCL and linkage editor control statements.

```
                .
                .
                .
//CB2           EXEC    PGM=IEWL,PARM='XREF,LIST,NCAL',
//                      COND=(0,NE,CB1),REGION=96K
//SYSLIB        DD      DSN=&QA..JOBLIB,DISP=SHR
//SYSLMOD       DD      DSN=SYS1.LINKLIB,DISP=OLD
//SYSUT1        DD      UNIT=SYSDA,SPACE=(1024,(200,20))
//SYSPRINT      DD      SYSOUT=A
//SYSLIN        DD      DSN=SYS1.PROCLIB(DIBCBINC),DISP=SHR
```

Figure 17. JCL statements in the second step of the COBIBLD procedure

```
INCLUDE SYSLIB(DIBRDR)
NAME    DIBRDR(R)
INCLUDE SYSLIB(DIBWTR)
NAME    DIBWTR(R)
```

Figure 18. Linkage editor control statements in the DIBCBINC procedure

MAINTAINING THE COBI DATA SETS

After a CALL-OS system with COBI has been in use for a period of time, it may be necessary to perform maintenance operations on the COBI data sets. IBM supplies three utility programs to aid in this maintenance, as follows:

- U#5CBXPN Expands the current COBI index and the COBI JCL data set by copying them into larger data sets; it will also contract the COBI JCL data set by copying it into a smaller data set
- U#5RINIT Reinitializes the COBI index, JCL, and input data sets
- U#5PURGE Deletes all references to unfinished jobs from the COBI data sets

The use, JCL requirements, and an example for each utility is given in the following text.

U#5CBXPN - EXPANDING THE COBI INDEX AND JCL DATA SET

When the COBI data sets are initialized, the sizes of the COBI index and the JCL data set are estimated according to the number of users who will be using the system at any one time. If the number of jobs being submitted is more than was originally expected, or if the average length of JCL listings is different than anticipated, the sizes of these data sets may have to be adjusted. Because the COBI index and JCL data set must have only one extent, a utility has been provided to copy the COBI index data set into a new data set and to compress or expand the JCL data set.

Specifically, the U#5CBXPN utility copies the COBI index from its present data set into another data set, which may differ in size; it

copies the COBI JCL data set from its present data set into another data set, which may be the same size, larger, or smaller. If a larger data set is used, in either case, U#5CBXPN formats the remainder of the data set into empty records, filled with zeros.

JCL Requirements

The current COBI index and the new COBI index must be defined by DD statements with the names CBOLDX and CBNDX, respectively. The current COBI JCL data set and the new COBI JCL data set must be defined by DD statements with the names CBOLDJCL and CBJCL, respectively. In addition, a SYSPRINT DD statement must be supplied to define the system printer to be used for utility messages.

The user may also specify the following information on the EXEC statement:

- The total number of records which the expanded COBI index is to contain
- The total number of records which the expanded (contracted) COBI JCL data set is to contain
- The number of records in a JCL record set

If the parameters are omitted, the defaults are the values used to initialize the current COBI index and JCL data sets. Thus, unless specific parameter values are given, the new data sets will have the same size and structure as the current data sets.

The complete JCL required to execute the U#5CBXPN utility is as follows:

```
//ANYNAME     JOB     ---
//JOBLIB      DD      DSN=qualifier.JOBLIB,DISP=SHR
//STEPNAME    EXEC    PGM=U#5CBXPN,PARM=('CBINDEX=aaa',
//                    'CBJCLRD=bbb','CBRDSET=mm'),REGION=88K
//CBOLDX      DD      DSN=qualifier.CBNDX,DISP=(OLD,DELETE,KEEP)
//CBNDX       DD      DSN=user-name-1,SPACE=(48,(aaa),,CONTIG),
//                    UNIT=2314,DISP=(NEW,CATLG,KEEP),
//                    VOL=SER=volid1
//CBOLDJCL    DD      DSN=qualifier.CBJCL,DISP=(OLD,DELETE,KEEP)
//CBJCL       DD      DSN=user-name-2,UNIT=2314,
//                    SPACE=(3440,(bbb),RLSE,CONTIG,ROUND),
//                    VOL=SER=volid2,DISP=(NEW,CATLG,KEEP)
//SYSPRINT    DD      SYSOUT=A
```

where

qualifier — is the high level index qualifier assigned to CALL-OS data sets during system build.

aaa — is the total number of 48-byte records to be allocated to the expanded COBI index. The number cannot exceed 32,000 and is specified in the CBINDEX parameter of the EXEC statement and in the SPACE parameter of the CBNDX DD statement. If the parameter is omitted from the EXEC statement, the value specified in the CBNDX DD statement must be equal to the number specified when the data set was initialized.

bbb — is the number of 3440-byte records to be allocated to the expanded or contracted COBI JCL data set. This number is specified in the CBJCLRD parameter of the EXEC statement

and in the SPACE parameter of the CBJCL DD statement. If the parameter is omitted from the EXEC statement, the value specified in the CBJCL DD statement must be equal to the number specified when the JCL data set was initialized.

mm — is the number of JCL records in each JCL record set. This number must be within the range from one through four and is specified only in the CBRDSET parameter of the EXEC statement. If the parameter is omitted, the default is the number of records per set specified when the JCL data set was initialized.

user-name-1
user-name-2 — are user-specified data set names; the qualifier is not needed. U#5CBXPN replaces these data set names with the old data set names (typically, qualifier.CBNDX and qualifier.CBJCL).

volid1
volid2 — are the volume serial numbers of the volumes which are to contain the expanded COBI index and the expanded (contracted) JCL data set, respectively; they may specify the volumes which contain the current data sets.

Note the way in which the disposition is specified on the CBOLDX, CBOLDJCL, CBNDX, and CBJCL DD statements. If the system should fail prior to completion of the expansion, these dispositions ensure that all data sets are retained. The user may then determine which data sets are to be deleted.

After the expanded data sets are created, the data set names originally assigned to the old data sets are assigned to the expanded data sets. To avoid duplicate names, an interim name is assigned to each old data set. These interim names have the following format:

```
DSN=DIB.Tnnnnnnn.Dmmmmm.CBNDX
DSN=DIB.Tnnnnnnn.Dmmmmm.CBJCL
```

where

nnnnnnn is the time of day

mmmmm is the date

For example, after the interim name is assigned to the old COBI index, the old name is assigned to the expanded COBI index. This entire process is then repeated for the JCL data set. The old COBI index and the old COBI JCL data sets are deleted after the utility terminates.

In the unlikely event that the utility should terminate abnormally while the data set names are being changed, the old data sets are retained. The actual names under which the old data sets have been stored can be determined by listing the volume table of contents (VTOC).

U#5CBXPN uncatalogs the old data sets. Note that if the new data sets are not cataloged, no message is issued.

If U#5CBXPN is executed in a multi-step job, the CBOLDX, CBOLDJCL, CBNDX, and CBJCL DD statements should not contain backward references to DD statements in previous steps of the job. Similarly, in subsequent steps, no backward references should be made to the CBOLDX, CBOLDJCL, CBNDX, and CBJCL DD statements of the current step.

Example

The following example shows the use of the U#5CBXPN utility:

```
//EXPND       JOB     ---
//JOBLIB      DD      DSN=OSRTS.JOBLIB,DISP=SHR
//CBX1        EXEC    PGM=U#5CBXPN,REGION=88K
//                    PARM='CBINDEX=200,CBJCLRD=400'
//CBOLDX      DD      DSN=OSRTS.CBNDX,DISP=(OLD,DELETE,KEEP)
//CBNDX       DD      DSN=TEMP1,UNIT=2341,SPACE=(48,(200),,CONTIG),
//                    VOL=SER=COBI01,DISP=(NEW,CATLG,KEEP),
//CBOLDJCL    DD      DSN=OSRTS.CBJCL,DISP=(OLD,DELETE,KEEP)
//CBJCL       DD      DNS=TEMP2,UNIT=2314,SPACE=(3440,(400),RLSE,
//                    CONTIG,ROUND),
//                    VOL=SER=COBI01,DISP=(NEW,CATLG,KEEP)
//SYSPRINT    DD      SYSOUT=A
/*
```

U#5RINIT - REINITIALIZING THE COBI DATA SETS

The U#5RINIT utility reinitializes the COBI index, JCL data set, and the two input data sets. In addition, the utility produces a SCRATCH control statement (for use with the OS/360 utility IEHPROGM) for each scannable output data set which has not already been scratched.

This utility is used when it becomes desirable to recreate the initial COBI operating environment. For example, although scannable data sets are designed as temporary data sets and should be scratched as soon as possible after their use, it is possible that too many scannable data sets have been kept; this utility prevents the data from being lost, while at the same time restores the COBI environment to its initial state. This utility could also be used on a periodic basis within certain time sharing environments to reinitialize COBI; for example, in a university environment, the U#5RINIT utility could be run at the end of a semester to prepare for the next term's time sharing work.

The reinitialization process is identical to that described for initialization (U#5INIT) with the following exceptions:

- The utility punches a SCRATCH control statement for each unscratched scannable output data set, prior to reinitializing the COBI index and JCL data sets.
- The space for the reinitialized input data sets may be either old or new. If old space is used (for example, if the current data set is to be reinitialized), the space does not have to be empty. If new space is used, the amount of space does not have to be equal to the amount allocated when the data sets were initialized.
- The COBI input data sets are expected to have an end-of-file record at the beginning of the data set. If this record is not found by U#5RINIT, it may indicate either that the jobs in the data set have not yet been read by the COBI reader or that the space is newly allocated. If the space is not newly allocated, a message is issued to the system operator requesting him to either continue or terminate U#5RINIT processing.

Except for the above, the requirements for reinitialization are identical to those for initialization. That is, newly allocated space must be on a 2314 or 2319 volume, have only one extent, and not request a secondary allocation. If an error is detected during U#5RINIT processing, the COBI index and COBI JCL data sets are generally invalid for a rerun of U#5RINIT. The correct recovery procedure is to run U#5INIT to initialize new data sets.

JCL Requirements

The process of reinitializing the COBI data sets consists of two steps. The first step executes the U#5RINIT utility, which performs the actual reinitialization and processes the unscratched scannable output data sets. The second step executes the JOBFIND function of the data base utility, which updates the user group catalogs according to the reinitialized COBI index.

Executing U#5RINIT: The COBI index, JCL data set, and the two input data sets must be defined by DD statements with the names CBNDX, CBJCL, CBSYSINA, and CBSYSINB, respectively. The system printer, which is used to print any messages resulting from utility processing, must be defined by a SYSPRINT DD statement. Finally, the data set for the SCRATCH statements must be defined by a SYSPUNCH DD statement.

The complete JCL required to execute the U#5RINIT utility step is as follows:

```
//ANYNAME     JOB     ---
//JOBLIB      DD      DSN=qualifier.JOBLIB,DISP=SHR
//STEPNAME    EXEC    PGM=U#5RINIT,REGION=88K
//CBNDX       DD      DSN=qualifier.CBNDX,DISP=OLD
//CBJCL       DD      DSN=qualifier.CBJCL,DISP=OLD
//CBSYSINA    DD      DSN=qualifier.CBSYSINA,DISP=disposition[,
//                    DCB=BLKSIZE=yyyy][,UNIT=2314,VOL=SER=volid3,
//                    SPACE=(CYL,(nn),,CONTIG)]
//CBSYSINB    DD      DSN=qualifier.CBSYSINB,DISP=disposition[,
//                    DCB=BLKSIZE=yyyy][,UNIT=2314,VOL=SER=volid4,
//                    SPACE=(CYL,(nn),,CONTIG)]
//SYSPRINT    DD      SYSOUT=A
//SYSPUNCH    DD      SYSOUT=B
```

where

qualifier — is the high level index qualifier assigned to CALL-OS data sets during system build.

disposition — specifies the disposition for the input data sets. If an old data set is used, the disposition is OLD; with the exception of the DCB parameter on the DD statements for the input data sets, all the information enclosed in brackets must be omitted. If a new data set is used, the disposition is (NEW,CATLG,DELETE); with the exception of the DCB parameter, all the information enclosed in brackets must be specified as shown. The DCB parameter may be specified for either old or new input data sets.

volid3, volid4 — are the volume serial numbers of the volumes which are to contain the newly allocated space for the input data sets. Both data sets may be on the same volume or on different volumes. These parameters are not specified if an old data set is used.

nn — is the number of cylinders to be allocated to each input data set. This number and its associated SPACE parameter is specified only if new space is to be used for the input data sets; the number specified must be the same for both input data sets.

yyyy — is optional and specifies the blocksize for the input data sets; this parameter may be used for either old or new space. The blocksize specified must be the same for

both input data sets, a multiple of 80, and within the range from 400 through 3200; if any of these conditions are not met, reinitialization terminates. If the DCB parameter is not specified, a blocksize of 3200 bytes is used.

Executing the JOBFIND Function: The second step necessary for reinitializing the COBI data sets is the execution of the JOBFIND function of the data base utility. Since this step is executed after the COBI index has been reinitialized, it causes all COBI job entries in the user group catalogs to be deleted. This step must be executed for all user groups associated with the system for which the COBI data sets were reinitialized. This includes any user groups from either cluster which may have been active during any session of this particular CALL-OS system during which submission of COBI jobs was permitted. Therefore, the user group DD statements may define user groups from both clusters, with identical, overlapping, or nonoverlapping user number ranges. All data sets associated with a user group must be present. The requirements for execution of the data base utility (DIBCADBU) are described in detail in the section on the utility in the section "Creating and Maintaining the Data Base".

Example

The following example shows the two steps required to reinitialize the four COBI data sets:

```
//RINIT       JOB     ---
//JOBLIB      DD      DSN=OSRTS.JOBLIB,DISP=SHR
//STEP1       EXEC    PGM=U#5RINIT,REGION=88K
//CBNDX       DD      DSN=OSRTS.CBNDX,DISP=OLD
//CBJCL       DD      DSN=OSRTS.CBJCL,DISP=OLD
//CBSYSINA    DD      DSN=OSRTS.CBSYSINA,DISP=OLD
//CBSYSINB    DD      DSN=OSRTS.CBSYSINB,DISP=OLD
//SYSPRINT    DD      SYSOUT=A
//SYSPUNCH    DD      SYSOUT=B
//STEP2       EXEC    PGM=DIBCADBU,REGION=96K
//INDEX       DD      DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT    DD      SYSOUT=A
//CBNDX       DD      DSN=OSRTS.CBNDX,DISP=OLD
//SYSGRP00    DD      DSN=OSRTS.SYSGRP00,DISP=SHR
//IBMIBM00    DD      DSN=OSRTS.IBMIBM00,DISP=OLD
//IBMIBM01    DD      DSN=OSRTS.IBMIBM01,DISP=OLD
//IBMIBM02    DD      DSN=OSRTS.IBMIBM02,DISP=OLD
//IBMIBM03    DD      DSN=OSRTS.IBMIBM03,DISP=OLD
//REGREG40    DD      DSN=OSRTS.REGREG40,DISP=OLD
//SYSIN       DD      *
./ JOBFIND USRGROUP=IBMIBM,CLUSTER=1,PASSWORD=COMMCON
./ JOBFIND USRGROUP=REGREG,CLUSTER=2,PASSWORD=COMMCON
/*
```

U#5PURGE - PURGING UNFINISHED JOBS FROM THE COBI DATA SETS

This utility purges the COBI data sets by removing all references to COBI jobs which are in an unfinished state. COBI jobs may enter this state when a system failure occurs while the COBI reader is reading one of the COBI input data sets. To make the system operational after such a failure, the operator must reformat the OS/360 system job queue, thereby destroying all OS/360 references to COBI jobs not yet finished. However, references to these unfinished jobs remain in the COBI data sets and in the catalogs of the submitting users.

The purge process removes references to unfinished COBI jobs from the COBI data sets, as well as the user group catalogs. The COBI data sets should be purged after the job queue has been reformatted and the OS/360 system is again operational. However, before the purge function can be executed successfully, the following conditions must be met:

1. The CBSYSINA and CBSYSINB data sets have been read by the COBI reader progrom until both data sets have the EOF record in the first record.

2. All the jobs read in by the COBI reader have been processed by OS/360 and have completed execution.

3. The COBI writer program has been executed and copied the necessary information into the COBI JCL data set, converted COBI job output to the appropriate OS/360 output class, and indicated in the COBI index that all the jobs thus processed are complete.

When these three conditions are met, the only COBI index job entries left in an unfinished state are those for which the job queue information was destroyed during reformatting. These job entries and their associated data sets may then be removed from the system by the purge process.

JCL Requirements

The process of purging unfinished jobs from the COBI data sets consists of two steps. The first step executes the U#5PURGE utility, which removes the entries for the unfinished jobs from the COBI data sets and scratches any data sets associated with those jobs. The second step executes the JOBFIND function of the data base utility, which deletes the references to the unfinished jobs from the user group catalogs affected.

Executing U#5PURGE: The COBI index, the two input data sets, and the system printer must be defined by DD statements with the names CBNDX, CBSYSINA, CBSYSINB, and SYSPRINT, respectively. Finally, all volumes which could contain scannable output data sets associated with any incomplete jobs must be mounted and defined by SCANxx DD statements. The complete JCL required to execute the U#5PURGE utility step is as follows:

```
//ANYNAME     JOB     ---
//JOBLIB      DD      DSN=qualifier.JOBLIB,DISP=SHR
//STEPNAME    EXEC    PGM=U#5PURGE,REGION=88K
//CBNDX       DD      DSN=qualifier.CBNDX,DISP=OLD
//CBSYSINA    DD      DSN=qualifier.CBSYSINA,DISP=OLD
//CBSYSINB    DD      DSN=qualifier.CBSYSINB,DISP=OLD
//SYSPRINT    DD      SYSOUT=A
//SCANxx      DD      VOL=SER=volidn,UNIT=2314,DISP=OLD
```

where

qualifier — is the high level index qualifier assigned to CALL-OS data sets during system build.

xx — identifies a DD statement which defines a volume which may contain scannable output data sets associated with unfinished jobs. One or more SCANxx DD statements must be supplied; the SCANxx statements must be identical to the SCANxx DD statements in the CALL-OS startup deck for

the session which was terminated abnormally. No more than 100 SCANxx DD statements may be supplied.

volidn — is the volume serial number of a volume which may contain scannable output data sets associated with unfinished jobs. All volumes upon which scannable output data sets were written must be mounted and defined by SCANxx DD statements.

Executing the JOBFIND Function: The second step necessary for purging unfinished jobs from the COBI data sets is the execution of the JOBFIND function of the data base utility. Since this step is executed after any reference to unfinished jobs have been removed from the COBI data sets, it causes all references to such jobs to be deleted from the user group catalogs. This step must be executed for all user groups which were active in the CALL-OS system in execution when the OS/360 system terminated abnormally. Therefore, the user group DD statements must be identical to those in the CALL-OS startup deck for this session. The requirements for execution of the data base utility (DIBCADBU) are described in detail in the section on the utility in the section "Creating and Maintaining the Data Base".

Example

The following example shows the two steps required to purge unfinished COBI jobs from the CALL-OS system:

```
//PURGE       JOB     ---
//JOBLIB      DD      DSN=OSRTS.JOBLIB,DISP=SHR
//STEP1       EXEC    PGM=U#5PURGE,REGION=88K
//CBNDX       DD      DSN=OSRTS.CBNDX,DISP=OLD
//CBSYSINA    DD      DSN=OSRTS.CBSYSINA,DISP=OLD
//CBSYSINB    DD      DSN=OSRTS.CBSYSINB,DISP=OLD
//SYSPRINT    DD      SYSOUT=A
//SCAN01      DD      DISP=OLD,UNIT=2314,VOL=SER=222222
//SCAN05      DD      DISP=OLD,UNIT=2314,VOL=SER=COBI01
//STEP2       EXEC    PGM=DIBCADBU,REGION=96K
//INDEX       DD      DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT    DD      SYSOUT=A
//CBNDX       DD      DSN=OSRTS.CBNDX,DISP=OLD
//SYSGRP00    DD      DSN=OSRTS.SYSGRP00,DISP=SHR
//DEVDEV00    DD      DSN=OSRTS.DEVDEV00,DISP=OLD
//DEVDEV01    DD      DSN=OSRTS.DEVDEV01,DISP=OLD
//ENGENG40    DD      DSN=OSRTS.ENGENG40,DISP=OLD
//ENGENG41    DD      DSN=OSRTS.ENGENG41,DISP=OLD
//SYSIN       DD      *
./ JOBFIND USRGROUP=DEVDEV,CLUSTER=1,PASSWORD=COMMCON
./ JOBFIND USRGROUP=ENGENG,CLUSTER=2,PASSWORD=COMMCON
/*
```

Before the process of system installation is undertaken, a decision must have been made by the user with respect to the following items:

- System configuration required to support CALL-OS
- Amount of disk storage facility required to fully support the installation
- Amount of core storage required to fully support the installation
- Performance level required by CALL-OS

The requirements of a given installation environment, coupled with the system performance level desired by the user, determine the proper amounts of the items mentioned above. Adequate planning and analysis, in light of these factors, is therefore required.

SYSTEM CONFIGURATION

The selection of a hardware configuration for CALL-OS is largely determined by the requirements of OS/360, CALL-OS, and the application requirements of the specific installation. Some of the factors involved in determining those needs are:

- Maximum number of terminals having concurrent access to CALL-OS
- Expected levels of performance
- Desired amount of concurrent activity in other OS/360 task areas

The total number of concurrently active terminal lines that can be supported by CALL-OS with suitable response times is directly proportional to the mix of terminal user applications actively on the system. This number may be further affected by other concurrently active OS/360 task areas as well as by the size and priority of the CALL-OS task area.

MINIMUM MACHINE CONFIGURATION

The configuration selected is comprised of the central processing unit and the peripheral equipment required for online operation of CALL-OS. The minimum central processing unit on which CALL-OS can be executed is any one of the following:

- System/360 Model 50HG
- System/370 Model 145H (384K) with:

  3345 Main Storage Frame
  4901 Main Storage Frame Adapter
  3046 Power Storage 3910 Extended Precision Floating-Point Feature (optional feature, but no charge)

- System/370 Model 155HG

The minimum peripheral equipment required for online operation of CALL-OS with each CPU is:

- System/360 Model 50HG:

  One selector channel
  One IBM 2314 Storage Control Model A1
  One IBM 2312 Disk Storage Model A1
  One IBM 2702 or 2703 Transmission Control
  Two terminal consoles (see below)

- System/370 Model 145H (384K, as defined above)

  One IBM 2319 Disk Storage Facility Model A1
  One IBM Integrated File Adapter feature (#4650)
  One IBM 2702 or 2703 Transmission Control
  Two terminal consoles (see below)

- System/370 Model 155HG:

  One block multiplexor channel
  One IBM 2314 Disk Storage Control Model A1
  One IBM 2312 Disk Storage Model A1
  One IBM 2702 or 2703 Transmission Control
  Two terminal consoles (see below)

The two terminal consoles are used for system communication. One serves as a command console from which the operator issues special system commands. The other serves as a communications console for recording system error messages and activity. The OS/360 system operator's console is used to initialize CALL-OS and may serve as the communications console, thus reducing to one the number of terminal consoles required. CALL-OS supports the following terminals:

- IBM 2741 Communications Terminal (Correspondence or EBCD)

- Teletype Units,* Type 33 or 35

Any of the above terminals can be used as a command console, communications console, or a user terminal. No more than 255 terminals (including the command and communications consoles) can be simultaneously online with CALL-OS.

In addition to the devices named in the preceding text, the utilities used for offline system support and maintenance require extra peripheral equipment. Depending on the utilities to be used, the following may be needed:

- One printer output unit, OS/360-supported, with 120 print positions and graphics equivalent to the PN print arrangement

- One punched output unit (see OS/360 minimum system requirements)

- One card input unit (see OS/360 minimum system requirements)

- One OS/360-supported magnetic tape unit (any model)

Any peripheral devices, in addition to those given above, will be supported within the limits of OS/360 support. Specifically, CALL-OS can use additional selector channels and any appropriate 2314 Direct Access Storage Facility A or B Series configuration.

The relationships of hardware devices to CALL-OS and OS/360 are depicted in Figure 19. Further details of system configuration can be found in CALL-OS System Description Manual.

---

*Trademark of the TELETYPE Corporation

MINIMUM STORAGE REQUIREMENTS

The minimum task area size required for use of CALL-OS is 224K. This allows a configuration of ten lines with BASIC and FORTRAN compilers and an object program size of 52K. A typical BASIC language program of 300 statements could be expected to require 52K. Additional task area space is required for use of the PL/I compiler or the CALL-OS Batch Interface (COBI) facility, larger user programs, larger terminal networks, or improved performance in a large network environment. For more information, see "Core Storage Requirements" later in this section.

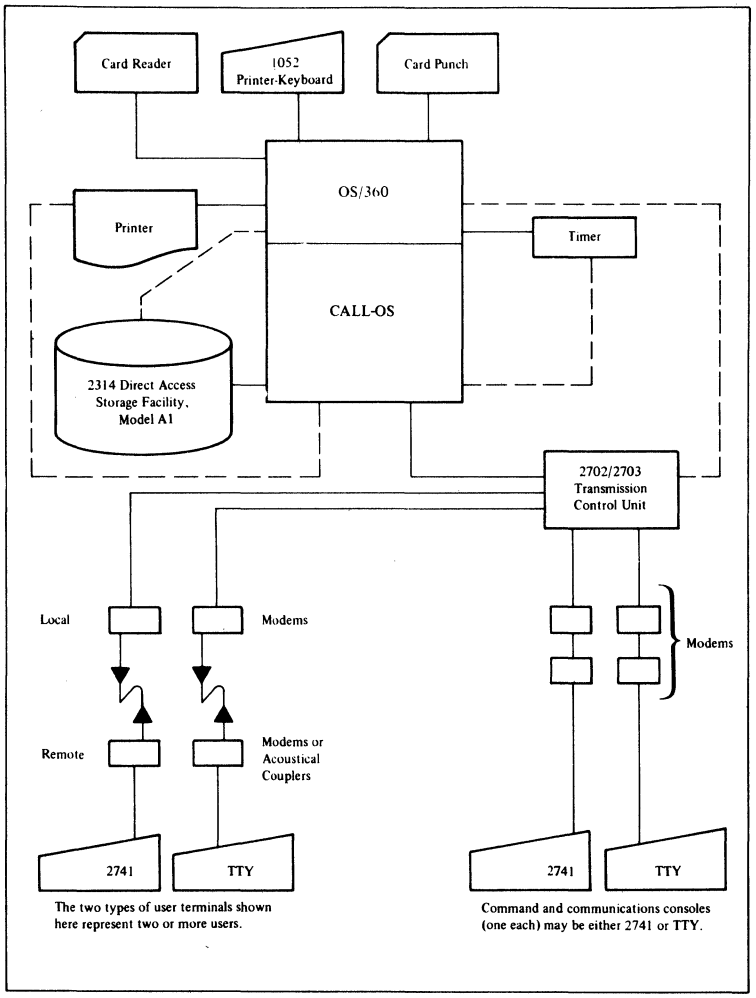


Figure 19. CALL-OS system hardware configuration

DATA SET ALLOCATION

A primary consideration attending the installation of CALL-OS is a well-planned distribution of disk storage allocation. Performance is affected much more significantly by disk layout, and the availability of a dedicated channel, than it is by disk storage capacity. The minimization of disk seek time, made possible by judicious data set allocation, is one of the keys to successful system installation.

CALL-OS data sets fall into two general categories: high usage, which consist of the work/swap, overlay, and compiler data sets; and low

usage, which consist of the index, system group, and user group data sets. High-usage data sets account for 70 to 80 percent of all disk accesses. These particular data sets are referred to as "central cylinder functions", because they should be placed physically in the center of the disk pack, while the user program save area and all other data sets are placed on either the inside or the outside of the pack.

Figure 20 illustrates the central cylinder concept as viewed from the top of a disk pack. Allocation of high-usage data sets begins with cylinder 100 and continues outward in either direction. For example, cylinder 99 and cylinder 101, cylinder 98 and cylinder 102, and so on.

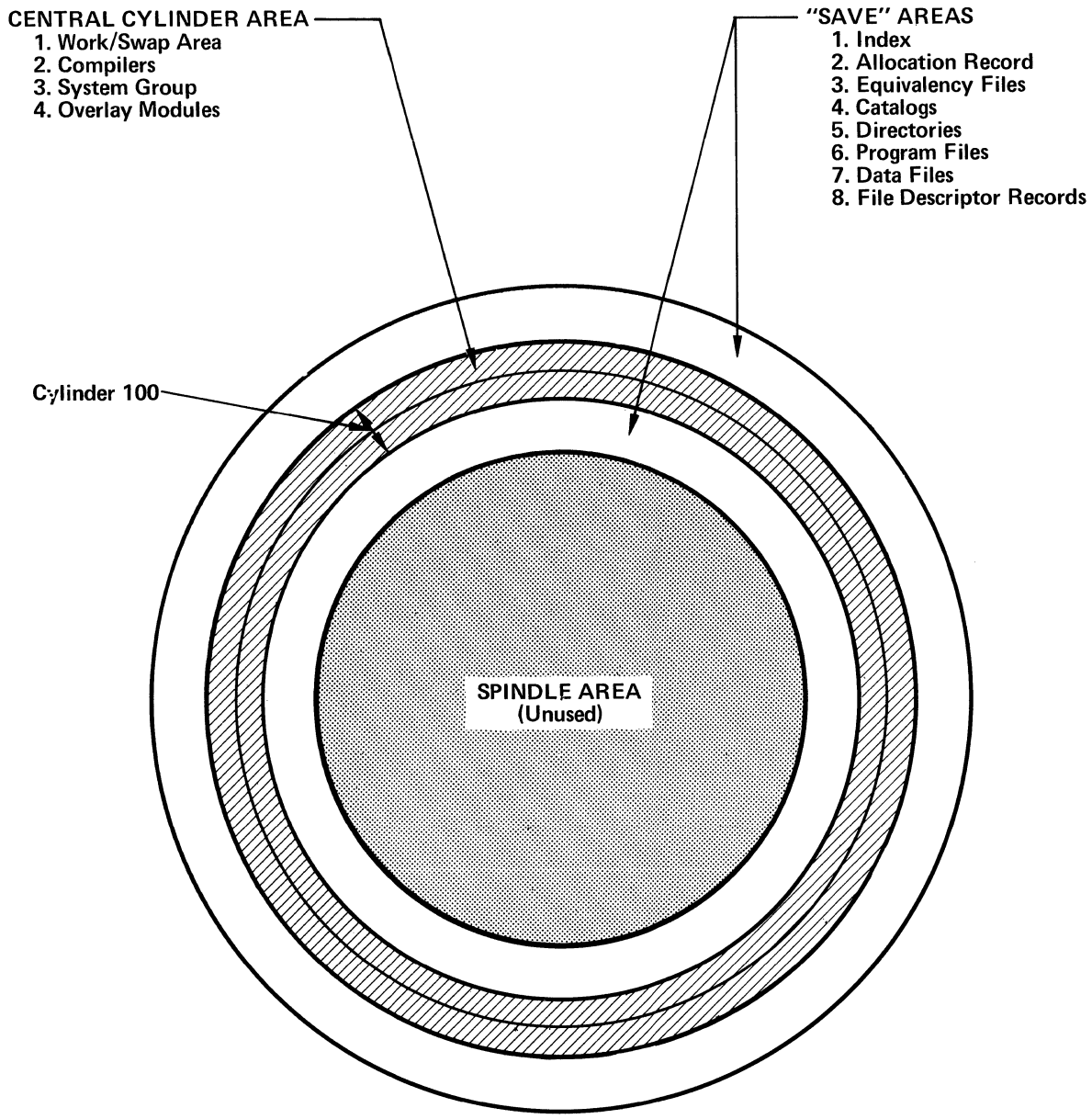


Figure 20. The central cylinder concept (top view)

The central cylinder functions contain six data sets, plus N work/swap data sets where N is the number of packs assigned to CALL-OS. The work/swap data sets should be allocated evenly across all packs. Each communications line supported by the system requires one cylinder;

hence, the number of cylinders of work/swap allocated on each pack should be the number of terminals divided by the number of packs.

The following tables indicate how the central cylinders should be laid out as a function of the number of packs. The physical placement of data sets within a group should also be as shown:

One Pack

(1) PLI
(2) PL2
(3) OVERLAY
(4) WORK/SWAP (physical center)
(5) BASIC
(6) FORTRAN
(7) SYSGROUP

Two Packs

Pack #1		Pack #2	
(1)	BASIC	(1)	OVERLAY
(2)	WORK/SWAP	(2)	WORK/SWAP
(3)	PLI	(3)	SYSGROUP
(4)	PL2	(4)	FORTRAN

Three Packs

Pack #1		Pack #2		Pack #3	
(1)	WORK/SWAP	(1)	OVERLAY	(1)	FORTRAN
(2)	PLI	(2)	WORK/SWAP	(2)	WORK/SWAP
(3)	PL2	(3)	BASIC	(3)	SYSGROUP

Four Packs

Pack #1		Pack #2		Pack #3	
(1)	BASIC	(1)	OVERLAY	(1)	WORK/SWAP
(2)	WORK/SWAP	(2)	WORK/SWAP	(2)	PLI
				(3)	PL2

Pack #4

(1) SYSGROUP
(2) WORK/SWAP
(3) FORTRAN

Five Packs

Pack #1		Pack #2		Pack #3	
(1)	BASIC	(1)	OVERLAY	(1)	FORTRAN
(2)	WORK/SWAP	(2)	WORK/SWAP	(2)	WORK/SWAP

Pack #4		Pack #5	
(1)	SYSGROUP	(1)	PLI
(2)	WORK/SWAP	(2)	PL2
		(3)	WORK/SWAP

Six Packs

Pack #1	Pack #2	Pack #3
(1) BASIC	(1) OVERLAY	(1) FORTRAN
(2) WORK/SWAP	(2) WORK/SWAP	(2) WORK/SWAP
Pack #4	**Pack #5**	**Pack #6**
(1) SYSGROUP	(1) WORK/SWAP	(1) PLI
(2) WORK/SWAP		(2) PL2
		(3) WORK/SWAP

For any configuration above six packs, the first six packs appear as above; all other packs contain only work/swap data sets.

The preceding configurations were arrived at by ranking the data sets as to number of I/O requests, most frequent to least frequent, as follows:

(1) WORK/SWAP
(2) OVERLAY
(3) BASIC
(4) SYSGROUP
(5) FORTRAN
(6) PLI
(7) PL2

If an installation finds that its data set usage is other than that described, appropriate substitutions should be made. Some conditions which affect the priority sequence are:

1. Making resident all high-usage overlays (for example, modules required for run, list, load, and save functions. This greatly reduces the usage of the overlay data set, shifting it from second place to probably last place in the priority list.

2. Having the majority of the terminal users with a language orientation other than that given in the priority sequence (BASIC, FORTRAN, and then PL/I). For example, some installations may have most terminal users using only FORTRAN, in which case the FORTRAN data set would shift from fifth position to third position.

The foregoing discussion assumes dedicated packs, even though dedication is not required. If a user installation chooses not to have dedicated packs, then the high activity data sets should be placed on packs with relatively little other activity.

The placement of data sets is important only as a function of the number of users. As the number of users increases, the data base should be spread over more volumes.

CORE STORAGE REQUIREMENTS

One approach to system configuration is to determine what other tasks are to be run concurrently with the CALL-OS application, how much core is available for a task area, and the nature of the CALL-OS jobs to be performed. After a decision is made regarding those modules which should be resident, it is possible to determine the number of communications lines that could be brought up during system initialization; this determination is based on the amount of storage remaining in the CALL-OS task area. Another approach is to commence

with those CALL-OS jobs which are to be run with acceptable response times, and the number of terminals required to be supported, and then determine the task area size necessary to accomplish these objectives. The rest of this section describes how to compute the task area size, the way storage is allocated within the task area, module residency considerations, and hierarchy support considerations. The section ends with several examples of core requirements.

COMPUTING TASK AREA SIZE

As an aid in arriving at the desireable task area size, an analysis of the core requirements for the CALL-OS system configuration desired should be made. The core requirements are divided into four catagories:

- Basic fixed core requirement, which is that storage required for the CALL-OS nucleus; the actual amount required depends on whether COBI is used
- Variable core requirements, which depend on those items that vary from system to system; for example, the number of lines, buffers, the types of terminals, and, if COBI is used, the number of scannable data sets and the volumes on which they reside
- Optional core requirements, which depend on the particular system configuration chosen; for example, the number of modules made resident, the compilers used, and the size of the user program area
- OS/360 core requirements, which consist of the core needed by OS/360 routines and control blocks

Once the CALL-OS task area size has been computed, it is then possible to determine the amount of storage available for batch processing. All core requirements given are decimal approximations.

- Basic Fixed Core Requirements (RTOS1) - Choose one

CALL-OS Nucleus	56,700
CALL-OS/COBI Nucleus	65,600

- Variable Core Requirements

Each Line	512
Each Input Buffer (Pot) (Four pots for each line with a minimum of 60 pots)	24
Each Output Buffer (One buffer for each set of three lines with a minimum of five buffers)	256
Each User or System Group Data Set	132
Each Terminal Type (L=number of lines of this type)	120+4L
Each Translate Table (One table for each terminal type)	512
Each Work/Swap Data Set	120

- Additional Variable Core Requirements for COBI

Volume Identification Table (M=number of entries in the table)	12*M
Enqueue/Dequeue Table (N=number of users scanning data sets at any one time; the default is one for every ten lines with a minimum of two users)	4*(N+4)

DCB plus its Work Area (N is as previously defined)	160*N
DEBs - in MFT only (N is as previously defined)	100*N

- Optional Core Requirements

Overlay Buffer (see note)	
With COBI	7,260
Without COBI	5,700
Compiler (choose largest in system)	
BASIC	82,000
FORTRAN	88,000
PL/I	106,000
Sort Buffer	14,400
User Program Area (choose one)	
Minimum size (Pmin=size of largest object program)	Pmin
Medium size - this allows an old job area of 69,632 bytes (Pmax=114,688 bytes)	1.6Pmax
Maximum size	3Pmax

Note: The size of the overlay buffer is based on the size of the largest potentially nonresident module. If COBI is used, the largest potentially nonresident module is M#SUB; if COBI is not used, the largest module is M#CAT (see "Module Residency Considerations"). If the module is resident, the overlay buffer size required decreases. However, the total core required by the resident modules must be added to the task area size.

- Operating System Core Requirements

ABEND	6,000 - MVT
(To ensure complete dumps)	8,000 - MFT
Subpools	6,000

ALLOCATION OF STORAGE WITHIN THE TASK AREA

The CALL-OS task area core is allocated by the initialization program. First, the resident modules, buffers, and the compiler area are allocated. The remainder of the task area is used for the user program area, which is divided between the new job area and the old job area. The following algorithm is used.

- The first 56 units (one unit = 2048 bytes), or 114,688 bytes, are allocated to the new job area.
- The next 34 units, or 69,632 bytes, are allocated to the old job area.
- The next 44 units are allocated, one to each of the two subareas, until the old job area reaches 56 units.
- Space is then given to the new job area until it reaches 112 units.

Sizes of the new and old job areas are printed on the OS/360 system operator's console when the initialization process is completed.

After the new job area reaches 112 units, any storage left in the task area remains unused by CALL-OS. Therefore, for efficient use of

storage within the entire system, it is important that the task area size be assigned accurately.

MODULE RESIDENCY CONSIDERATIONS

To provide flexibility in the configuration of core storage used by CALL-OS, certain modules used by the system can be made either resident or nonresident. The residency option is controlled by the RESMODS and OVLY DD statements. The RESMODS DD statement specifies a data set containing a list of those modules which are to be made resident during the current system run. Module names which can appear in this list and their approximate sizes are:

Modules	Size (Dec. Bytes)	
M#ABSUB	1280	(see note)
M#ACCT	700	
M#CANCL	2100	(see note)
M#CAT	5700	
M#CBST	7240	(see note)
M#CCBA	400	
M#CCCO	900	(see note)
M#CCDA	400	
M#CCDI	1500	
M#CCME	400	
M#CCOF	1200	
M#CCRE	4700	
M#CCST	1000	
M#CCTE	2000	
M#CCUS	600	
M#CCVA	1500	
M#CCWA	600	
M#DIR	1600	
M#ECHO	500	
M#EDIT	2600	
M#ESCN	3100	
M#HELP	200	
M#IJCL	700	(see note)
M#ISCAN	6300	(see note)
M#ISUB	4600	(see note)
M#JCL	6100	(see note)
M#LDRD	600	
M#LIB	1600	
M#LIST	500	
M#LOAD	500	
M#LOG	2000	
M#MREM	3100	
M#MWSC	1800	
M#NAME	200	
M#NOTFY	1100	(see note)
M#OBJR	700	
M#PASS	700	
M#RDSO	1200	
M#RUN	800	
M#SAVE	500	
M#SCAN	6500	(see note)
M#SCR	3600	(see note)
M#SORT	1300	
M#STAT	600	
M#STOR	1700	
M#SUB	7260	(see note)
M#TIME	800	
M#WEAV	600	
M#WID	400	

M#WRSO	1600
T#TTYTAB	512
T#27CTAB	512
T#27ETAB	512

Note: This is a COBI module and should not be made resident if COBI is not to be used in this session of CALL-OS.

Four members of SYS1.PROCLIB are included with the system which can be used to provide a variety of resident configurations. The functions of these members are:

RTOSALL -- All modules are made resident. (Actual list specifies ALLRES.) For an all-resident system which is desired to be loaded into LCS, ALLRES(1) may be specified.

RTOSNONE -- All potentially nonresident modules are made nonresident. (Actual list specifies NONRES.)

RTOSLLRS -- The modules associated with the load, list, run, and save functions are made resident. These modules are M#CAT, M#DIR, M#LDRD, M#LIST, M#LOAD, M#RDSO, M#RUN, M#SAVE, M#SORT, and M#WRSO.

RTOSUSER -- All modules are made resident except for those modules associated with operator commands (that is, modules whose names begin with M#CC).

If a module is to be made nonresident, it is read from the JOBLIB data set during system initialization and rewritten into the overlay data set for later retrieval by the system executive. If the OVLY DD statement is present, but no RESMODS DD statement is provided, a nonresident system is assumed. If no OVLY DD statement is provided, a totally resident system is assumed.

The members listed provide a certain amount of flexibility in trading space and time. Finally, note that the size of the overlay buffer is based on the size of the largest module to be nonresident. If all modules are resident, no overlay buffer is allocated.

LCS AND HIERARCHY SUPPORT CONSIDERATIONS

OS/360 Hierarchy support may be used to place portions of CALL-OS in the IBM 2361 Large Capacity Storage (LCS). This support is implemented in three areas, as follows:

1. The linkage editor can be employed to spread the various parts of the system executive into the two hierarchies provided. For a description of this process, refer to the publication IBM System/360 Operating System: Linkage Editor and Loader.

2. To place modules which are potentially nonresident in hierarchy (H-1) storage, the module name in the RESMODS list can be followed by a hierarchy indicator in parentheses. Thus, M#LOAD(1) would cause M#LOAD to be placed in hierarchy 1. M#LOAD(0) can also be coded for hierarchy zero, but a zero value is automatically assumed if no indicator is present. Note that even when no OVLY DD statement is provided, the RESMODS data set is still read (since a totally resident system is assumed) to ensure the processing of any hierarchy indicators.

3. With the LCSRES parameter in the startup deck, any or all of the nine dynamic areas of core storage obtained by CALL-OS during system initialization can be placed in hierarchy 1. See the

description of the LCSRES parameter in the section "Initializing the System."

The following points should be carefully considered when hierarchy options are used:

1. No attempt is made by the system to obtain core from another hierarchy when no core remains in the specified hierarchy.

2. Error messages printed by the system indicate the system component for which storage allocation has failed and the initialization process has stopped.

3. It is anticipated that the user may have to try various task area sizes before he obtains the exact configuration desired.

4. To run CALL-OS in either an MFT partition or an MVT region which is defined entirely in H-1 storage, no LCS options should be specified, since requests for H-0 storage in such a task area are filled from the H-1 storage available.

No general statements may be made regarding the levels of performance attained when various system components are placed in hierarchy 1. It is assumed that the user employing this support has carefully studied both his own response requirements and the functions of the various CALL-OS system components, and that, furthermore, his selection of portions of the system for placement in H-1 storage is based on such a study. Hierarchy support is provided so that users planning to operate with such a configuration can easily place selected components in hierarchy 1 without being required to modify CALL-OS.

EXAMPLES OF CORE REQUIREMENTS

This subsection contains examples of core requirements for several CALL-OS systems. The system configurations are not meant to be typical systems, but were chosen for their value as examples. The figures used are those given in the preceding subsections.

Example 1

This example shows the small CALL-OS configuration with a task area size of approximately 219K. This system supports ten lines, one terminal type, the BASIC and FORTRAN compilers, and the minimum user program area size. The core requirements are as follows:

Fixed Core Requirement	56,700
10 Lines	5,120
60 Input Buffers	1,440
4 Output Buffers	1,280
1 User Group Data Set	132
1 System Group Data Set	132
1 Terminal Type	160
1 Translate Table	512
1 Work/Swap Data Set	120
Overlay Buffer	5,700
BASIC and FORTRAN Compilers	88,000
User Program Area (minimum size)	53,248
Operating System Core Requirements	12,000
TOTAL	224,544

Example 2

The CALL-OS system in this example supports 25 lines, one terminal type, all three compilers, and the medium user program area size. The core requirements are as follows:

Fixed Core Requirement	56,700
25 Lines	12,800
100 Input Buffers	2,400
9 Output Buffers	2,304
2 User Group Data Sets	264
1 System Group Data Set	132
1 Terminal Type	220
1 Translate Table	512
1 Work/Swap Data Set	120
Overlay Buffer	5,700
BASIC, FORTRAN, and PL/I Compilers	106,000
User Program Area (112K program size, medium size)	182,400
Operating System Core Requirements	12,000
TOTAL	381,552

Example 3

The CALL-OS system in this example supports 60 lines, three terminal types, all three compilers, and a user program area size of 2Pmax. In addition, the modules for the load, list, run, and save functions are made resident. The core requirements are as follows:

Fixed Core Requirement	56,700
60 Lines	30,720
240 Input Buffers	5,760
20 Output Buffers	5,120
5 User Group Data Sets	660
2 System Group Data Sets	264
3 Terminal Types	600
3 Translate Tables	1,536
2 Work/Swap Data Sets	240
Overlay Buffer	4,700
Resident load, list, save, and run functions	14,200
BASIC, FORTRAN, and PL/I Compilers	106,000
Sort Buffer	14,400
User Program Area (2Pmax)	229,376
Operating System Core Requirements	12,000
TOTAL	482,276

Note: Since M#CAT is to be made resident, the size of the overlay buffer decreases to the size of the next largest nonresident module, M#CCRE.

Example 4

The CALL-OS system in this example uses the COBI facility under MVT and supports 100 lines, three terminal types, all three compilers, and the maximum user program area size. The COBI volume identification table has a maximum of ten entries. (Note that the number of entries is not necessarily the number of scannable data set volumes mounted for this session; rather, this number must be the maximum number of such volumes mounted for this or any other previous session.) The number of users permitted to scan data sets at any one time is estimated as one per ten lines, or in this example, ten. In addition, the modules for the load, list, save, and run functions are made resident. The core requirements are as follows:

Fixed Core Requirement	65,600
100 Lines	51,200
400 Input Buffers	9,600
34 Output Buffers	8,704
10 User Group Data Sets	1,320
2 System Group Data Sets	264
3 Terminal Types	760
3 Translate Tables	1,536
4 Work/Swap Data Sets	480
Volume Identification Table (M=10)	120
Enqueue/Dequeue Table (N=10)	56
DCB plus Work Area (N=10)	1,600
Overlay Buffer	7,260
Resident load, list, save, and run functions	14,200
BASIC, FORTRAN, and PL/I compilers	106,000
Sort Buffer	14,400
User Program Area(3Pmax, maximum size)	344,064
Operating System Core Requirements	12,000
TOTAL	639,164

SUMMARY OF PERFORMANCE CONSIDERATIONS

The following is a list, in order of relative importance, of suggestions that will improve system performance. The items concerning disk channel and data set attributes are important only in relationship to the amount of background work activity.

1. Make CALL-OS the highest-priority task area.
2. Allocate enough core so that the user program area is at least 180K.
3. Make resident the modules that control the run, load, list, and save functions.
4. Allocate disk channels so that the CALL-OS task area has a dedicated channel.
5. Plan swap, overlay, and compiler data sets (central cylinder function) in the physical center of the packs, and distribute them evenly over the number of drives available.
6. Allocate a sort buffer.
7. Increase the number of 256-byte buffers if queues seem to occur regularly (examine *REPORT output)
8. Place the user group data sets on either side of the central cylinder functions, and distribute them evenly over the number of drives available.
9. Allocate two dedicated channels with data sets appropriately distributed.
10. Allocate enough core so that the user program area is 336K.
11. Make resident all modules that process user commands.
12. Make all modules resident.
13. Allocate more dedicated channels.
14. If COBI is not being used, adjust the background time-slice algorithm to give 100% to CALL-OS. This is done with the *BATCH command, as described in the publication CALL-OS Operator's Manual.

BUILDING THE SYSTEM

The process of building a system with which to run CALL-OS may require an OS/360 system generation, as well as the execution of certain programs which actually build the CALL-OS system. This section summarizes the system generation requirements, as they apply to CALL-OS, and describes in detail the actual CALL-OS system build process.

Note: If COBI is to be used and a new OS/360 system is to be generated, the user should read "Modification of the IEEVLNKT Load Module" in the section "CALL-OS Batch Interface Facility". The modified load module may be incorporated into the system during its generation.

OS/360 SYSTEM GENERATION REQUIREMENTS AND CONSIDERATIONS

This subsection explains the operands that must be included in the system generation macro instructions when an operating system is generated for the support of CALL-OS. Also mentioned are those operands which constitute a basic requirement for an operating system regardless of CALL-OS, but whose values should be considered in the light of CALL-OS. Only those macro instructions and operands directly related to CALL-OS are mentioned. For other macro instructions and operands required for operating system generation, see IBM System/360 Operating System: System Generation.

CTRLPROG MACRO INSTRUCTION

The MAXIO operand in the CTRLPROG macro instruction represents the maximum number of I/O operations that can be simultaneously processed by the operating system. IOS support for terminals and the 2314 or 2319 storage facility is required by the operating system to support CALL-OS. A recommended minimum value for this parameter should be N+N/10, where N is the number of lines supported, plus one for every four direct access devices on the system.

Note: The value specified in the MAXIO operand determines the number of request elements (RQEs) generated in the OS/360 operating system. If this number is too small, it is possible to exhaust the RQE queue, thereby causing unpredictable system errors.

With an MFT system, the amount of storage allocated to the system queue area is specified in the SYSQUE parameter. If COBI is to be used with CALL-OS, an additional amount must be specified in this parameter sufficient to allow two subtasks, as well as the normal system queue area requirements.

Because CALL-OS performs its own time slicing operations, it cannot be run in a task area for which time slicing has been specified. Therefore, care must be taken to ensure that the TMSLICE parameter is not specified either for the partition in which CALL-OS is run on an MFT system or for the priority under which CALL-OS is run on an MVT system.

IOCONTRL MACRO INSTRUCTION

IOCONTRL identifies to the operating system the type of transmission control unit (TCU) to be attached to a System/360 channel. Specify one IOCONTRL macro for each TCU to be operated under CALL-OS.

If the user wishes to specify an IBM 2702 TCU containing the 31-line expansion feature, a separate IOCONTRL macro must be coded for each of the two sets of lines.

IODEVICE MACRO INSTRUCTION

IODEVICE describes to the operating system the characteristics of an input/output device and its operating system requirements. For CALL-OS, IODEVICE identifies the type of terminal (IBM 2741 or TTY) that is connected to each line. The operands that are associated with this macro, and which are pertinent to CALL-OS, are discussed below:

UNIT=type — Specifies the type of terminal that is to communicate with the computer over the line address given by the ADDRESS operand. Valid parameters for terminals to be used with CALL-OS are 2741 and TWX.

ADDRESS=address — Specifies the three-digit address of the line over which the type of terminal given in the UNIT parameter is to communicate. Valid parameters are within the range 000-6FF, inclusive.

ADAPTER=type — Specifies the type of TCU terminal control and terminal adapter associated with the line address given in the ADDRESS parameter. Code one of the following values:

IBM1 for an IBM 2741 Communications Terminal communicating with an IBM 2702 or 2703 TCU through an IBM Terminal Adapter, Type I, and either (1) an appropriate data set, or (2) an IBM line adapter.

TELE2 for a TWX, Type 33 or 35, communicating with an IBM 2702 or 2703 TCU through a Telegraph Terminal Control, Type II, and a data set line adapter and an appropriate data set.

SETADDR=value — Specifies which of the three set address (SAD) commands is to be issued to the TCU (IBM 2702 only) for operations on the line specified by the ADDRESS parameter. The SAD command selects the appropriate line speed for the type of terminal connected to the line. The association between the specific command (SADZER, SADONE, or SADTWO) and the corresponding line speed is established by internal connection within the 2702. This is accomplished by the customer engineer when the 2702 is installed. This operand must be coded if the TCU to which the line is connected is an IBM 2702. When RPQ E54838 is installed, the original SAD command will continue to be specified (do not specify SAD 3).

RESMODS MACRO INSTRUCTION

The RESMODS macro instruction is used to add user-written routines, in load module form, to the nucleus library (SYS1.NUCLEUS) to be generated. Before these modules can be included in the nucleus library, they must be members of a partitioned data set. Since the CALL-OS Type I SVC is added to the operating system nucleus at system build time, the

user need not include the RESMODS macro instruction unless he plans to include the Type I SVC from the CALL-OS load module library during a subsequent system generation.

SCHEDULR MACRO INSTRUCTION

When COBI is used, the terminal user may transmit messages to the OS/360 system operator's console. If multiple console support is included in the system, he may use routing codes with the messages. This support is specified in the SCHEDULR and SECONSLE macro instructions during system generation.

SUPRVSOR MACRO INSTRUCTION

For CALL-OS support, the SUPRVSOR macro instruction must specify operands as follows:

The OPTIONS operand must specify PROTECT for both MFT and MVT systems:

```
OPTIONS=PROTECT
```

The OPTIONS operand must specify ATTACH for MFT systems only:

```
OPTIONS=(ATTACH,PROTECT)
```

The TIMER operand must specify INTERVAL: TIMER=INTERVAL

The TRACE operand is not required for CALL-OS. If the CALL-OS trace option is desired, the operating system generation must include the TRACE operand.

SVCTABLE MACRO INSTRUCTION

CALL-OS requires a Type I SVC routine. This routine is supplied as part of the basic program material from the IBM Program Information Department (PID), and is added to the System/360 Operating System during the CALL-OS system build procedure. The operating system is prepared for this SVC routine through the use of the SVCTABLE macro instruction.

UNITNAME MACRO INSTRUCTION

The UNITNAME macro instruction is used to assign a unique name to a collection of I/O devices assigned to specific addresses. The addresses must be the same as those specified in the IODEVICE macro instruction. Then this name can be used in the UNIT parameter of the appropriate DD statement. For example, all the teletype terminals in a system could be assigned to one named collection, all the 2741 correspondence to another, and so on. Or, if an IBM System 370 Model 145 is being used and 2319 devices are assigned to 2314, then all CALL-OS cataloged procedures which assign data sets to 2314 storage devices will operate without change.

In addition, when COBI is used, it is recommended that the devices upon which volumes with scannable output data sets will be mounted be assigned to a device class with the UNITNAME macro. The name of this class may then be referenced in the DD statements for the volumes. Scannable output data sets must reside on 2314 or 2319 disk storage. It is further recommended that SYSDA not be assigned to devices used for scannable output; there may be difficulty in varying them offline at

CALL-OS completion time because OS/360 system data sets may be placed there.

CALL-OS SYSTEM BUILD

System build for CALL-OS follows a simple process. This process has the following advantages:

- Minimizes operator intervention
- Reduces JCL and control statement requirements
- Facilitates the addition of CALL-OS routines to OS/360 libraries

The basis of the system build process is the CALL-OS system as released by IBM.

SYSTEM RELEASE TAPES

When the CALL-OS system is released, it may consist of up to a maximum of four release tapes: one tape for the executive and utility programs and one tape for each compiler requested by the installation. Each release tape contains several unloaded data sets produced by the OS/360 utility program IEHMOVE. These data sets contain source, macro and load module libraries for the component, and, in the case of the executive and utility release tape, two additional data sets contain IBM-supplied JCL procedures and the changes applied to the previous version, respectively.

The executive and utility release tape contains the following data sets:

- RTOSPROC — which contains the JCL procedures required to build a system, create a data base, and initialize COBI data sets, as well as standard module residency lists
- OSRTS.EXEC.MODLIB — which contains, in load module form, all the executive and utility modules necessary to run the system
- OSRTS.EXEC.MACLIB — which contains the executive and utility macros
- OSRTS.EXEC.SOURCE — which contains the executive and utility source modules
- OSRTS.EXEC.CHANGE — which contains the change cards applied to the previous version of CALL-OS to obtain this (the current) version of CALL-OS.

During the system build process, the procedure, macro, and load module libraries are loaded from the release tape. When system build is completed, the macro library and certain procedures may be deleted from the system (see Step I). The source library does not have to be loaded until updates are made to the system as described in the section "Maintaining the System". The change data set does not have to be loaded unless the installation has modified CALL-OS modules and wants to see the changes applied to those modules (see "System Build Considerations for an Installation-Modified System" at the end of this section).

The release tape for each compiler selected by the installation contains the following data sets:

- OSRTS.component.MODLIB
- OSRTS.component.SOURCE
- OSRTS.component.MACLIB

where component is a CALL-OS language name (BASIC, FORTRAN, or PLI). During system build, only the load module library for the compiler is loaded. The source and macro libraries are used to maintain the system and do not need to be loaded until updates are to be made to the compiler.

SYSTEM BUILD PROCESS SUMMARY

As illustrated in Figure 21, the system build process for a new system may consist of the following steps:

Step I — Execute the IEHMOVE utility to load the supplied executive and utility libraries from the release tape to a disk pack.

Step II — Execute the IEHMOVE utility to load the supplied compiler libraries.

Step III — Execute the RTOSJOB1 procedure to accomplish the following:

- Assemble the global table macro.
- Link-edit the CALL-OS nucleus with the assembled global table.
- Relink-edit the OS/360 nucleus with the CALL-OS SVC.
- Link-edit the CALL-OS device-dependent error routine into SYS1.SVCLIB.

Step IV — Build the CALL-OS data base. At this point in the process, the user can elect one of three options. He can (1) establish a new data base to conform with CALL-OS formats and structure, (2) rebuild an existing data base or connect an existing data base to his system, or (3) create and format a default data base. For option 1, see the description of U#UTIL1 in the section "Creating and Maintaining the Data Base". Options 2 and 3 are discussed under Step IV later in this subsection.

Step V — If desired, execute the U#UTIL5 utility to create JCL statements from the CALL-OS index. These statements form the base for the startup deck used to initialize the CALL-OS system.

Steps I through IV must be performed when a new CALL-OS system is built. If an existing system is being modified, Steps I and III are required, Step II is required only if a new version of a compiler used in the system is released, and Step IV is required only if a data base is to be built or recreated. Step V is optional during any system build.

The following directions should be read thoroughly before beginning actual machine operation to ensure that no prerequisite detail has been overlooked. Ten disk cylinders must be available for an online JOBLIB, one track in SYS1.PROCLIB for system build procedures, and one track in SYS1.SVCLIB for the device-dependent error routine.

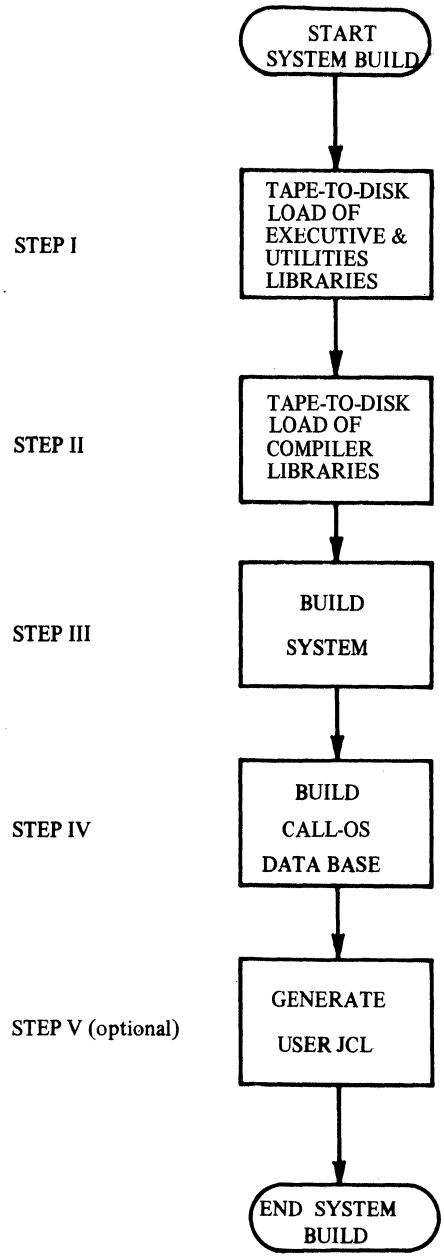


Figure 21. The system build process for a new system

STEP I - LOADING THE EXECUTIVE AND UTILITY LIBRARIES

Prior to executing the RTOSJOB1 procedure in Step III, the user must have satisfied the following requirements:

1. The library RTOSPROC must be merged with SYS1.PROCLIB. RTOSPROC contains the procedures listed in Table 4.

2. The library OSRTS.EXEC.MODLIB must be loaded from tape to disk, renamed to qualifier.JOBLIB, and cataloged (where qualifier is an index level qualifier selected by the user).

3. The library OSRTS.EXEC.MACLIB must be loaded from tape to disk, renamed to qualifier.MACLIB, and cataloged.

The IEHMOVE utility is used for all three steps. The following example shows the use of the utility to perform all Step I requirements:

```
//LOAD     JOB      ---
//MOVE     EXEC     PGM=IEHMOVE
//SYSPRINT DD       SYSOUT=A
//SYSUT1   DD       UNIT=2314,DISP=OLD,VOL=SER=scrvol
//DD1      DD       UNIT=2314,DISP=OLD,VOL=SER=volid1
//DD2      DD       UNIT=2314,DISP=OLD,VOL=SER=volid2
//TAPE1    DD       UNIT=2400,DISP=(OLD,PASS),VOL=SER=RTOSYS,
//                  LABEL=(,NL),DCB=(LRECL=80,BLKSIZE=800,
//                  RECFM=FB)
//SYSIN    DD       *
   COPY    PDS=RTOSPROC,TO=2314=volid2,FROM=2400=(RTOSYS,1),    X
               FROMDD=TAPE1
   COPY    PDS=RTOSPROC,TO=2314=volid1,FROM=2314=volid2,        X
               RENAME=SYS1.PROCLIB
   COPY    PDS=OSRTS.EXEC.MODLIB,TO=2314=volid2,                X
               FROM=2400=(RTOSYS,2),FROMDD=TAPE1,               X
               RENAME=qualifier.JOBLIB
   COPY    PDS=OSRTS.EXEC.MACLIB,TO=2314=volid2,                X
               FROM=2400=(RTOSYS,3),FROMDD=TAPE1,               X
               RENAME=qualifier.MACLIB
/*
//BLDXCAT  EXEC     PGM=IEHPROGM
//SYSPRINT DD       SYSOUT=A
//DD1      DD       DISP=OLD,VOL=SER=volid2,UNIT=2314
//SYSIN    DD       *
   BLDX    INDEX=qualifier
   CATLG   DSNAME=qualifier.JOBLIB,VOL=2314=volid2
   CATLG   DSNAME=qualifier.MACLIB,VOL=2314=volid2
/*
```

Table 4. Contents of RTOSPROC

When Used	Members	Use
System Build (See note)	RTOSJOB1	Procedure to build CALL-OS
Data Base Build (See note)	RTOSDB01	Procedure to build one-pack default data base
	RTOSDB02	Procedure to build two-pack default data base
	RTOSDB03	Procedure to build three-pack default data base
COBI Initialization	COBIBLD	Procedure to initialize COBI data sets and link edit the COBI reader and writer modules into SYS1.LINKLIB (see the section on COBI)
	DIBCBINC	Used in second step of COBIBLD procedure
System Initialization Module Residency Lists	RTOSALL	Module list for totally resident system
	RTOSLLRS	Module list for load, list, run, save functions
	RTOSNONE	Module list for nonresident system
	RTOSUSER	Module list for user terminal commands
Note: The system and data base build procedures may be deleted from SYS1.PROCLIB after CALL-OS has been built.		

where

qualifier — is the index level qualifier chosen by the user for CALL-OS data sets; the default is OSRTS

scrvol — is the volume identification of a scratch volume

volid1 — is the volume identification of volume on which SYS1.PROCLIB resides

volid2 — is the volume identification of volume on which the user desires qualifier.JOBLIB and qualifier.MACLIB to reside

STEP II - LOADING THE COMPILER LIBRARIES

The optional compiler libraries OSRTS.PLI.MODLIB, OSRTS.BASIC.MODLIB, and OSRTS.FORTRAN.MODLIB must be merged with qualifier.JOBLIB (see the example that follows).

The following example illustrates the copying of the load module library for all three language compilers:

```
//COMPUPDT JOB      ---
//MOVE     EXEC     PGM=IEHMOVE
//SYSPRINT DD       SYSOUT=A
//SYSUT1   DD       DISP=OLD,UNIT=2314,VOL=SER=scrvol
//DD1      DD       DISP=OLD,UNIT=2314,VOL=SER=volid1
//BASTAPE  DD       DISP=OLD,UNIT=2400,VOL=SER=BASIC,LABEL=(,NL),
//                  DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//FORTAPE  DD       DISP=OLD,UNIT=2400,VOL=SER=FORT,LABEL=(,NL),
//                  DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//PLITAPE  DD       DISP=OLD,UNIT=2400,VOL=SER=PLI,LABEL=(,NL),
//                  DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSIN    DD       *
   COPY    PDS=OSRTS.BASIC.MODLIB,TO=2314=volid1,                X
               FROM=2400=(BASIC,1),FROMDD=BASTAPE,               X
               RENAME=qualifier.JOBLIB
   COPY    PDS=OSRTS.FORTRAN.MODLIB,TO=2314=volid1,              X
               FROM=2400=(FORT,1),FROMDD=FORTAPE,                X
               RENAME=qualifier.JOBLIB
   COPY    PDS=OSRTS.PLI.MODLIB,TO=2314=volid1,                  X
               FROM=2400=(PLI,1),FROMDD=PLITAPE,                 X
               RENAME=qualifier.JOBLIB
/*
```

where

qualifier	is the index level qualifier chosen by the user for CALL-OS data sets; the default is OSRTS
volid1	is the volume identification of volume on which qualifier.JOBLIB resides
scrvol	is the volume identification of a scratch volume

Note: The DD statements for compilers which are not to be supported must be deleted when this step is executed.

STEP III - LINK EDITING THE SYSTEM

Step III of the system build process link edits the system. For convenience, a cataloged procedure has been provided to perform this link edit. This procedure is named RTOSJOB1 and resides in SYS1.PROCLIB after completion of Step I of the system build process. The procedure consists of the following steps:

- Step S01 Causes execution of U#UTIL2, a CALL-OS utility program which produces assembler and linkage editor control statements used in the subsequent steps of the procedure
- Step S02 Causes the assembly of the CALL-OS global table macro for the purpose of making user-selected options available to the CALL-OS system
- Step S03 Causes the link edit of the CALL-OS nucleus with the previously-assembled global table
- Step S04 Causes the link edit of an OS/360 nucleus with the CALL-OS Type I SVC routine
- Step S05 Causes the link edit of the device-dependent error routine into the system SVC library (SYS1.SVCLIB)

Figure 22 shows the JCL statements in the procedure. The following JCL is required to execute the procedure:

```
//SYSBLD    JOB     ---
//JOBLIB    DD      DSN=qualifier.JOBLIB,DISP=OLD
//          EXEC    RTOSJOB1,QA=qualifier,
//                  PARM.S01='user-specified options'
```

where

qualifier	is the index level qualifier for CALL-OS data sets chosen by the user. The default is OSRTS and is used in examples in other portions of this manual.
user-specified options	indicates information passed to the CALL-OS U#UTIL2 utility program. These options are used in the building of control statements and are described in greater detail in the following text.

```
//RTOSJOB1 PROC    QA=OSRTS
//S01      EXEC    PGM=U#UTIL2,COND=(0,NE) ** UPDATE CONTROL CARDS *
//PROCCC03 DD      UNIT=SYSDA,DISP=(,PASS),DSN=&CC03,
//                 SPACE=(TRK,1),DCB=(BLKSIZE=80,LRECL=80,RECFM=FB)
//PROCCC04 DD      UNIT=SYSDA,DISP=(,PASS),DSN=&CC04,
//                 SPACE=(TRK,1),DCB=(BLKSIZE=80,LRECL=80,RECFM=FB)
//PROCCC08 DD      UNIT=SYSDA,DISP=(,PASS),DSN=&CC08,
//                 SPACE=(TRK,1),DCB=(BLKSIZE=80,LRECL=80,RECFM=FB)
//GTABOPTS DD      DSN=&GTABMAC,DISP=(,PASS),UNIT=SYSDA,
//                 SPACE=(TRK,1),DCB=(BLKSIZE=80,LRECL=80)
//SYSUDUMP DD      SYSOUT=A
//SYSPRINT DD      SYSOUT=A
//S02      EXEC    PGM=IEUASM,PARM=LOAD,REGION=80K,  ** ASM GTAB **
//                 COND=(0,NE)
//SYSUT1   DD      UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSUT2   DD      UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSUT3   DD      UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSLIB   DD      DSN=&QA..MACLIB,DISP=OLD
//SYSIN    DD      DSN=&GTABMAC,DISP=(OLD,DELETE)
//SYSG0    DD      DSN=&OBJMOD,UNIT=SYSDA,SPACE=(TRK,4),DISP=(,PASS)
//SYSPUNCH DD      DUMMY
//SYSPRINT DD      SYSOUT=A
//S03      EXEC    PGM=IEWL,PARM='XREF,LIST,LET,NCAL',
//                 COND=(4,LT),REGION=96K       ** L/E BASESYS **
//SYSLIB   DD      DSN=&OBJMOD,DISP=(OLD,DELETE)
//SYSLMOD  DD      DSN=&QA..JOBLIB,DISP=(OLD,PASS)
//SYSLIN   DD      DSN=&CC04,DISP=(OLD,DELETE)
//SYSUT1   DD      UNIT=SYSDA,SPACE=(1024,(200,20))
//SYSPRINT DD      SYSOUT=A
//S04      EXEC    PGM=IEWL,PARM='XREF,LIST,LET,NCAL,DC,SCTR',
//                 COND=(4,LT),REGION=96K       **L/E  SVC **
//SYSLIB   DD      DSN=&QA..JOBLIB,DISP=(OLD,PASS)
//SYSLMOD  DD      DSN=SYS1.NUCLEUS,DISP=OLD
//SYSLIN   DD      DSN=&CC03,DISP=(OLD,DELETE)
//SYSUT1   DD      UNIT=SYSDA,SPACE=(1024,(200,20))
//SYSPRINT DD      SYSOUT=A
//S05      EXEC    PGM=IEWL,PARM='XREF,LIST,LET,NCAL',
//                 COND=(4,LT),REGION=96K       ** L/E I/ORTN **
//SYSLIB   DD      DSN=&QA..JOBLIB,DISP=(OLD,PASS)
//SYSLMOD  DD      DSN=SYS1.SVCLIB,DISP=OLD
//SYSLIN   DD      DSN=&CC08,DISP=(OLD,DELETE)
//SYSUT1   DD      UNIT=SYSDA,SPACE=(1024,(200,20))
//SYSPRINT DD      SYSOUT=A
```

Figure 22. JCL statements in the RTOSJOB1 procedure

User-Specified Options

When the cataloged procedure is executed, the user supplies optional information in the parameter field of the EXEC statement. These options are used in the first step of the catalog procedure to produce the control statements used in subsequent steps. The parameter field has the following format:

```
PARM.S01='SVC=nnn,NUC=(x,y),ERRNO=mmm,TYPE=aaa,COBI=bbb'
```

where

nnn	is the number assigned to the CALL-OS SVC during system generation and must be in the range 200 through 255. If this parameter is omitted, the default is 255.
x	identifies the OS/360 nucleus (IEANUC0x) to which the CALL-OS SVC load module is to be added and must be in the range 1 through 9. If this parameter is omitted, the default is 1.
y	identifies the OS/360 nucleus (IEANUC0y) to be created and must be in the range 1 through 9. If this parameter is omitted, the default is 2.
mmm	is the number to be assigned to the device-dependent error routine and must be in the range 220 through 229. If this parameter is omitted, the default is 229.
aaa	specifies the type of the OS/360 system under which CALL-OS is to be run and must be either MFT or MVT. If this parameter is omitted, the default is MFT.
bbb	specifies whether or not the COBI modules are to be part of the CALL-OS system and must be either YES or NO. If this parameter is omitted, the default is NO.

Note: If the NUC Parameter is specified, both the x and y parameters must be supplied.

Table 5 shows the parameter defaults for RTOSJOB1.

The following example shows the specification of user-specified options for use with the cataloged procedure:

```
PARM.S01='SVC=246,NUC=(,1,6),ERRNO=227'
```

This results in an SVC number of 246 being assigned to the CALL-OS Type I SVC and a number of 227 being assigned to the error routine. The OS/360 nucleus to be used as input is IEANUC01 and the resulting nucleus is to be named IEANUC06. The OS/360 system is an MFT system and the CALL-OS system is not to contain COBI.

Table 5. Parameter defaults for the RTOSJOB1 procedure

Parameter	Use	Default
COBI	Specifies whether or not the resident COBI modules are to be included in the CALL-OS nucleus	NO
ERRNO	Specifies number to be assigned to the device-dependent error routine	229
NUC	Specifies both the old OS/360 nucleus, to which the CALL-OS SVC is to be added, and the new OS/360 nucleus, to be created	old nucleus: 1 new nucleus: 2
QA	Specifies the high level index qualifier to be used for CALL-OS data sets	OSRTS
SVC	Specifies the number to be assigned to the CALL-OS Type I SVC	255
TYPE	Specifies the type of OS/360 system under which CALL-OS is to be run	MFT

Subsequent Processing

The first step of the cataloged procedure is the execution of the CALL-OS utility program U#UTIL2. This utility builds four sets of statements which are placed in temporary sequential data sets. Three of these sets are used as control statement input to the linkage editor to produce the OS/360 nucleus containing the CALL-OS Type I SVC, the CALL-OS nucleus, and the CALL-OS terminal device-dependent error routine. The fourth set of statements is used as source input to the OS/360 assembler to produce the CALL-OS global table. The user supplies the information necessary to build these statements with the parameters described previously.

OS/360 Nucleus - Linkage Editor Control Statements: The first set of statements produced by U#UTIL2 is placed in the temporary data set defined by the PROCCC03 DD statement. These statements are the linkage editor control statements used to link edit the CALL-OS SVC into either an OS/360 MFT or an OS/360 MVT nucleus. The statements produced depend on the OS/360 system specified and are as follows:

```
INSERT     IEAANIP0            For MFT
INSERT     IEAAIH00
CHANGE     IGC255(IGCnnn)
INCLUDE    SYSLIB(IGC255)
INCLUDE    SYSLMOD(IEANUC0x)
NAME       IEANUC0y(R)

      or

INSERT     IEAANIP0            For MVT
INSERT     IEAQFX00
CHANGE     IGC255(IGCnnn)
INCLUDE    SYSLIB(IGC255V)
INCLUDE    SYSLMOD(IEANUC0x)
NAME       IEANUC0y(R)
```

where

nnn is either the SVC number supplied with the SVC parameter or a default of 255

x is either the from nucleus number supplied with the NUC parameter or a default of 1

y is either the to nucleus number supplied with the NUC parameter or a default of 2

This data set is used as input to a step which link edits an existing copy of the OS/360 nucleus into the same or new copy. During this link edit, the CALL-OS Type I SVC is included in the OS/360 nucleus and given the number specified by the user.

CALL-OS Nucleus - Linkage Editor Control Statements: The second set of statements produced by U#UTIL2 is placed in the temporary data set defined by the PROCCC04 DD statement. These statements are the linkage editor control statements used to link edit the CALL-OS nucleus. The statements produced depend on the OS/360 system specified (MFT or MVT) and whether COBI is used or not. The appropriate control statements are as follows:

```
 INCLUDE    SYSLIB                For MFT
 INCLUDE    SYSLMOD(C#CPID)
 INCLUDE    SYSLMOD(RTOS1)
[INCLUDE    SYSLMOD(C#IOREQ,C#NOTFY,M#QIOR,M#CBIO,M#VTBL,S#INDXQ,
                   S#PARM,S#QNOT)]
 ENTRY      N#LINIT
 NAME       RTOS1(R)
```

or

```
 INCLUDE    SYSLIB                For MVT
 INCLUDE    SYSLMOD(C#CPIDV)
 INCLUDE    SYSLMOD(RTOS1)
[INCLUDE    SYSLMOD(C#IOREQ,C#NOTFY,M#QIOR,M#CBIO,M#VTBL,S#INDXQ,
                   S#PARM,S#QNOT)]
 ENTRY      N#LINIT
 NAME       RTOS1(R)
```

The INCLUDE statement enclosed in brackets is present in the data set only if COBI=YES is specified. This data set is used as input to the step which link edits the CALL-OS nucleus. This nucleus is tailored to suit the needs of the installation as specified with parameter information.

CALL-OS Error Routine - Linkage Editor Control Statements: The third set of control statements produced by U#UTIL2 is placed in the temporary data set defined by the PROCCC08 DD statement. These statements are the linkage editor control statements used to link edit the CALL-OS device dependent error routine. The statements produced assign the user-specified number to the routine and place the routine in the SVC library (SYS1.SVCLIB). The control statements are as follows:

```
 CHANGE     IGEDUMMY (IGE00nnn)
 INCLUDE    SYSLIB(IGEDUMMY)
 NAME       IGE00nnn(R)
```

where

nnn is either the user-desired number assigned to the device-dependent error routine or a default of 229.

Global Table Assembly Control Statements: The fourth set of statements produced by U#UTIL2 is placed in the temporary data set defined by the GTABOPTS DD statement. These statements are assembler source statements used in the assembly of the CALL-OS global table macro. The statements are as follows:

```
Z#GTAB     CSECT=YES,SVC=nnn,ERRNO=mmm,OSTYPE=aaa,COBI=bbb
END
```

where

nnn — is either the user-specified SVC number or a default of 255

mmm — is either the user-specified error routine number or a default of 229

aaa — specifies the type of OS/360 system, either MFT or MVT, with a default of MFT

bbb — specifies whether or not COBI is to be added to the system, either YES or NO, with a default of NO

STEP IV - ESTABLISHING THE DATA BASE

To use the CALL-OS product, which includes the executive system, language facilities, and offline utility programs, the customer must establish his data base (directories, catalogs, file and program space, etc.) to conform to CALL-OS formats and structure. In practice, the customer's programming staff prepares the appropriate JCL to allocate all necessary data sets, and to invoke the proper CALL-OS utilities to format those allocated data sets. This is the way a system should be installed if system performance and disk storage are to be properly considered, but as a convenience, the system build package can create and format for the customer a default data base on from one to three completely dedicated packs. This provides the shortest path to installation, but does not necessarily provide the best system performance. The user should be cognizant of the data base philosophy, before commencing data base build, if he does not accept one of the default data bases.

System Build with Existing Data Base

If it should become necessary to rebuild the system, the CALL-OS index and the OS/360 catalog must be updated. As an example, assume that a new release of CALL-OS takes place, and that the user does not desire to make any changes to the existing data base. In this case, the following steps should be taken:

1. Scratch and uncatalog the JOBLIB data set.
2. Scratch the CALL-OS members of SYS1.PROCLIB which were copied into SYS1.PROCLIB during Step I of the previous sytem build (see Table 1).
3. Run Steps I and II of system build to copy the release libraries to disk, copy the new procedures into SYS1.PROCLIB, and copy the language compilers into the JOBLIB.
4. Run Step III of system build to generate the new system.
5. Run program U#UTIL1 specifying the COMPILER function to replace an old compiler with a new compiler supplied on the release tape. The DD statement should specify DISP=OLD for each compiler. The index is updated only with the length of the new compiler.

Should new space for a compiler be required, uncatalog and scratch the existing space in step 1 above, and reallocate and catalog the new area in step 5 above.

The means by which to reestablish the index and/or to move the data base are covered in the next examples.

Assuming that you are generating CALL-OS for the first time, but wish to connect an existing data base which is to remain on the same pack(s) but which is not cataloged (that is, new OS/360 SYSGEN or generating a unique OS/360 and CALL-OS), the following steps may be followed:

1. Run steps I, II, and III of system build to generate CALL-OS.

2. Run U#UTIL1 for compilers, using previous compiler data sets (DISP=OLD) to update the index and obtain the new versions of the compilers.

Given the circumstances above, with the exception of operating the data base on different packs, the following steps could be taken:

1. Run steps I, II, and III of system build to generate the system.

2. Run U#UTIL1 for all functions (that is, overlay, swap, system group, user group, compiler), to allocate, format, and validate space on the new packs, add the entries to the index data set, and catalog the data sets.

3. Move the data base to the newly allocated space, using the copy function of the OS/360 utility IEHMOVE. The REORGANIZE function of the data base utility DIBCADBU may also be used to move the data set and regain purged space.

Specific customer requirements vary from installation to installation; however, the examples and guidelines above should prove helpful in solving the problems of system rebuild and/or using an existing data base.

System Build with Default Data Base

Three default data base options are provided: one pack, two packs, and three packs. The user provides volume identifications and the number of lines to be supported. The packs should have been initialized and be free of flagged tracks in the central cylinders. The VTOC for these packs must reside on cylinder zero, and not extend past cylinder zero, track 18. Absolute track allocation is used in these procedures. Therefore, the packs may not have any space allocated before data base build.

The central cylinders start at cylinder 81. To find the last central cylinder (LCC) used, use the following formulas:

one pack		LCC=81+9+lines supported
two packs	pack 1	LCC=81+5+lines on this pack
	pack 2	LCC=81+4+lines on this pack
three packs	pack 1	LCC=81+2+lines on this pack
	pack 2	LCC=81+2+lines on this pack
	pack 3	LCC=81+5+lines on this pack

Note that if U#UTIL1 is not in SYS1.LINKLIB, a JOBLIB statement must be inserted in the JCL pointing to the user library which contains U#UTIL1.

For compiler runs of U#UTIL1, a task area of 150K is required. For MFT, the data base procedures require a minimum task area of 150K. For MVT, the task area size is specified on the default data base procedures.

Note that all language compilers supplied with the system are allocated one cylinder of disk space during Step 4 of the default data base build. In addition, an attempt is made to convert all supplied compilers to fast-load-and-go format.

For each compiler not ordered with the system, and, therefore, not to be supported in this data base, an error message is generated by the utility U#UTIL1 during Step 4 of the default data base build. The format of this message is as follows:

```
nnnnnn
***COMPILER - BAD BLDL ON COMPILER NAME
```

where

nnnnnn is the compiler name that could not be found in JOBLIB.

This message should be ignored for language compilers which are not to be supported. However, any other messages encountered during this step must be further investigated.

Default Data Base on One Pack

For one pack, punch and execute the following statements:

```
//DB01      JOB     ---
//          EXEC    RTOSDB01,VOL1=volid1,LINES1=number,
//                  QA=qualifier
```

where

volid	is the volume identification of disk pack on which the data base is to be built.
number	is the maximum number of lines to be supported (not exceeding 99); this determines the number of cylinders for work/swap space allocation.
qualifier	is the index level qualifier chosen by the user for CALL-OS data sets; the default is OSRTS.

These statements cause the RTOSDB01 procedure to be executed. This procedure contains the JCL statements shown in Figure 23; it builds a data base on one pack as shown in Figure 24.

```
//          PROC    QA=OSRTS
//STEP0     EXEC    PGM=U#UTIL3                    ** FORMAT INDEX **
//INDEX     DD      DSN=&QA..INDEX,DISP=(,PASS),VOL=SER=&VOL1,
//                  UNIT=2314,SPACE=(ABSTR,(1,19)),
//                  DCB=(BLKSIZE=7294,LRECL=7294)
//SYSPRINT  DD      SYSOUT=A
//STEP1     EXEC    PGM=U#UTIL1,PARM='USRGROUP',COND=(0,NE)
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//SYSPRINT  DD      SYSOUT=A
//AAAZZZ00  DD      DSN=&QA..AAAZZZ00,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(1600,20)),
//                  DCB=DSORG=DA
//STEP2     EXEC    PGM=U#UTIL1,PARM='OVERLAY',COND=(0,NE)
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//SYSPRINT  DD      SYSOUT=A
//OVLY      DD      DSN=&QA..OVLY,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1620))
//STEP3     EXEC    PGM=U#UTIL1,PARM='SYSGROUP',COND=(0,NE)
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//SYSGRP00  DD      DSN=&QA..SYSGRP00,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(80,1640)),
//                  DCB=DSORG=DA
```

Figure 23. JCL statements in RTOSDB01 procedure (part 1 of 2)

```
//STEP4     EXEC    PGM=U#UTIL1,PARM='COMPILER',COND=(0,NE),
//                  REGION=150K
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//LANG      DD      DSN=&QA..JOBLIB,DISP=OLD
//FORTRAN   DD      DSN=&QA..FORTRAN,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1720))
//PL2       DD      DSN=&QA..PL2,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1740))
//PLI       DD      DSN=&QA..PLI,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1760))
//BASIC     DD      DSN=&QA..BASIC,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1780))
//STEP5     EXEC    PGM=U#UTIL1,PARM='WORKSWAP'
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//SWAP00    DD      DSN=&QA..SWAP00,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,&LINES1.)
//STEP6     EXEC    PGM=U#UTIL1,PARM='USRGROUP',
//                  COND=((0,NE,STEP0),(0,NE,STEP1),(0,NE,STEP2),
//                  (0,NE,STEP3),(0,NE,STEP5))
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//AAAZZZ01  DD      DSN=&QA..AAAZZZ01,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,(10),,MXIG),DCB=DSORG=DA
//STEP7     EXEC    PGM=IEHPROGM,
//                  COND=((0,NE,STEP0),(0,NE,STEP1),(0,NE,STEP2),
//                  (0,NE,STEP3),(0,NE,STEP5),(0,NE,STEP6))
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,CATLG)
//UGRP1     DD      DSN=*.STEP1.AAAZZZ00,DISP=(OLD,CATLG)
//          DD      DSN=*.STEP6.AAAZZZ01,DISP=(OLD,CATLG)
//OVLAY     DD      DSN=*.STEP2.OVLY,DISP=(OLD,CATLG)
//SYSGRP    DD      DSN=*.STEP3.SYSGRP00,DISP=(OLD,CATLG)
//COMP1     DD      DSN=*.STEP4.FORTRAN,DISP=(OLD,CATLG)
//COMP2     DD      DSN=*.STEP4.PL2,DISP=(OLD,CATLG)
//COMP3     DD      DSN=*.STEP4.PLI,DISP=(OLD,CATLG)
//COMP4     DD      DSN=*.STEP4.BASIC,DISP=(OLD,CATLG)
//SWAP      DD      DSN=*.STEP5.SWAP00,DISP=(OLD,CATLG)
//SYSIN     DD      DUMMY
```

Figure 23. JCL statements in RTOSDB01 procedure (part 2 of 2)

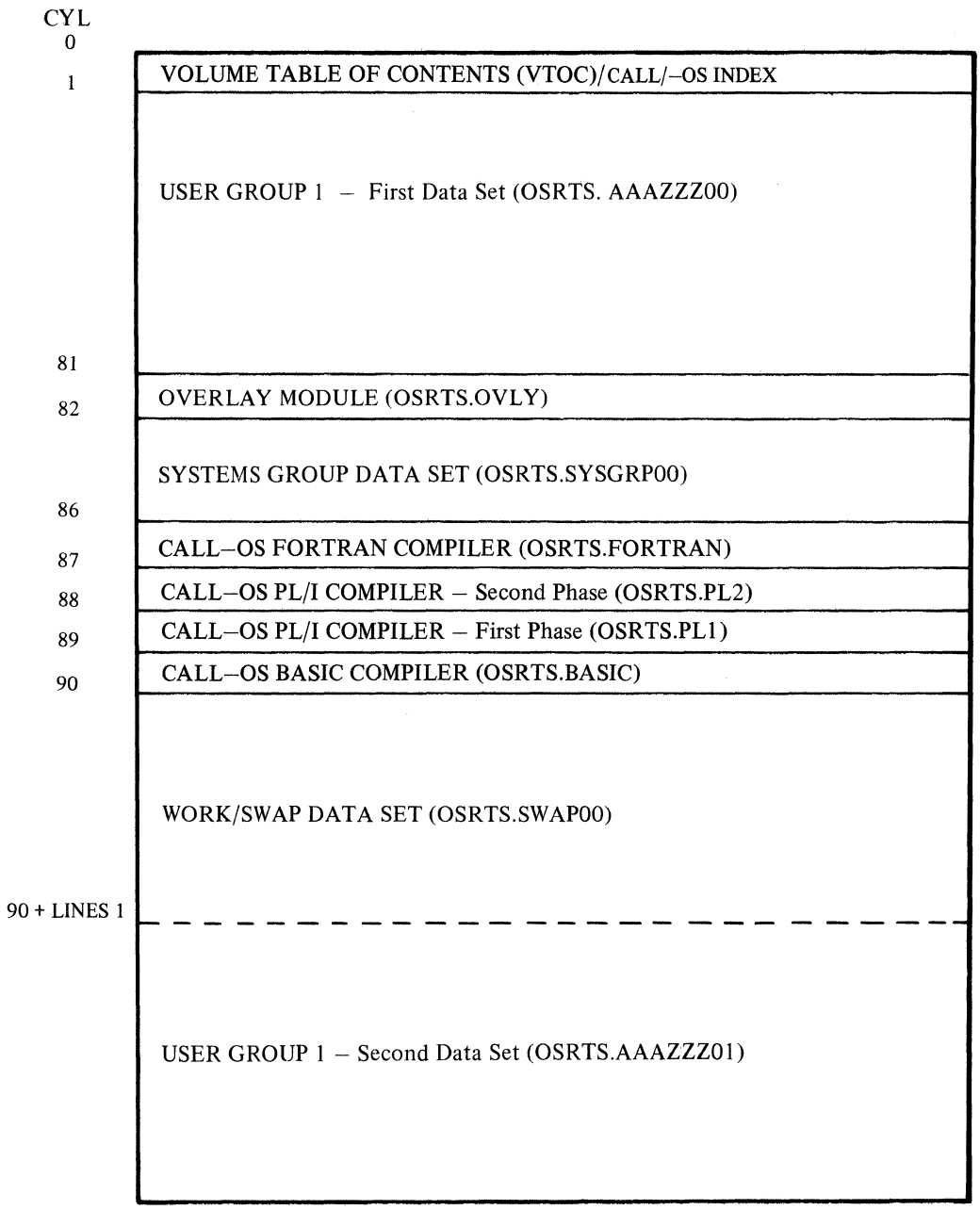


Figure 24. Default data base option 1 - single pack

Default Data Base on Two Packs

For two packs, punch and execute the following statements:

```
//DB02      JOB     ---
//          EXEC    RTOSDB02,VOL1=volid1,VOL2=volid2,LINES1=num1,
//                  LINES2=num2,QA=qualifier
```

where

volid1 volid2	is the volume identification of the packs on which the data base is to be built
num1 num2	specify the total number (num1 and num2) of lines to be supported. Split the lines as evenly as possible between the two packs; for example, if 21 lines are to be supported, then: LINES1=11,LINES2=10. num1 may not exceed 103. num2 may not exceed 104.
qualifier	is the index level qualifier chosen by the user for CALL-OS data sets; the default is OSRTS.

These statements cause the RTOSDB02 procedure to be executed. This procedure contains the JCL statements shown in Figure 25; it builds a data base on two packs as shown in Figure 26.

```
//          PROC    QA=OSRTS
//STEP0     EXEC    PGM=U#UTIL3                    ** FORMAT INDEX **
//INDEX     DD      DSN=&QA..INDEX,DISP=(,PASS),VOL=SER=&VOL1,
//                  UNIT=2314,SPACE=(ABSTR,(1,19)),
//                  DCB=(BLKSIZE=7294,LRECL=7294)
//SYSPRINT  DD      SYSOUT=A
//STEP1     EXEC    PGM=U#UTIL1,PARM='USRGROUP',COND=(0,NE)
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//SYSPRINT  DD      SYSOUT=A
//AAAMZZ00  DD      DSN=&QA..AAAMZZ00,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(1600,20)),
//                  DCB=DSORG=DA
//NAAZZZ00  DD      DSN=&QA..NAAZZZ00,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(1600,20)),
//                  DCB=DSORG=DA
//STEP2     EXEC    PGM=U#UTIL1,PARM='OVERLAY',COND=(0,NE)
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//OVLY      DD      DSN=&QA..OVLY,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1620))
//STEP3     EXEC    PGM=U#UTIL1,PARM='SYSGROUP',COND=(0,NE)
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//SYSGRP00  DD      DSN=&QA..SYSGRP00,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(80,1620)),
//                  DCB=DSORG=DA
//STEP4     EXEC    PGM=U#UTIL1,PARM='COMPILER',COND=(0,NE),
//                  REGION=150K
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//LANG      DD      DSN=&QA..JOBLIB,DISP=OLD
//FORTRAN   DD      DSN=&QA..FORTRAN,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1640))
//PL2       DD      DSN=&QA..PL2,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1660))
//PLI       DD      DSN=&QA..PLI,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1680))
//BASIC     DD      DSN=&QA..BASIC,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1700))
```

Figure 25. JCL statements in the RTOSDB02 procedure (part 1 of 2)

```
//STEP5     EXEC    PGM=U#UTIL1,PARM='WORKSWAP'
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//SWAP00    DD      DSN=&QA..SWAP00,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,&LINES1.)
//SWAP01    DD      DSN=&QA..SWAP01,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,&LINES2.)
//STEP6     EXEC    PGM=U#UTIL1,PARM='USRGROUP',
//                  COND=((0,NE,STEP0),(0,NE,STEP1),(0,NE,STEP2),
//                  (0,NE,STEP3),(0,NE,STEP5))
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//AAAMZZ01  DD      DSN=&QA..AAAMZZ01,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,(10),,MXIG),
//                  DCB=DSORG=DA
//NAAZZZ01  DD      DSN=&QA..NAAZZZ01,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,(10),,MXIG),DCB=DSORG=DA
//STEP7     EXEC    PGM=IEHPROGM,
//                  COND=((0,NE,STEP0),(0,NE,STEP1),(0,NE,STEP2),
//                  (0,NE,STEP3),(0,NE,STEP5),(0,NE,STEP6))
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,CATLG)
//UGRP1     DD      DSN=*.STEP1.AAAMZZ00,DISP=(old,CATLG)
//          DD      DSN=*.STEP6.AAAMZZ01,DISP=(OLD,CATLG)
//UGRP2     DD      DSN=*.STEP1.NAAZZZ00,DISP=(OLD,CATLG)
//          DD      DSN=*.STEP6.NAAZZZ01,DISP=(OLD,CATLG)
//OVLAY     DD      DSN=*.STEP2.OVLY,DISP=(OLD,CATLG)
//SYSGRP    DD      DSN=*.STEP3.SYSGRP00,DISP=(OLD,CATLG)
//COMP1     DD      DSN=*.STEP4.FORTRAN,DISP=(OLD,CATLG)
//COMP2     DD      DSN=*.STEP4.PL2,DISP=(OLD,CATLG)
//COMP3     DD      DSN=*.STEP4.PLI,DISP=(OLD,CATLG)
//COMP4     DD      DSN=*.STEP4.BASIC,DISP=(OLD,CATLG)
//SWAP      DD      DSN=*.STEP5.SWAP00,DISP=(OLD,CATLG)
//          DD      DSN=*.STEP5.SWAP01,DISP=(OLD,CATLG)
//SYSIN     DD      DUMMY
```

Figure 25. JCL statements in the RTOSDB02 procedure (part 2 of 2)

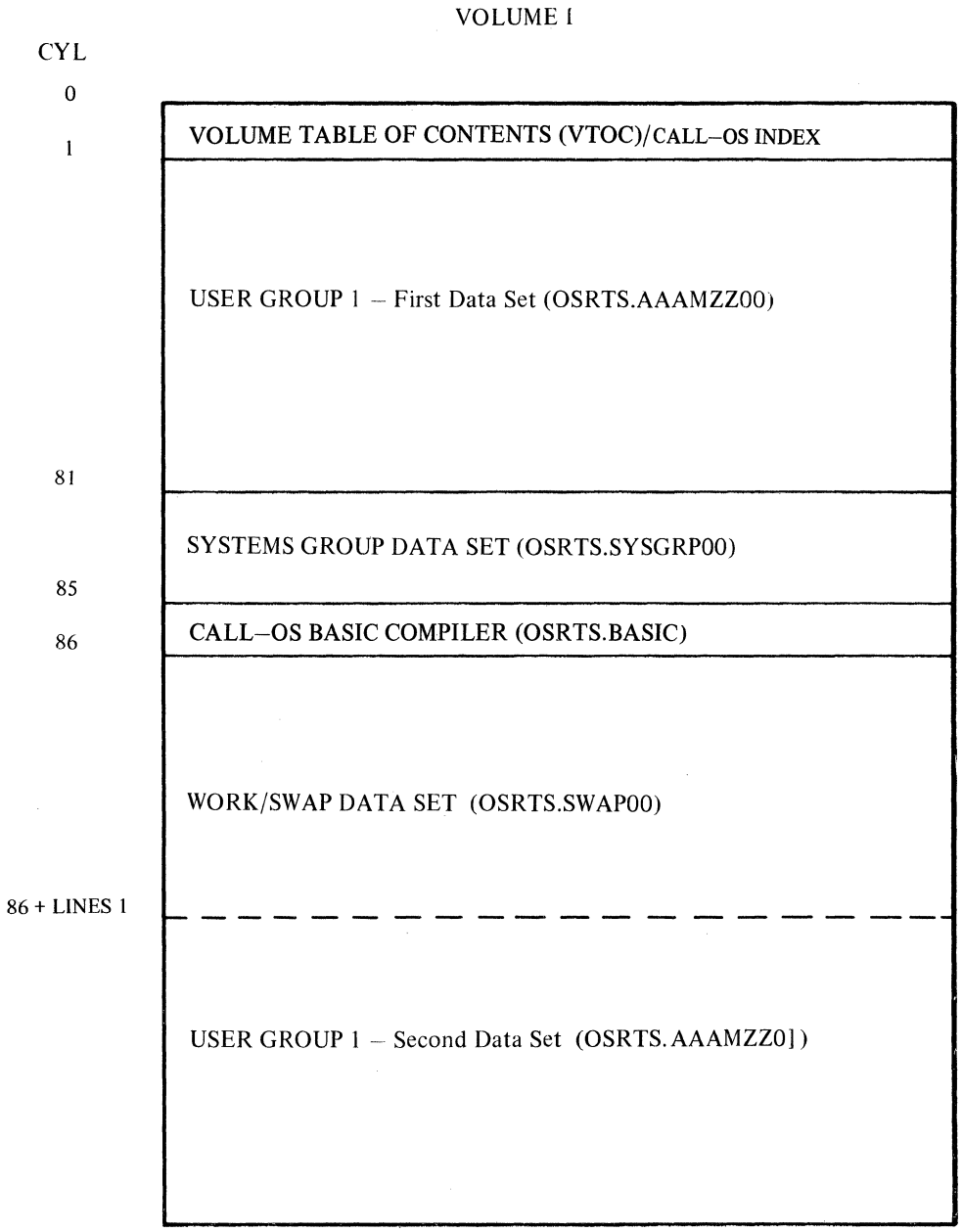


Figure 26. Default data base option 2 - two packs (part 1 of 2)

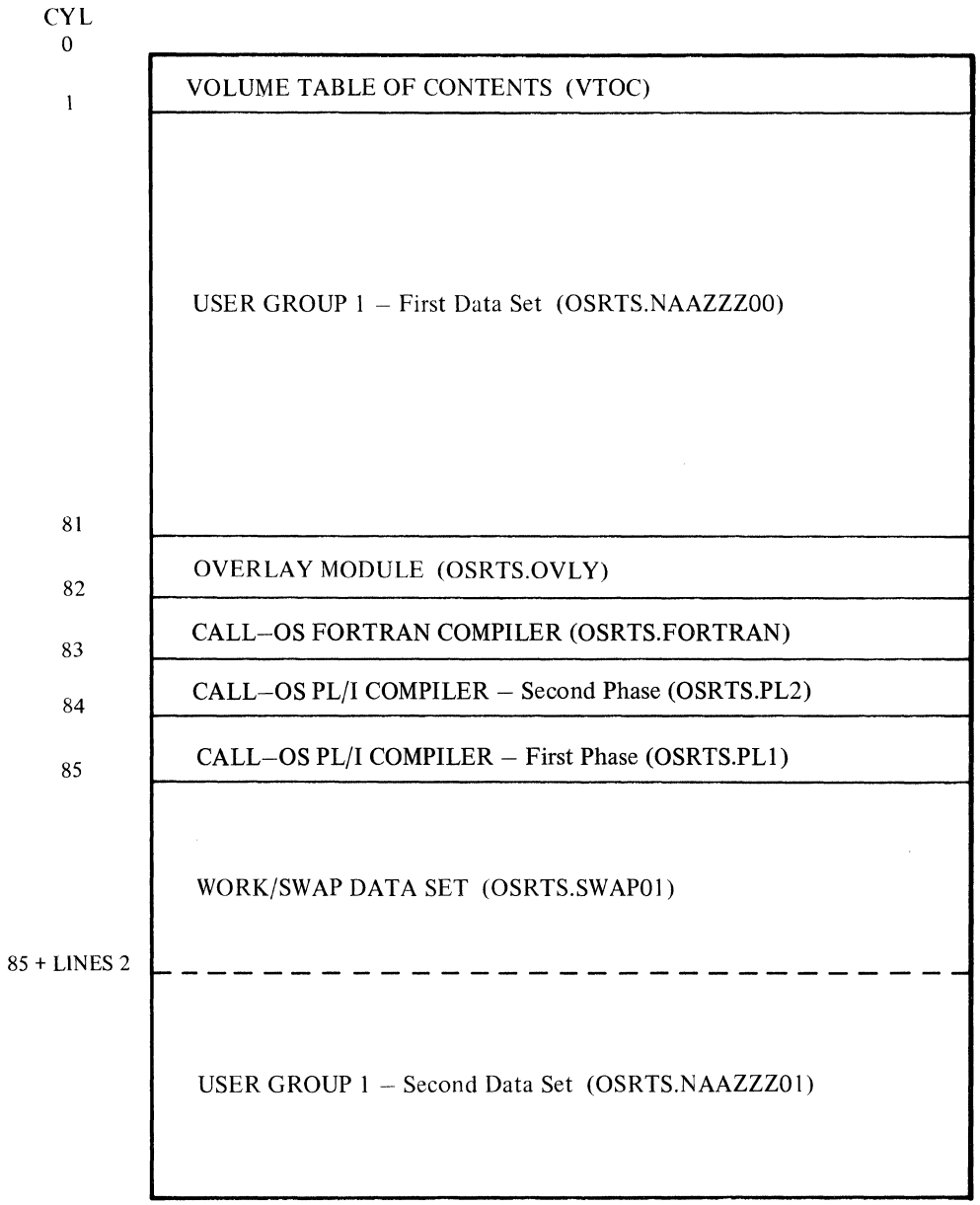


Figure 26. Default data base option 2 - two packs (part 2 of 2)

Default Data Base on Three Packs

For three packs, punch and execute the following statements:

```
//DB03      JOB     ---
//          EXEC    RTOSDB03,VOL1=volid1,VOL2=volid2,VOL3=volid3,
//                  LINES1=num1,LINES2=num2,LINES3=num3,
//                  QA=qualifier
```

where

volid1 volid2 volid3	is the volume identification of packs on which the data base is to be built
num1 num2 num3	specify the total number (num1, num2, and num3) of lines to be supported. Divide the lines as evenly as possible across the packs; for example, if 31 lines are to be supported, then: LINES1=10, LINES2=10, LINES3=11. num1 may not exceed 106. num2 may not exceed 106. num3 may not exceed 103.
qualifier	is the index level qualifier chosen by the user for CALL-OS data sets; the default is OSRTS.

These statements cause the RTOSDB03 procedure to be executed. This procedure contains the JCL statements shown in Figure 27; it builds a data base on three packs as shown in Figure 28.

```
//          PROC    QA=OSRTS
//STEP0     EXEC    PGM=U#UTIL3                    ** FORMAT INDEX **
//INDEX     DD      DSN=&QA..INDEX,DISP=(,PASS),VOL=SER=&VOL1,
//                  UNIT=2314,SPACE=(ABSTR,(1,19)),
//                  DCB=(BLKSIZE=7294,LRECL=7294)
//SYSPRINT  DD      SYSOUT=A
//STEP1     EXEC    PGM=U#UTIL1,PARM='USRGROUP',COND=(0,NE)
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//SYSPRINT  DD      SYSOUT=A
//AAAIZZ00  DD      DSN=&QA..AAAIZZ00,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(1600,20)),
//                  DCB=DSORG=DA
//JAARZZ00  DD      DSN=&QA..JAARZZ00,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(1600,20)),
//                  DCB=DSORG=DA
//SAAZZZ00  DD      DSN=&QA..SAAZZZ00,VOL=SER=&VOL3,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(1600,20)),
//                  DCB=DSORG=DA
//STEP2     EXEC    PGM=U#UTIL1,PARM='OVERLAY',COND=(0,NE)
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//OVLY      DD      DSN=&QA..OVLY,VOL=SER=&VOL3,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1620))
//STEP3     EXEC    PGM=U#UTIL1,PARM='SYSGROUP',COND=(0,NE)
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//SYSGRP00  DD      DSN=&QA..SYSGRP00,VOL=SER=&VOL3,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(80,1640)),
//                  DCB=DSORG=DA
//STEP4     EXEC    PGM=U#UTIL1,PARM='COMPILER',COND=(0,NE),
//                  REGION=150K
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//LANG      DD      DSN=&QA..JOBLIB,DISP=OLD
//FORTRAN   DD      DSN=&QA..FORTRAN,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1620))
//PL2       DD      DSN=&QA..PL2,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1620))
//PLI       DD      DSN=&QA..PLI,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1640))
//BASIC     DD      DSN=&QA..BASIC,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(ABSTR,(20,1640))
```

Figure 27. JCL statements in the RTOSDB03 procedure (part 1 of 2)

```
//STEP5     EXEC    PGM=U#UTIL1,PARM='WORKSWAP'
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//SWAP00    DD      DSN=&QA..SWAP00,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,&LINES1.)
//SWAP01    DD      DSN=&QA..SWAP01,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,&LINES2.)
//SWAP02    DD      DSN=&QA..SWAP02,VOL=SER=&VOL3,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,&LINES3.)
//STEP6     EXEC    PGM=U#UTIL1,PARM='USRGROUP',
//                  COND=((0,NE,STEP0),(0,NE,STEP1),(0,NE,STEP2),
//                  (0,NE,STEP3),(0,NE,STEP5))
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,PASS)
//AAAIZZ01  DD      DSN=&QA..AAAIZZ01,VOL=SER=&VOL1,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,(10),,MXIG),
//                  DCB=DSORG=DA
//JAARZZ01  DD      DSN=&QA..JAARZZ01,VOL=SER=&VOL2,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,(10),,MXIG),
//                  DCB=DSORG=DA
//SAAZZZ01  DD      DSN=&QA..SAAZZZ01,VOL=SER=&VOL3,UNIT=2314,
//                  DISP=(,PASS),SPACE=(CYL,(10),,MXIG),
//                  DCB=DSORG=DA
//STEP7     EXEC    PGM=IEHPROGM,
//                  COND=((0,NE,STEP0),(0,NE,STEP1),(0,NE,STEP2),
//                  (0,NE,STEP3),(0,NE,STEP5),(0,NE,STEP6))
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSN=&QA..INDEX,DISP=(OLD,CATLG)
//UGRP1     DD      DSN=*.STEP1.AAAIZZ00,DISP=(OLD,CATLG)
//          DD      DSN=*.STEP6.AAAIZZ01,DISP=(OLD,CATLG)
//UGRP2     DD      DSN=*.STEP1.JAARZZ00,DISP=(OLD,CATLG)
//          DD      DSN=*.STEP6.JAARZZ01,DISP=(OLD,CATLG)
//UGRP3     DD      DSN=*.STEP1.SAAZZZ00,DISP=(OLD,CATLG)
//          DD      DSN=*.STEP6.SAAZZZ01,DISP=(OLD,CATLG)
//OVLAY     DD      DSN=*.STEP2.OVLY,DISP=(OLD,CATLG)
//SYSGRP00  DD      DSN=*.STEP3.SYSGRP00,DISP=(OLD,CATLG)
//COMP1     DD      DSN=*.STEP4.FORTRAN,DISP=(OLD,CATLG)
//COMP2     DD      DSN=*.STEP4.PL2,DISP=(OLD,CATLG)
//COMP3     DD      DSN=*.STEP4.PLI,DISP=(OLD,CATLG)
//COMP4     DD      DSN=*.STEP4.BASIC,DISP=(OLD,CATLG)
//SWAP      DD      DSN=*.STEP5.SWAP00,DISP=(OLD,CATLG)
//          DD      DSN=*.STEP5.SWAP01,DISP=(OLD,CATLG)
//          DD      DSN=*.STEP5.SWAP02,DISP=(OLD,CATLG)
//SYSIN     DD      DUMMY
```

Figure 27. JCL statements in the RTOSDB03 procedure (part 2 of 2)

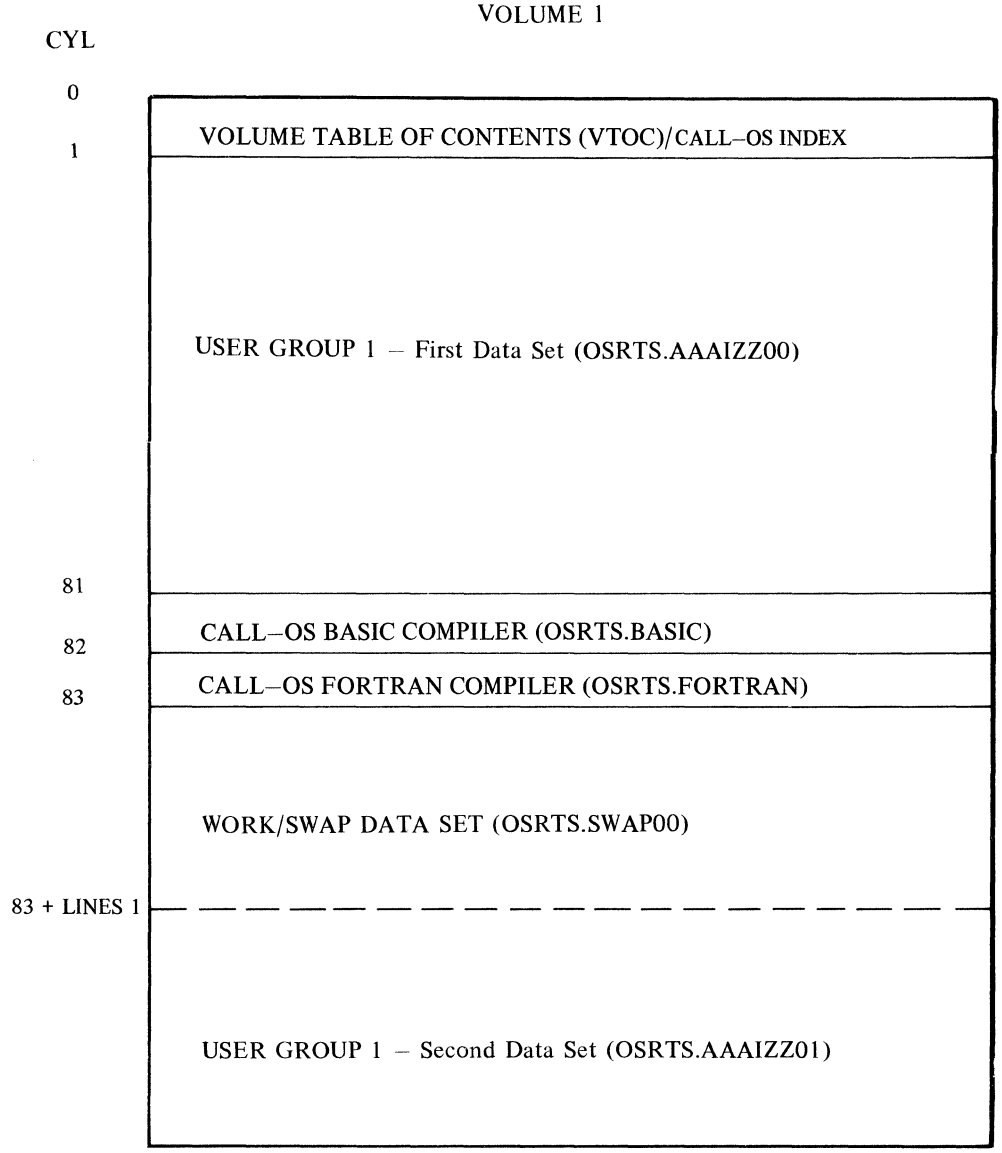


Figure 28. Default data base option 3 - three packs (part 1 of 3)

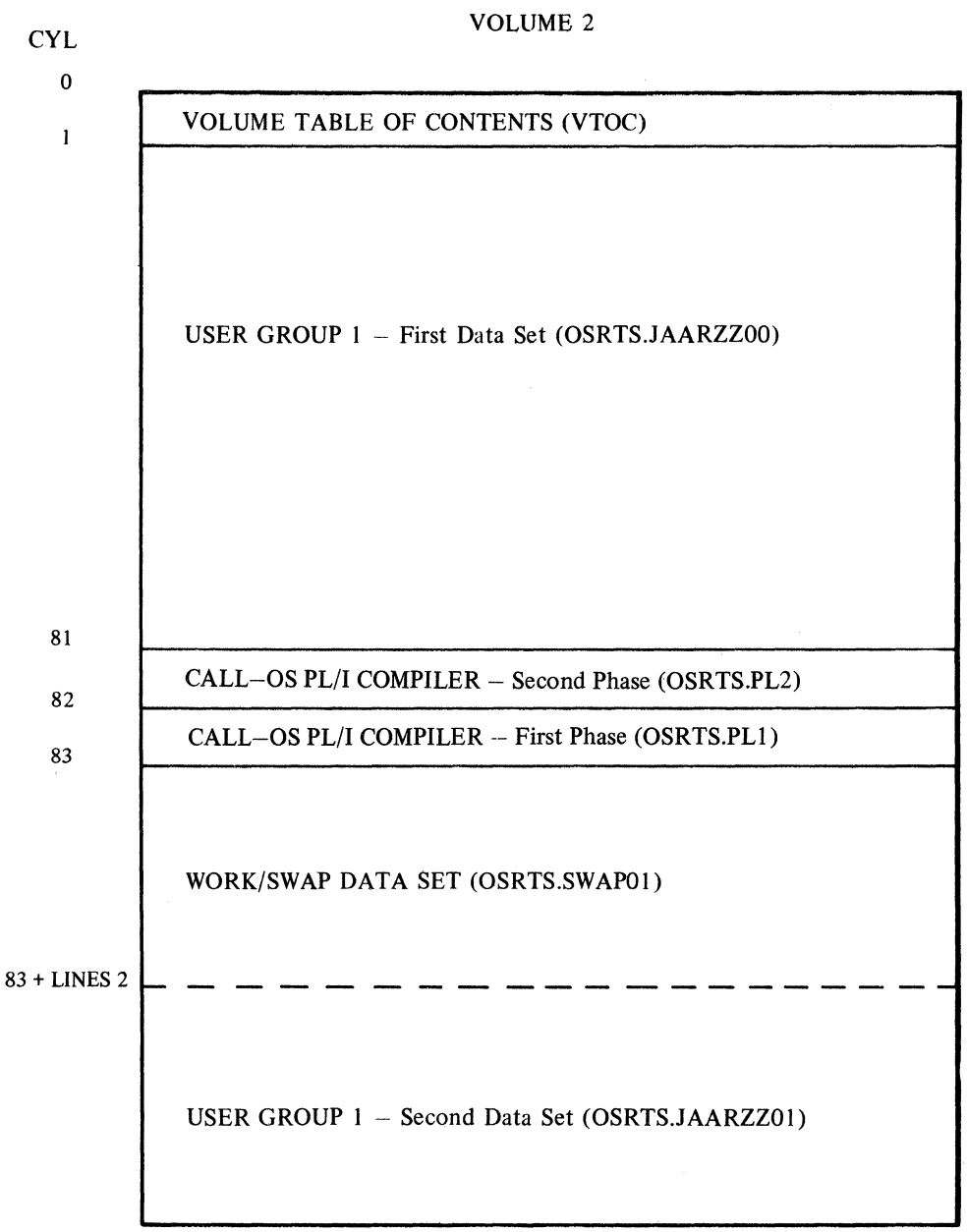


Figure 28. Default data base option 3 - three packs (part 2 of 3)

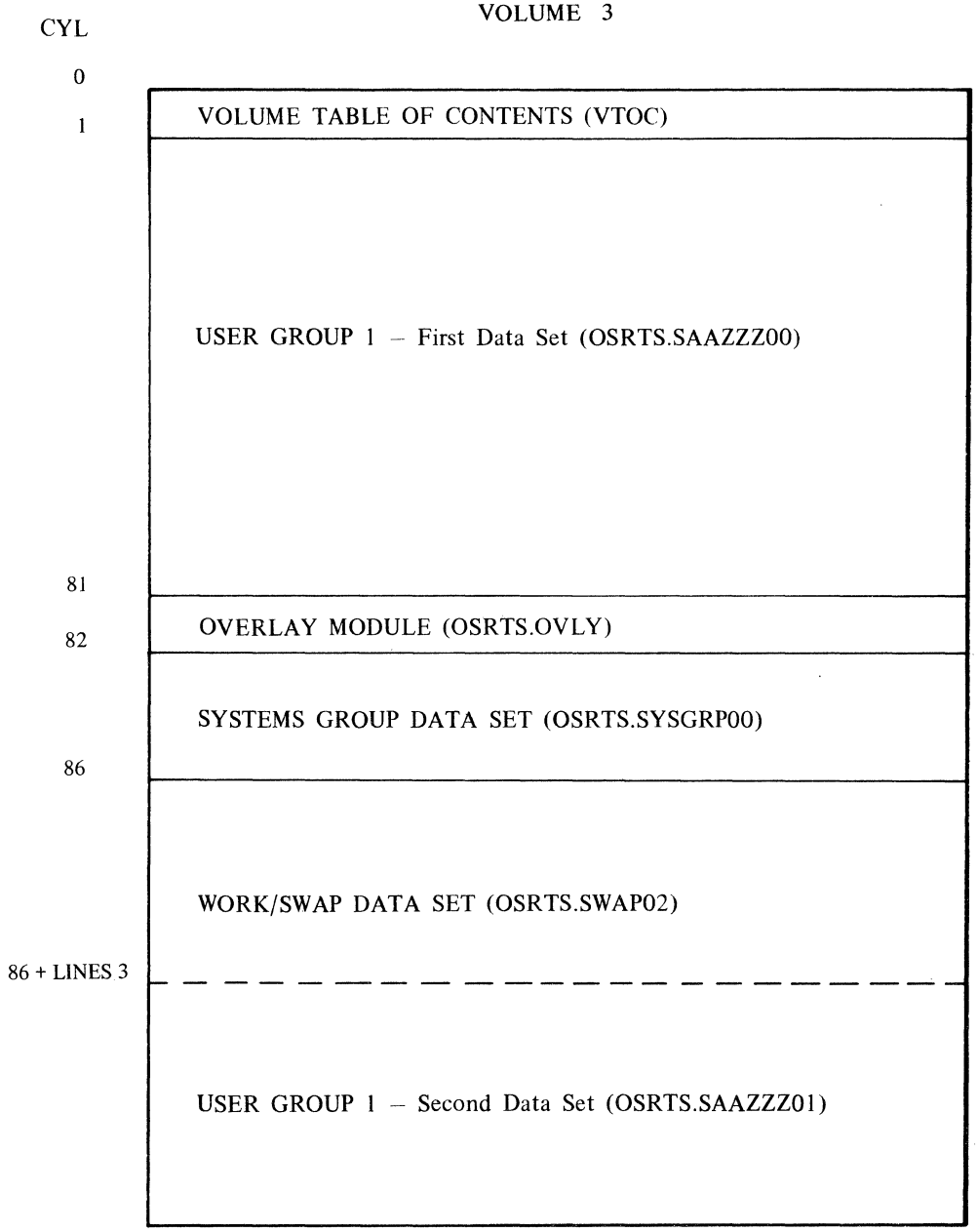


Figure 28. Default data base option 3 - three packs (part 3 of 3)

Restarting The Default Data Base

In the establishment of a default data base, certain conditions (such as miscalculation of user requirements, abnormal termination, etc.) may result in an incorrect definition of the data base and, furthermore, require a restart of the default data base build procedure. To recover from such a situation, the default data base procedures are restartable.

STEP V - PUNCHING THE STARTUP DECK (OPTIONAL)

This part of the system build process may be omitted if the startup deck has been prepared in some other way. For example, if an existing deck is used or the deck is punched by hand. The utility is U#UTIL5, which uses the information in the index to create the deck. The following JCL is required:

```
//SYSBLD3    JOB    ---
//JOBLIB     DD     DSN=OSRTS.JOBLIB,DISP=SHR
//STEP       EXEC   PGM=U#UTIL5
//INDEX      DD     DSN=OSRTS.INDEX,DISP=OLD
//CARD       DD     SYSOUT=B
//SYSABEND   DD     SYSOUT=A
//SYSPRINT   DD     SYSOUT=A
```

For an example of U#UTIL5 output, see the section "Initializing the System."

SYSTEM BUILD CONSIDERATIONS FOR AN INSTALLATION-MODIFIED SYSTEM

If an installation has modified CALL-OS modules, it may want to examine the changes made to the previous IBM version of CALL-OS to produce the current version. The change data set on the executive and utility release tape contains the changes to those modules which were not completely rewritten. This data set is loaded from tape to disk in the same way the source and macro libraries are loaded. If desired, the IEHMOVE control statements to load the change data set may be added to Step I of system build.

Once loaded, the change data set becomes a partitioned data set. The member named INFO contains the names of the new modules added to the current version, as well as the names of the modules from the previous version which were completely rewritten. The rest of the members contain the IEBUPDTE change cards that were applied to modules which were modified, one member for each modified module.

The following JCL can be used to print member INFO:

```
//PRINT        JOB
//             EXEC    PGM=IEBPTPCH
//SYSPRINT     DD      SYSOUT=A
//SYSUT1       DD      DSN=OSRTS.EXEC.CHANGE(INFO),DISP=OLD,
//                     UNIT=2314,VOL=SER=volid
//SYSUT2       DD      SYSOUT=A
//SYSIN        DD      *
   PRINT      TYPORG=PS,MAXFLDS=1
   RECORD FIELD=(80)
/*
```

where

volid is the volume serial number of the volume which contains the change data set

This same JCL, with minor modifications, may be used to punch out the change cards for a module. These modifications are:

1. On the SYSUT1 DD statement, change the member name from INFO to the name of the module

2. On the SYSUT2 DD statement, change SYSOUT=A to SYSOUT=B

3. On the first utility control statement, change PRINT to PUNCH

INITIALIZING THE SYSTEM

It has already been pointed out that system build refers to the initial establishment of CALL-OS to be run under the control of OS/360. System initialization is that portion of CALL-OS which tailors the CALL-OS time-sharing system to meet the user's run-time requirements. This function is carried out immediately after OS/360 gives control to the CALL-OS job, and just before the enabling of user terminals for online operations.

SYSTEM INITIALIZATION

The CALL-OS system operates as a task under control of the OS/360 system, and as such, is entered into the OS/360 system as part of the job stream. The CALL-OS job is defined by the installation system programmer in the form of a startup deck, which causes the CALL-OS system to be initialized and put into operation. CALL-OS initialization determines the system environment required to support CALL-OS at a particular installation. The system environment consists of the execution characteristics desired and the data set configuration required for the current session.

The execution characteristics affect total system performance and control the functions available during this session of CALL-OS. These characteristics are specified by initialization options in the parameter field of the EXEC statement in the startup deck. These options specify the number of buffers in the system, the logical line numbers for the system consoles, the maximum size of new data files, the time slice values assigned to new and old jobs as well as compilers, which portions of the system are to reside in hierarchy storage, and COBI operating information. During initialization of CALL-OS, the parameters are used to tailor the system for execution during this session only. The execution characteristics of the system may be altered each time the system is initialized.

The data set configuration provides the disk space required for system operation and determines the number of user groups supported for the current session. The entire configuration is defined by the DD statements in the startup deck. Required statements define the index, the system group data sets, the work/swap data sets, and if COBI is used, the COBI storage requirements as well. Other DD statements determine the type of terminals supported, the number of lines to be enabled, the user groups to be allowed access to the system, the compilers to be used, and the modules to be resident for this session. By varying the optional DD statements, a variety of system configurations and available facilities are possible.

The rest of this section describes the startup deck in detail.

STARTUP DECK

The startup deck to be used to initiate CALL-OS may be punched either by hand or by using the U#UTIL5 utility. If the utility is used, there must be at least one qualifier in the data set names, and the following modifications must be made to the statements in the output deck:

1. The JOB statement must be completed or replaced.

2. The EXEC statement must be completed with the appropriate parameter information.

3. For more than one data set qualifier, the JOBLIB and INDEX DD statements must be corrected.

4. The TWX, T2741, and T2741E DD statements must be supplied to define the line configuration.

5. Any DD statements not desired should be removed (for example, the DD statements for the alternate cluster).

6. If COBI is to be used, DD statements must be supplied to define the OS/360 system job queue, COBI index, JCL data set, and the input data sets; in addition, if scanning is to be permitted, one or more volumes for scannable data sets must be defined.

Figure 29 shows the JCL statements that may be used in a CALL-OS startup deck. In the figure, required information on each statement is shown in uppercase letters and, along with special characters, must be punched as shown. Optional information is shown in lowercase letters and is supplied in accordance with the system to be initialized. The following sections contain detailed descriptions of the parameter information which may be supplied on the EXEC statement and the JCL statements in the startup deck.

Note: CALL-OS monitors its own time sharing and should not be run under OS/360 time sharing facilities. If, however, CALL-OS is initialized either in a time sharing partition for MFT or under a time sharing priority for MVT, an error message is issued and initialization terminates.

```
//CALLOS        JOB     MSGLEVEL=1,CLASS=b
//JOBLIB        DD      DSN=OSRTS.JOBLIB,DISP=SHR
//CALL          EXEC    PGM=RTOS1,ROLL=(NO,NO),PARM=(option1,
//                      option2,etc')
//SYSABEND      DD      SYSOUT=A
//SYSPRINT      DD      SYSOUT=A
//INDEX         DD      DSN=OSRTS.INDEX,DISP=SHR
//RESMODS       DD      DSN=SYS1.PROCLIB(membername),DISP=SHR
//OVLY          DD      DSN=OSRTS.OVLY,DISP=OLD
//BASIC         DD      DSN=OSRTS.BASIC,DISP=OLD
//FORTRAN       DD      DSN=OSRTS.FORTRAN,DISP=OLD
//PLI           DD      DSN=OSRTS.PLI,DISP=OLD
//PL2           DD      DSN=OSRTS.PL2,DISP=OLD
//SWAPnn        DD      DSN=OSRTS.SWAPnn,DISP=OLD
//SYSGRPnn      DD      DSN=OSRTS.SYSGRPnn,DISP=OLD
//aaabbbnn      DD      DSN=OSRTS.aaabbbnn,DISP=OLD
   .
   .
   .
//yyyzzzn       DD      DSN=OSRTS.yyyzzznn,DISP=OLD
//TWX           DD      UNIT=(generic name or unit address)
//T2741         DD      UNIT=(generic name or unit address)
//T2741E        DD      UNIT=(generic name or unit address)
//SYSJOBQ       DD      DSN=SYS1.SYSJOBQE,DISP=SHR
//CBNDX         DD      DSN=OSRTS.CBNDX,DISP=OLD
//CBJCL         DD      DSN=OSRTS.CBJCL,DISP=OLD
//CBSYSINA      DD      DSN=OSRTS.CBSYSINA,DISP=SHR
//CBSYSINB      DD      DSN=OSRTS.CBSYSINB,DISP=SHR
//SCANxx        DD      VOL=SER=yyyyyy,DISP=SHR,UNIT=2314
//SYSIN         DD      *

   Insert parameter information here; this information and the SYSIN
   DD statement are required only when more than 100 characters of
   parameter information are specified.

/*
```

Figure 29. JCL statements present in a CALL-OS startup deck

DESCRIPTION OF INITIALIZATION PARAMETERS

The parameter field on the EXEC statement and/or the SYSIN DD statement allows the user to specify a number of options available for execution. The options are divided into two groups: overall system options, which apply to the entire system, and additional options, which are used when the COBI facility is used. Table 6 shows the defaults for the system and COBI options in alphabetical order; those parameters not listed have no default.

It may be necessary to exceed the 100-character limitation on the length of this field. Under such circumstances, the additional parameter information may be included by using the SYSIN DD statement. Parameter information may be supplied with either or both methods, as long as no more than 400 characters are supplied altogether.

Table 6. Parameter defaults for RTOS1

Parameter	Use	Default
ACTIME	Specifies the time interval which is to elapse between accounting checkpoints	30 minutes
ALOCTYPE	Specifies whether the space allocation for user-defined scannable output data sets created by COBI jobs is in tracks or cylinders	Cylinder allocation
CBCLASS	Specifies the output class for the JCL and unscannable SYSOUT data sets for COBI jobs	Class Z
COMCON	Specifies the logical line number for the communications console	Logical line 2
COMTSL	Specifies the time slice allotted to each compiler or compiler phase	BASIC: one second FORTRAN: two secs. PLI,PL2: two secs.
DFLINK	Specifies the maximum number of half tracks allowed for a data file	100 half tracks
DSPACE	Specifies the space allocation for user-defined scannable data sets created by COBI jobs	One for primary and one for secondary if ALOCTYPE=CYL; ten otherwise
IPBUFS	Specifies the number of pots to be allocated for each line	Four pots per line with a minimum of 60 total
OPBUFS	Specifies the total number of 256-byte buffers to be allocated	One for every three lines with a minimum of five total
MAXDCB	Specifies the number of users permitted to scan data sets at the same time	One for every ten lines with a minimum of two total
OSCLASS	Specifies the output class to be used to return COBI JCL and data sets to OS/360 for processing	Class A
RDRQTY	Specifies the maximum number of jobs to be submitted before the COBI input data sets are switched	Ten jobs
RDRTIM	Specifies the number of minutes to elapse before the COBI input data sets are switched	15 minutes
RUNTSL	Specifies the time slice allotted to new and old jobs	New jobs: three secs. Old jobs: ten secs.
SHRTSL	Specifies the increment of time used to share time between user program area jobs and background	One second

Table 6. Parameter defaults for RTOS1 (continued)

Parameter	Use	Default
SYSCON	Specifies the logical line number for one or both command consoles	Primary: line 1 Alternate: line 0
UNITNM	Specifies the direct access unit name to be used for scannable output data sets	2314

Overall System Options

These options apply to the overall operation of the entire system. If any change is to be made in the specifications, the system must be reinitialized. The options which may be specified and their associated parameters are:

- The logical line number for the communication console - COMCON parameter
- The logical line number for one or both command consoles - SYSCON parameter
- The time slice allotted to new and old jobs - RUNTSL parameter
- The increment of time used to share time between user program area jobs and background processing - SHRTSL parameter
- The time slice alloted to each compiler or compiler phase - COMTSL
- The time interval to elapse between accounting checkpoints - ACTIME parameter
- The portions of the CALL-OS system which are to be loaded into hierarchy storage - LSCRES parameter
- The maximum number of half tracks allowed for a data file - DFLINK parameter
- The total number of 256-byte buffers to be allocated - OPBUFS parameter
- The number of 24-byte buffers (pots) to be allocated for each line - IPBUFS parameter
- The absence of the sort buffer - NOSORT parameter
- The absence of CALL-OS trace table entries - NOTRACE parameter

Appropriate defaults are assigned when a parameter is omitted. The parameters, their formats, and the defaults are described in the following text in alphabetical order by parameter.

ACTIME=nnn — User accounting information is updated on disk whenever a user initiates or terminates a terminal session. In addition, periodic checkpoints are taken during the terminal session to avoid loss of billing data in the event of a system failure. The number of minutes between checkpoints is specified with the ACTIME parameter, where nnn must be an integer. The default is 30 minutes between checkpoints.

COMCON=nnn

This is used to enter the communications console logical line number. The default is logical line number 2.

COMCON=0 signifies that no communications console is to be allocated, and that CALL-OS error messages are to be printed on the OS/360 system operator's console.

(A logical line number is defined as the order in which terminals dedicated to CALL-OS are assigned a UTT. See also TWX, T2741, and T2741E DD statement descriptions.)

COMTSL=(nnn,lname)

Each compiler or compiler phase in the CALL-OS system has a default time slice associated with it. To override these values, one COMTSL parameter must be included for each value to be modified,

where

nnn represents the new time slice value
lname indicates the name of the compiler or compiler phase, and must be BASIC, FORTRAN, PLI, or PL2

The minimum time slice for any compiler is one second; the maximum is ten seconds.

Note that for PL/I, an additional COMTSL parameter must be specified for the second phase. The default values provided with the system are one second for BASIC and three seconds each for FORTRAN, PLI, and PL2. These times are the times recommended for the IBM System/360 Model 50 and should be modified if another CPU is used. For example, on a Model 65 and above, possible values are one second for all compilers and compiler phases.

DFLINK=nnn

Each time a user either creates a new data file or updates an old data file, he may optionally specify the number of half tracks (links) he wishes to associate with his file. The DFLINK parameter determines the maximum number of links which can be requested. The nnn value may range from 4 to 100, inclusive. The default is 100 links.

IPBUFS=nnn

Specifies the number of 24-byte buffers (pots) to be allocated for each terminal line. The default is four buffers per terminal line with a minimum total of 60 buffers. If the total number of buffers in the system is less than four times the number of terminal lines, then the total is increased, either to four times the number of terminal lines or to the minimum of 60. The maximum is 15 pots per line, or the total of 60 if the number of lines is less than or equal to four.

LCSRES=aaaaaaaa

Specifies that the indicated portions of the CALL-OS system are to be located in Hierarchy 1 storage. Only those areas indicated by a letter code reside in Hierarchy 1 storage. The letters permitted and their meanings are as follows:

Letter	Area
B	256-byte buffers
C	Compiler area
D	Data control blocks (DCB)
G	COBI tables and bit strings
J	New job area and old job area
O	Overlay buffer (see note)
P	Pots (24-byte buffers) and terminal translate tables
S	Sort buffer
U	User terminal tables (UTT)

Extreme care should be exercised in the use of this option. Some considerations for its usage are discussed in the section "Designing the System."

Note: If LCS is available, an overlay buffer should not be needed since an all-resident system is clearly desirable.

NOSORT

If this parameter is present, the dedicated sort buffer is not included at job initialization and sorting takes place in the user program area. If the parameter is not present, the sort buffer is included at initialization time.

Exercise of this option should be considered with a minimal CALL-OS system supporting a small number of users. The 14,400 bytes required for the sort buffer would be available to run larger programs.

NOTRACE

If the CALL-OS customer includes the trace table option in his OS/360 system generation (SUPRVSOR macro), CALL-OS trace entries become optionally available in the same table. If these entries are not desired, NOTRACE should be coded. The default condition of no tracing is assumed if the OS/360 option is excluded at system generation time.

OPBUFS=nnn

Specifies the total number of 256-byte buffers in the system. The default is one buffer for every three terminal lines with a minimum of five buffers. A number of buffers less than one-third the number of terminal lines is increased, either to one-third the number of terminal lines or to a minimum of five buffers. The maximum is one buffer per line, or the default total of five if the number of lines is less than or equal to five.

RUNTSL=(nnn,mmm)

Specifies time-slice values for new(nnn) and old(mmm) problem program jobs. These values are used to determine the amount of CPU time a new or old job should have before being swapped out. However, the total elapsed time a job is in core is based on the amount of time given to background; this is because the time slice is allocated in increments rather than all at once (see SHRTSL). This parameter is used solely for CALL-OS, and is not to be confused with OS/360 task area time-slicing. The range allowed is 0.5 through 5.0 seconds for a new job and 1 through 20 seconds for an old job.

The default condition for the system is (3,10), which provides a three-second time slice for a new job and a ten-second time slice for an old job.

Note that the times given are the times for the IBM System/360 Model 50 and should be modified if another CPU is used. For example, a possible value for the Model 65 could be (1,3).

Note: For these and other time-slice values, the user can specify time to within one tenth of a second. Examples of acceptable time-slice values are: 0.5, 1.6, 0.7, and 3.0. Examples of unacceptable formats are: 3.45 and 0.47.

SHRTSL=ttt

Specifies the increment of time to be used to share time on a rotating basis between user programs and background processing. When an increment is used up, a decision is made whether the next increment is to be given to the currently-executing user program or to the background. This decision is based on the mode of allocation in effect at the time; the mode is controlled with the *BATCH command, as described in the publication CALL-OS Operator's Manual.

The minimum value of ttt is 0.1 second, the maximum value is 10 seconds, and the default value is one second; it must always be less than the old job time slice for this session (see RUNTSL). Reducing this value below one second increases system overhead involved in task switching, but it may improve overall system performance through more effective use of the I/O devices associated with background jobs.

SYSCON=(ppp,aaa)

A maximum of two command consoles is permitted on the system; this parameter assigns logical line numbers to the command console(s),

where

ppp is the logical line number of the primary command console
aaa is the logical line number of the alternate command console.

The default is logical line number 1 for the primary command console and logical line number 0 for the alternate.

Note that a zero value for the primary console is treated as null, in which case the default value is used.

Additional COBI Options

These options are specified only when the COBI facility is used; If COBI is not used, these options are ignored if present. The options which may be specified and their associated parameters are:

- Whether or not COBI is to be active for this session of CALL-OS - NOCOBI parameter
- The automatic starting of the reader - AUTRDR parameter

- The maximum number of jobs to be submitted before the COBI input data sets are switched - RDRQTY parameter
- The time interval that is to elapse before the COBI input data sets are switched - RDRTIM parameter
- The ability to allow the user to specify a name for each job submitted - ANYJNAME parameter
- The primary and secondary allocation for user-defined scannable output data sets created by COBI jobs - DSPACE parameter
- Whether the above allocation is in cylinders or tracks - ALOCTYPE parameter
- The direct access unit name to be used for scannable output data sets created by COBI jobs - UNITNM parameter
- The maximum number of users permitted to scan data sets simultaneously - MAXDCB parameter
- The high level index qualifiers of system data sets that may be scanned by the user - SCANDS parameter
- The output class for the JCL and unscannable SYSOUT data sets for COBI jobs - CBCLASS parameter
- The output class to be used to return COBI JCL and SYSOUT data sets to OS/360 for printing - OSCLASS parameter

The parameters, their formats, and appropriate defaults are described in the following text in alphabetical order by parameter.

ALOCTYPE=xxx

Specifies whether the space allocation for user-defined scannable output data sets created by COBI jobs (established by the DSPACE parameter) is in tracks or cylinders, indicated by TRK or CYL, respectively. The default for this parameter is CYL, since cylinder allocation leads to better performance.

ANYJNAME

Indicates that the terminal user may supply a job name for his jobs; COBI does not alter the job name. If this parameter is omitted, COBI creates a job name for each submitted job. The COBI-created name consists of the user number and a two-byte identifier which is obtained from the assigned job number.

Note: If ANYJNAME is specified, duplicate job names may occur. If a user later cancels a job, no guarantee can be made that his job will be cancelled.

AUTRDR

Indicates that CALL-OS is to start the reader (DIBRDRA or DIBRDRB) to read the prepared COBI input data set. If this parameter is omitted, the operator must start the reader.

This parameter may be overridden during execution of CALL-OS by specifying the RESET,AUTRDR function of the *COBI command. For a complete description of this command, see the publication CALL-OS Operator's Manual.

CBCLASS=a

Specifies the COBI output class and must be a valid output class (0 through 9 or A through Z). If this parameter is omitted, the default is output class Z.

This class contains any COBI job SYSOUT data sets which were not specified as scannable when the job was submitted and all the JCL associated with the COBI jobs. COBI intercepts the COBI output class when the job terminates and records status information about the job and its data sets. In addition, if the JCL is to be saved, it is copied into the JCL data set. The output for the job is then reassigned to an OS/360 output class (see the OSCLASS parameter) for printing by an output writer.

The COBI output class must be the class specified in the CBCLASS parameter for the DIBCONPR utility program when the cataloged procedures were converted.

DSPACE=(ppp,sss)

Specifies, in cylinders or tracks, the primary (ppp) and secondary (sss) space allocation for user-defined scannable output data sets created by COBI jobs. (A space allocation is assigned to procedure-defined SYSOUT data sets when the procedures are converted.) Either or both allocation specifications may be omitted. If a parameter is omitted or if a zero value is specified, the default is one cylinder for the primary allocation and one cylinder for the secondary allocation if ALOCTYPE=CYL, or ten tracks for the primary and ten tracks for the secondary if ALOCTYPE=TRK.

MAXDCB=nn

Specifies the number of users permitted to scan data sets at the same time; the maximum which may be specified is the number of logical lines defined in the startup deck for this session. If this parameter is omitted, the default is one user for every ten lines with a minimum of two users. Any number above this amount should be determined by the anticipated number of COBI users who will be scanning data sets. Daily operation of the installation will indicate if the number should be increased. If a particular installation uses COBI heavily, it may be necessary to allow six or eight users for every ten lines.

NOCOBI

Specifies that COBI is not to be active for this session of CALL-OS. The resident COBI modules are not loaded with the CALL-OS nucleus.

OSCLASS=a

Specifies an OS/360 output class and must be a valid output class (0 through 9 or A through Z). It must not be the same as the class specified in the CBCLASS parameter. If this parameter is omitted, the default is output class A.

The output associated with a COBI job is reassigned from the COBI output class (see the CBCLASS parameter) to the OS/360 output class after the job has been processed by the COBI JCL module (see the description of the JCL data set in the chapter on COBI). The OS/360 output class is processed by an OS/360 output writer which prints the output on the high-speed printer.

RDRQTY=nnn

Specifies the maximum number of jobs (nnn) to be accepted into a COBI input data set before a switch is made to allow the other to accept input. The switch is made only if the reader has finished reading the other data set. The number specified must be within the range 0 through 999; a value of 0 makes the function inoperative. If this parameter is omitted, the default is ten jobs.

The parameter, or the default, may be overridden by specifying the RESET,RDRQTY function of the *COBI command. Any RDRQTY specification is ignored if the ANYBATCH function of the *COBI command is used. For a complete description of the *COBI command, see the publication CALL-OS Operator's Manual.

RDRTIM=mmm

Specifies the time interval in minutes (mmm) before a switch is made between the COBI input data sets. The switch is made only if the current input data set has one or more jobs in it and if the reader has finished reading the other data set. The number specified must be within the range 0 through 999; a value of 0 makes the function inoperative. If this parameter is omitted, the default is 15 minutes.

The parameter, or the default, may be overridden by specifying the RESET,RDRTIM function of the *COBI command. Any RDRTIM specification is ignored if the ANYBATCH function of the *COBI command is used. For a complete description of the *COBI command, see the publication CALL-OS Operator's Manual.

SCANDS=
(index1,index2)

Specifies the high level index qualifier of system data sets that may be scanned by the terminal user; the user is not permitted to modify or scratch the data sets. If, for example, SCANDS=SYS1 is specified, any COBI user may scan any data set with SYS1 as the high level index qualifier, such as SYS1.MACLIB or SYS1.PROCLIB. It is recommended that the SCANDS parameter contains the high level index qualifier of the COBI procedure library; it is this library which contains the converted procedures.

Only two qualifiers may be specified and each may be up to eight characters in length. If this parameter is omitted, the user is permitted to scan only those data sets with his user number as a qualifier in the data set name.

UNITNM=name

Specifies the group of devices containing the volumes to be used for scannable output data sets created by COBI jobs, either by device type or generic name. The generic name must have been specified in the UNITNAME macro during system generation. In addition, a specific unit address may also be specified. If this parameter is omitted, the default is device type 2314.

DESCRIPTION OF JCL STATEMENTS

The data set names used in the startup deck must correspond to the names supplied for the system build utility programs. The data set names shown in the startup deck in the figure are the names created and cataloged by default during system build. The user may create his own

data set names with the JCL statements. In this case, the data set names are optional and the VOLUME and UNIT parameters must be supplied on the DD statements. The names of the DD statements must be as shown.

The following text describes the JCL statements in the order in which they appear in the startup deck shown in Figure 29.

JOB Statement

The JOB statement in the startup deck for CALL-OS must meet certain requirements; it:

- must specify a job name (in the sample the name is CALLOS); this name appears in the termination message issued by OS/360 when CALL-OS terminates abnormally,
- must specify ROLL=(NO,NO) unless this parameter is specified on the EXEC statement,
- should assign a unique job class to CALL-OS,
- must specify a REGION for MVT unless this parameter is specified on the EXEC statement.

JOBLIB DD Statement

The JOBLIB DD statement designates the private library containing all CALL-OS load modules. Instead of a JOBLIB statement, a STEPLIB DD statement may be used. If all modules reside in SYS1.LINKLIB, the JOBLIB (or STEPLIB) statement may be omitted.

EXEC Statement

The EXEC statement in the startup deck for CALL-OS must meet certain requirements; it:

- must specify the CALL-OS program name; that is PGM=RTOS1,
- must specify ROLL=(NO,NO) unless this option is specified on the JOB statement,
- may specify initialization options in the PARM field if less than 100 characters of information is supplied, and
- must specify a REGION for MVT unless this parameter is specified on the JOB statement.

SYSABEND DD Statement

The SYSABEND DD statement identifies the data set for ABEND dumps and need only specify the appropriate SYSOUT class.

SYSPRINT DD Statement

The SYSPRINT DD statement specifies the output destination for statistics produced by the *REPORT command and, if COBI is present, for status requests issued by the operator with a *COBI-P command. The DD statement may specify either a SYSOUT class or a dedicated printer. When the data set is directed to SYSOUT, the default block size

parameter is 665 bytes. The user may override this value in 133-byte increments up to a maximum of 1995 bytes.

INDEX DD Statement

The INDEX DD statement identifies the CALL-OS index data set created at data base build time. This data set contains, among other items, all possible data base DD statement names permissible in this startup deck.

RESMODS DD Statement

The RESMODS DD statement identifies the list of potentially nonresident modules that are to be made resident during the current run of CALL-OS. For convenience the procedure library supplied with the system contains four members which may be used to specify module residency. When one of these members is used, the DSNAME specification must contain both the data set name and the member name, where the data set name specifies the procedure library and the member name specifies one of the following:

RTOSALL	All modules to be made resident.
RTOSNONE	All potentially nonresident modules to be made nonresident.
RTOSLLRS	All modules for the load, list, run, and save functions to be made resident.
RTOSUSER	All modules for user terminal command functions to be made resident; all modules for operator command functions to be nonresident.

In addition, the user may use lists other than those in the procedure library. The user-supplied list may be a sequential data set, a member of a partitioned data set, or a data set in the input stream. The format of the list is identical in all three cases. The list consists of 80-byte card images, beginning in column 1 and continuing up to and including column 71. Column 72 must be blank. Scanning of any card is terminated when the first blank is encountered. As many card images as are necessary may be used to complete a list. Module names in the list must be separated by commas except for the last name on the card.

If hierarchy support is part of the OS/360 system, the RESMODS list may be used to designate the hierarchy into which a module is loaded. The hierarchy specification is placed in parentheses after the module name, for example, M#LOAD(1) indicates that M#LOAD is to be placed in hierarchy 1. Hierarchy 0 may also be specified, but it is the default. More information on using hierarchy support is contained in the section "Designing the System".

If the RESMODS and OVLY DD statements are omitted, all potentially nonresident modules are to be made resident; if the RESMODS DD statement is omitted and the OVLY DD statement is included, all of these modules are to be nonresident.

OVLY DD Statement

The OVLY DD statement specifies the data set to be used by the CALL-OS initialization routines. This data set contains copies of modules selected from the CALL-OS load module library that are to remain nonresident for today's run. For a totally resident system the OVLY DD statement should be omitted.

BASIC, FORTRAN, PLI, and PL2 DD Statements

The BASIC and FORTRAN DD statements specify the location of the BASIC and FORTRAN compilers, respectively. The PLI and PL2 DD statements specify the location of the first and second phases, respectively, of the PL/I compiler; if PL/I is to be used, both statements must be present. The compiler data sets are created and loaded during the system build process.

Only those compilers to be made active during the current online session need to be represented by DD statements. The initialization routines determine the amount of core storage to be set aside for the compiler area by using the value of the size for the largest compiler to be used in the system. For example, if the PLI and PL2 DD statements are omitted, the compiler area will be large enough for either BASIC or FORTRAN.

SWAPnn DD Statement

Each SWAPnn DD statement identifies one of several (or only one) work/swap data sets to be used by the online system. At least one SWAP DD statement must be present; additional work/swap data sets are identified by nn, in the range 00 through 19.

Any subset of the possible 20 work/swap data sets can be specified during initialization. However, the total space in all data sets specified must allow one cylinder for each terminal line (specified with the TWX, T2741, and T2741E DD statements). All the lines are opened during initialization and each line assigned to one work/swap cylinder.

The cylinders are assigned on a round-robin, first come first served basis: the first line (by logical line number) is assigned to the first cylinder of the first data set; the second line is assigned to the first cylinder of the second data set, etc. The data sets are used in ascending sequence, according to the nn in the DD statement name. For example, if SWAP12, SWAP05, and SWAP08 are defined in the startup deck, the first line is assigned to the data set defined by the SWAP05 DD statement.

SYSGRPnn DD Statement

The SYSGRPnn DD statement identifies one of several (or only one) system group data sets (that is, those data sets used for system libraries). At least one SYSGRPnn DD statement must be present. If only one is present, the number must be either 00 for the primary cluster or 40 for the alternate cluster; if several DD statements are required to define the system group, the numbers must be in sequence, either from 00 through 39 or from 40 through 79. All statements must belong to the same cluster.

User Group DD Statements

User group DD statements have the form aaabbbnn and identify those user groups to be present in the current session. For each DD statement, aaabbb specifies the range of user numbers in the group, where aaa indicates the first three characters of the low-order user number and bbb indicates the first three characters of the high-order user number. For example, AAABBB indicates that user numbers AAA001 through BBB999 are in this group.

At least one user group DD statement must be present, and each user group may have more than one data set. The data sets for a group are

sequenced by nn, starting with 00 for the primary cluster and 40 for the alternate cluster. If only one DD statement is required for a group, the number must be either 00 or 40; if more than one DD statement is required, the numbers must be in sequence, from 00 through 39 for the primary cluster and 40 through 79 for the alternate cluster.

If a group is to be present for a session, all the data sets for the group must be present and from the same cluster. Entire groups from either cluster may be present during the same session as long as their user number ranges do not overlap.

TWX, T2741, and T2741E DD Statements

The TWX, T2741, and T2741E DD statements identify the terminals to be brought up by CALL-OS during initialization and enabling of the system. The statements and the associated terminal type are:

- TWX Identifies a group of Teletype (TTY) terminals and causes the T#TTYTAB translation table to be loaded
- T2741 Identifies a group of 2741 Correspondence Terminals and causes the T#27CTAB translation table to be loaded
- T2741E Identifies a group of 2741 EBCD terminals and causes the T#27ETAB translation table to be loaded

If the system is not using one or more of the terminal types, the appropriate DD statement may be omitted.

At least one of the TWX, T2741, or T2741E DD statements must be part of the startup deck. The unit addresses may be expressed either in generic terms by using the OS/360 system generation convention or in the form of actual unit addresses. If actual unit addresses are used, the DD statements for the terminals must be concatenated.

Logical Line Numbers: Logical line numbers begin with the Teletype terminals, followed by 2741 correspondence and 2741 EBCD terminals in that order. For example, if there are two teletype terminals and two 2741 correspondence terminals, the first 2741 correspondence terminal is logical line number 3 while the first 2741 EBCD terminal is logical line number 5. The order of the DD statements containing the ddnames has no effect on the assignment of logical line numbers.

Translate Table Loading: Translate table loading may be indicated either by the presence of a DD statement for the terminal type and/or by the presence in the RESMODS list of the translate table name: T#TTYTAB for TTY, T#27CTAB for 2741 correspondence, and T#27ETAB for 2741 EBCD. Hierarchy specifications are honored if defined in the RESMODS list; otherwise, the default is the hierarchy in which the system pots are loaded. Separate DCBs are built for each of the three terminal types only if that terminal type is requested via a DD statement.

Considerations for 2741 Terminals: If only one type of 2741 terminal is physically connected to a system, then that customer installation may specify this particular terminal type (correspondence or EBCD) in the appropriate DD statement, and instruct its 2741 terminal users to depress the RETURN only for the sign-on procedure.

A customer installation may employ a large number of one type of 2741 terminal on a system where terminal users depress the RETURN only for the sign-on procedure. If a relatively small number of the second type

of 2741 terminal is now added to the system, this installation may specify all the 2741 terminals with one type of 2741 DD statement and instruct users of the second 2741 terminal type to perform the standard sign-on procedure. Users of the first 2741 terminal type can continue to sign on by depressing the RETURN key only.

Both the 2741 correspondence and 2741 EBCD terminal types can be supported by one DCB. This is accomplished by specifying one type of DD statement for all 2741 line addresses, and requesting the loading of the translate table for the type of 2741 terminal not specified on the DD statement via the RESMODS list. This gives a customer installation the advantage of employing only one rotary for all 2741 lines, and, in addition, conserves core storage for DCBs.

Additional DD Statements for COBI

If COBI is to be used, additional DD statements are required. If COBI is not to be used, these statements are ignored if present. The statements and their requirements are as follows:

SYSJOBQ	Defines the OS/360 system job queue data set, SYS1.SYSJOBQE.
CBNDX	Defines the COBI index data set, which contains information about each job submitted to OS/360 from a user terminal.
CBJCL	Defines the COBI JCL data set. If the user requests that the JCL for a submitted job be saved for scanning at the terminal, the JCL is written into the JCL data set. If the user does not request JCL scanning but his job contained a JCL error, the JCL is also saved.
CBSYSINA CBSYSINB	Define the data sets which are to contain jobs submitted through COBI for execution under OS/360. Ideally, one data set is attached to COBI to receive jobs while the other is being read into OS/360 batch processing. Both statements must be present when COBI is used.
SCANxx	Defines those volumes which contain scannable data sets, where xx is a two-character identifier used to make each DD statement unique. At least one SCANxx DD statement must be present if scanning of COBI job output at a terminal is to be allowed. One DD statement must be supplied for each volume which is to contain scannable data sets produced by COBI jobs. These volumes are mounted on the class of devices specified in the UNITNM parameter. In addition, if the SCANDS parameter is specified, a DD statement must be supplied for each volume which contains the system data sets associated with the index level qualifier. For example, if SCANDS=SYS1 is specified, one DD statement must be supplied with the volume serial number of each of the system residence packs. The DSNAME parameter must not be specified on any SCANxx DD statement.

SYSIN DD Statement

If more than 100 characters of parameter information are required, additional parameter information may be entered following a SYSIN DD * statement as 80-column card images. Information in these card images must always begin in column 1 and must not extend beyond column 71. Keyword parameters and their accompanying values may not extend from one card to the next. The format for all keywords is identical to the format required when the information is entered with the EXEC statement. Note that scanning of each card continues until the first blank is reached. No comma is required after the last parameter on any card, and no use is made of any continuation indicators in column 72. As many additional cards as required may be used, provided that no more than 400 characters of information are specified altogether.

CALL-OS provides several utilities that may be used in the building and maintaining of the data base. Because these utilities are diversified in function, they are gathered into one chapter for the convenience of the user. The utilities are:

- U#UTIL3 Formats the index data set
- U#UTIL1 Constructs the data base
- UTILX Maintains the index data set
- DIBCADBU Maintains user base data sets

Two of these utilities are involved in the creation of the index and the data base: U#UTIL3 and U#UTIL1. U#UTIL3 formats the index data set to a full track of X'FF' (hexadecimal); this formatting is the only function carried out by U#UTIL3. Following the execution of U#UTIL3, information describing each data set within the data base is added to the index by running U#UTIL1, which must process every part of the data base. In addition to making entries in the index for each data set, the utility may perform other special processing depending on the type of data set which is being processed. After the necessary U#UTIL1 runs, the data base should be ready for use by the system.

The other two utilities are involved in the maintenance of the index and the data base: UTILX and DIBCADBU. UTILX may be used to update and modify the index data set. DIBCADBU consists of a number of separate functions which are used to perform a variety of operations on the data base. For example, this utility may be used to reorganize the data base, perform backup or accounting operations, output a portion of the data base, and perform additions, deletions, or replacements of information in the data base.

This section describes the four utilities in greater detail.

U#UTIL3 - FORMATTING THE INDEX

The only purpose of this utility is to format the one-track index data set to hexadecimal 'FF's. It is run as part of the third step of the system build process, and should be run thereafter only when some exceptional situation makes a complete recreation of the index necessary. An example of the JCL required to execute U#UTIL3 follows:

```
//NAME     JOB     ---
//JOBLIB   DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//         EXEC    PGM=U#UTIL3
//INDEX    DD      DSNAME=OSRTS.INDEX,DISP=(NEW,CATLG),
//                 UNIT=2314,VOLUME=SER=XXXXXX,
//                 SPACE=(TRK,1)
```

Since the index is read only once by the online system, its physical location has no effect on the performance of the system.

U#UTIL1 - BUILDING THE DATA BASE

This utility performs those functions necessary to prepare the data base data sets for use with the system. It is employed as part of the

default data base build procedures; its use is also required when building a custom-tailored data base and when adding additional data sets to an existing data base. U#UTIL1 must also be used to preformat any output data sets required for the RECONSTRUCT and REORGANIZE functions of the data base utility. In this case, if the data sets already exist in the data base, UTILX must be used first to delete the appropriate index entries.

Five different types of data sets can be processed by U#UTIL1. The desired function is indicated by specifying one of the five literals listed below in the PARM field of the EXEC statement.

```
//          EXEC    PGM=U#UTIL1,PARM='COMPILER'
                                     'SYSGROUP'
                                     'USRGROUP'
                                     'WORKSWAP'
                                     'OVERLAY'
```

Only one function per run may be invoked, but more than one data set within a function can be processed during a run. During execution of U#UTIL1, the ddname from each compiler, system group, user group, work/swap, or overlay data set DD statement is printed on the device indicated by the SYSPRINT DD statement (which would normally be the system printer SYSOUT=A).

This information appears below an appropriate heading, together with the corresponding ddname and dsname entered in the index. Any violation of the ddname naming conventions for CALL-OS data sets or conflict with existing index entries causes an appropriate error message to be printed below the pertinent DD statement and index information; the program proceeds to the next statement without performing any data set processing or index updating function for the current DD statement. If an error occurs during the processing of any of these data sets, an error message is printed, the processing of the data set is terminated, and the index is not updated with the new DD statement information. (See the CALL-OS Operator's Manual for a list of U#UTIL1 system printer (SYSPRINT) error messages.)

The following subsections describe the special processing that takes place for each of the five types of data sets.

COMPILER DATA SETS

One load module version of each compiler in the system resides in the CALL-OS JOBLIB partitioned data set. Performance considerations require that the reading of these compilers be done as rapidly as possible; for that reason, the version of each compiler which executes as part of the online system resides in a modified format on a separate data set. This special version of each compiler is created by U#UTIL1.

For each compiler run of U#UTIL1, the LANG DD statement points to the JOBLIB data set which contains members whose names match the ddnames of the other DD statements provided. These other DD statements point to the data sets into which the modified versions of the compilers are to be written. The utility first ensures that no multiple extents or alternate tracks have been assigned within the space allocated to the compiler data set, and that the data set does not cross any cylinder boundaries. It then rewrites the compiler into that area in the necessary format. Where optimum performance is desired, care should be taken in allocating the compiler data sets. (Particular attention should be given to the discussion of the central cylinder functions in the section "Designing the System".)

In addition to rewriting the compilers in the data sets provided, the index is updated to reflect the new status of each compiler. If no previous entry with the compiler name exists in the index, an entry is created which contains the compiler name, the name of the data set in which it is being rewritten, and the length of the compiler. If the index already has an entry for this compiler, the entry is updated only to reflect the current data set name and the compiler length. If the user desires to retain the old copy of a compiler for backup purposes, he may specify a new data set name. However, to return to a previous version of the compiler, he must update the index with the UTILX utility (see "UTILX-Modifying the Index").

Note that the online system still requires the JCL necessary to define the compiler data sets; the index is used only for maintaining control information about the data sets.

The following example shows the JCL necessary to run U#UTIL1 on the three compilers provided with the system. Note particularly the manner in which the two-phase PL/I compiler must be handled, and that compiler allocations must not cross cylinder boundaries.

```
//NAME     JOB     ---
//JOBLIB   DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//         EXEC    PGM=U#UTIL1,PARM='COMPILER'
//SYSPRINT DD      SYSOUT=A
//INDEX    DD      DSNAME=OSRTS.INDEX,DISP=OLD
//LANG     DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//FORTRAN  DD      DSNAME=OSRTS.FORTRAN,SPACE=(CYL,1),
//                 DISP=(NEW,CATLG),UNIT=2314,VOL=SER=XXXXXX
//BASIC    DD      DSNAME=OSRTS.BASIC,SPACE=(CYL,1),
//                 DISP=(NEW,CATLG),UNIT=2314,VOL=SER=XXXXXX
//PLI      DD      DSNAME=OSRTS.PLI,SPACE=(CYL,1),
//                 DISP=(NEW,CATLG),UNIT=2314,VOL=SER=XXXXXX
//PL2      DD      DSNAME=OSRTS.PL2,SPACE=(CYL,1),
//                 DISP=(NEW,CATLG),UNIT=2314,VOL=SER=XXXXXX
```

WORK/SWAP DATA SETS

Each data set specified by a SWAPnn DD statement is validated to ensure that no multiple extents or alternate tracks have been assigned, and that the data set starts and ends at cylinder boundaries. If the data set is satisfactory, an index entry is created or updated.

One cylinder in a work/swap data set must be available to the online system for each line being enabled. If optimum system performance is desired, care should be taken in allocating these data sets. (For more information on this subject, refer to the discussion of the central cylinder functions under "Designing the System".)

The following is an example of the JCL required to add two work/swap data sets to the system:

```
//NAME     JOB     ---
//JOBLIB   DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//         EXEC    PGM=U#UTIL1,PARM='WORKSWAP'
//SYSPRINT DD      SYSOUT=A
//INDEX    DD      DSNAME=OSRTS.INDEX,DISP=OLD
//SWAP00   DD      DSNAME=OSRTS.SWAP00,DISP=(NEW,CATLG),
//                 SPACE=(CYL,30,,CONTIG),UNIT=2314,VOL=SER=XXXXXX
//SWAP01   DD      DSNAME=OSRTS.SWAP01,DISP=(NEW,CATLG),
//                 SPACE=(CYL,30,,CONTIG),UNIT=2314,VOL=SER=YYYYYY
```

OVERLAY DATA SET

The data set specified by the OVLY DD statement is validated to ensure that the data set resides on a single volume; an index entry is then created or updated. If optimum system performance is desired, care should be taken in allocating this data set. (Refer to the discussion of central cylinder functions under "Designing the System".)

The following is an example of the JCL required to add the overlay data set to the system:

```
//NAME      JOB     ---
//JOBLIB    DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//          EXEC    PGM=U#UTIL1,PARM='OVERLAY'
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSNAME=OSRTS.INDEX,DISP=OLD
//OVLY      DD      DSNAME=OSRTS.OVLY,DISP=(NEW,CATLG),
//                  UNIT=2314,VOLUME=SER=XXXXXX,SPACE=(CYL,1)
```

SYSTEM GROUP DATA SETS

Each data set specified by a SYSGRPnn DD statement is validated to ensure that the data set resides on a single volume; if the data set is the first in the group, it is checked to ensure that at least three tracks have been allocated. The data set is then formatted, and an index entry is created. Existing system group index entries cannot be updated by U#UTIL1; however, additional data sets which come immediately after the existing data sets in sequence may be added at any time. The formatting of the data set consists of:

1. Building an allocation record and nine empty records on the first track of the data set.

2. Writing an equivalency record on the second track of the data set, and a catalog record (for the *** programs and data files) and a directory record (for the ** programs and data files) on the third track, if the data set is the first in the group.

3. Formatting the remainder of the space in the data set into half track records.

Existing system group data sets cannot be processed by U#UTIL1 unless they are first deleted from the index by UTILX. This should only be done when it is desired to delete an entire system group, thus destroying the accumulated data for that group. (See the description of UTILX elsewhere in this section.)

An example of the JCL required to add a system group data set follows.

```
//NAME      JOB     ---
//JOBLIB    DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//          EXEC    PGM=U#UTIL1,PARM='SYSGROUP'
//SYSPRINT  DD      SYSOUT=A
//INDEX     DD      DSNAME=OSRTS.INDEX,DISP=OLD
//SYSGRP00  DD      DSNAME=OSRTS.SYSGRP00,DISP=(NEW,CATLG),
//                  SPACE=(CYL,4),VOL=SER=XXXXXX,
//                  UNIT=2314,DCB=DSORG=DA
```

Note: U#UTIL1 assigns a password of SECURITY to the SYSLIB user. The person in charge of system security should alter this password as soon as possible to prevent unauthorized use of the data base utility DIBCADBU.

USER GROUP DATA SETS

These data sets are processed by U#UTIL1 in the same manner as system groups, except that U#UTIL1 does not write a catalog record and a directory record on the third track of the first data set in each group. Therefore, user group data sets are checked for a minimum of two tracks rather than three. The ddnames supplied must meet the specifications listed under "CALL-OS Data Base".

Whenever a new user group is needed, or the data sets in any existing user group have been filled up, the USRGROUP option of this utility should be used to add new data sets to the data base. When this utility is employed to add additional data sets to an existing user group, the new data sets must immediately follow the existing data sets in that group sequence.

Existing user group data sets cannot be processed by U#UTIL1 unless they are first deleted from the index by UTILX. This should only be done when it is desired to delete an entire user group, thus destroying the accumulated data in that group. (See the description of UTILX elsewhere in this section.)

An example of the JCL required to add two user groups to the system follows:

```
//NAME     JOB     ---
//JOBLIB   DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//         EXEC    PGM=U#UTIL1,PARM='USRGROUP'
//SYSPRINT DD      SYSOUT=A
//INDEX    DD      DSNAME=OSRTS.INDEX,DISP=OLD
//AAABZZ00 DD      DSNAME=OSRTS.AAABZZ00,DISP=(NEW,CATLG),
//                 UNIT=2314,VOLUME=SER=XXXXXX,
//                 SPACE=(CYL,80),DCB=DSORG=DA
//CAADZZ00 DD      DSNAME=OSRTS.CAADZZ00,DISP=(NEW,CATLG),
//                 UNIT=2314,VOLUME=SER=YYYYYY,
//                 SPACE=(CYL,40),DCB=DSORG=DA
//CAADZZ01 DD      DSNAME=OSRTS.CAADZZ01,DISP=(NEW,CATLG),
//                 UNIT=2314,VOLUME=SER=ZZZZZZ,
//                 SPACE=(CYL,40),DCB=DSORG=DA
```

UTILX - MODIFYING THE INDEX

After a system and its data base have been built, situations may arise where it is necessary to modify the index without modifying the individual data sets within the data base. UTILX provides this capability. It may be used to add, delete, or replace entries in the index. As part of its processing, UTILX produces a listing of the index before updating, as well as another list of the updated copy. It can also produce a punched card version of the index which can be used to recreate the index if necessary.

JCL STATEMENTS

The following is a sample of the JCL required to execute UTILX:

```
//NAME     JOB     ---
//JOBLIB   DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//         EXEC    PGM=UTILX
//INDEX    DD      DSNAME=OSRTS.INDEX,DISP=OLD
//SYSPUNCH DD      SYSOUT=B      <---PUNCHED OUTPUT---
//SYSPRINT DD      SYSOUT=A
//SYSIN    DD      *
ADD
```

```
(Detail cards for entries to be added)
DEL
(Detail cards for entries to be deleted)
/*
```

The program is controlled through the cards read in following the SYSIN DD statement. A function control card (with either ADD or DEL in columns 1-3) controls processing of detail cards until another function control card is read.

DETAIL CARDS

The detail cards are free-form, with the fields separated by one or more blanks or a comma surrounded by optional blanks. Successive commas are used to indicate an omitted field. The fields are as follows:

Field	Field Definition
1	Entry type. This must be up to three decimal digits identifying the appropriate code from the list below:
2	Relative data set number, from 0 through 39 for the primary cluster, and 40 through 79 for the alternate cluster. For detail cards other than work/swap, system, or user groups, this field should be zero.
3	Group identification. For system groups, the value is SYSLIB. For user groups, the value is of the form aaabbb where aaa is the three-character lower range and bbb is the three-character upper range for user numbers in the group. For compilers, the value is the first six characters of the ddname. For work/swap entries the value is SWAP and for overlay entries, the value is OVLY.
4	The name (up to eight characters) of the DD statement which defines the data set.
5	The data set name (up to 44 characters) assigned to the data set.
6	Compiler size. Up to six decimal digits, indicating total bytes in each compiler data set. For entries other than compiler entries, this field should be zero.

Code	Meaning
1	Compiler
2	Overlay
100	Work/swap
150	Primary cluster for system group
160	Primary cluster for user group
170	Alternate cluster for system group
180	Alternate cluster for user group

For an ADD entry, all six data fields must be present. A previous entry in the index containing the same entry type and ddname is replaced by the ADD function. For the DEL function, only the first and fourth data fields need be present.

OUTPUT

The output produced by UTILX consists of a three-part printed listing and, optionally, a card deck. The first printout that appears is a listing of the index entries before any changes are made. The second printout is a list of the detail card images. If an error is found during the reading of a detail card, a message is printed below the card, the index is not updated, and the program processes the next card. The third printout is a listing of the index after it has been updated. The card deck which is punched consists of the index entries, exactly as they appear in the updated listing of the index. These cards can be used as detail cards for subsequent runs of UTILX, as they adhere to the format described for the detail cards.

DIBCADBU - MAINTAINING THE DATA BASE

The CALL-OS Data Base Utility (DIBCADBU) provides capabilities for manipulation and maintenance of the CALL-OS data base. Principle components of the data base to be manipulated are user catalogs, program and data files, and shared directories and libraries. Program files and data files are created by remote terminal users or with the data base utility. In this chapter, the terms "program" and "program file" imply either a saved source program file or a stored object program file.

INTRODUCTION

The data base utility is a comprehensive group of generalized utility and maintenance routines designed to assist the user in the day-to-day operation of the system. By this means, the most frequently required services of data base maintenance and manipulation can be performed with a minimum of effort. The data base utility can be executed as an offline batch program in any OS/360 environment to provide a complete range of services.

All data sets used by the data base utility are standard OS/360 data sets. Among these are the CALL-OS system group and user group data sets, which are suballocated into logical files of several types. All such files created or modified by the utility are standard CALL-OS files. For a detailed description of these files, see the "CALL-OS Data Base" section.

Using the Data Base Utility

The data base utility consists of nine separate functions: ACCOUNT, DELETE, JOBFIND, INSERT/REPLACE, RECONSTRUCT, REORGANIZE, TAPE, VALIDATE, and WRITE. Each function is processed by one serially reusable module which operates under control of the main utility control module (see Figure 30). The actual processing performed by each module is specialized by means of user-provided utility control statements.

When a control statement is recognized, the utility control module (which is always resident) loads the appropriate function module and transfers control to it. When the function module has finished processing the request, the module is deleted unless the next control statement requests the same function. For this reason, all requests for the same function should be grouped together to reduce the time required to load function modules and improve overall utility performance.

All functions of the data base utility except the WRITE function operate offline, when CALL-OS is not in operation against the same system group and/or user group data sets. The WRITE function may be executed concurrently with CALL-OS as long as certain rules are

followed. These rules are described in more detail under "WRITE Function".

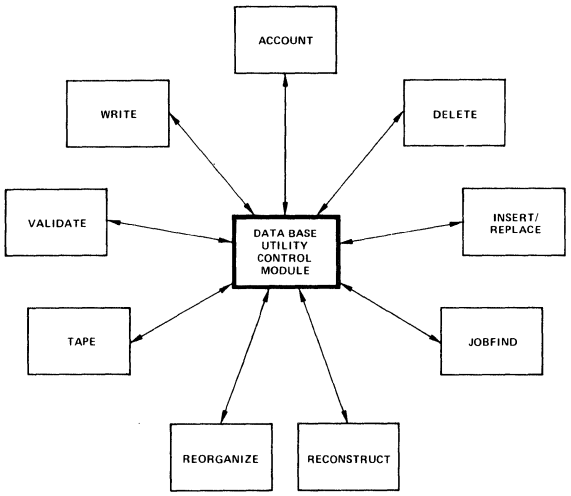


Figure 30. Data base utility program structure

Ensuring File Security

The data base utility is designed to maintain the standards of system integrity established by CALL-OS online system operation. To protect against inadvertent access to or destruction of retained program and data files, the CALL-OS online system requires that each terminal user follow a specific sign-on procedure, using his user number and password. All resources belonging to a user are protected under this identification. No file can be added to or deleted from an individual user library unless the required user password has been specified.

In a similar manner, the data base utility requires specification of the required user password prior to insertion, replacement, or deletion of a program or data file from an individual user library. This user number and password security measure applies to both system group and user group data sets; additional security measures are available for user groups.

System Group Security: Since the data base utility may also be used to manipulate various system control files, a SYSLIB password facility is provided in addition to the user password. A system group created by U#UTIL1 has a default SYSLIB password of SECURITY; to ensure security, the system programmer must issue the PASSWORD terminal command through the command console and assign a new password to SYSLIB for use by the data base utility. To perform certain data base manipulations, some of which are equivalent to online command console functions, the function statement must contain the password that has been assigned to SYSLIB. This password is checked by the data base utility before these functions can be executed.

Note: The default password of SECURITY applies to the current version of CALL-OS. If a data base created by a previous version of CALL-OS is used, the default password for the system group may be RTOS or PASSWORD. This password must be changed with the PASSWORD command from the command console. The PASSWORD command is described in the publication CALL-OS Terminal Operations Manual.

User Group Security: The terminal user can also control the copying or inserting of his program and data files into another user's library. All program or data files saved or stored by a terminal user or inserted in a user library by means of the data base utility are initially in an unreleased, or secure, state. When a file is in this state, it cannot be inserted by another user into his library unless the originating user releases the file with the RELEASE command. The data base utility returns the file to a secure state after accessing the file. If a user releases a file but does not use the data base utility, the originating user may return the file to the secure state by issuing a SECURE command at his terminal.

Additional security features can be imposed by a terminal user through use of the terminal command language. The following features are available to remote terminal users with the terminal command language, and are also available to users of the data base utility with parameters on the function statement.

- LOCK

  Prevents a program or data file in the user's library from being accidentally destroyed. No program or data file with the same name can be saved or stored in the library by a remote terminal user, and the file cannot be purged from the library. Similarly, a locked file cannot be replaced or deleted using the data base utility. A file can be locked either by issuance of the LOCK command from the terminal or as it is inserted in a user library with the data base utility.

- UNLOCK

  Removes protection from a previously locked program or data file. A file can be unlocked either by issuance of the UNLOCK command from the terminal or as it is inserted in or removed from a user library with the data base utility.

- PROTECT

  Specifies that the named, saved, or stored program or data file is run only. The name of the program or data file is in the *Directory, the **Directory, or the ***Library. A protected program file cannot be listed, altered, saved, or stored by any user other than the originator. A protected data file may not be opened by other users of the library. A file can be protected either by issuance of the PROTECT command from the terminal or as it is inserted in a user library by the data base utility. The user who protected the file may remove the protection by issuing the ALLOW command at his terminal.

- POOL *program or data file name
- POOL **program or data file name

  Makes a user's stored or saved program or data file available to other users through the shared library facility. If a file having the same name is already listed in the *Directory or **Directory, pooling of the program is not allowed.

- PULL *program or data file name
- PULL **program or data file name

  Removes the program or data file name from the *Directory or **Directory. A file can be pulled either by issuance of the PULL command from the terminal or with the data base utility.

For a complete description of the CALL-OS terminal command language, see the CALL-OS Terminal Operations Manual.

CONTROL STATEMENTS FOR EXECUTION

Two types of control statements are required to execute the data base utility: job control statements, which provide communication between the data base utility and OS/360, and function statements, which control specific operations performed by the utility. Each statement is entered on one or more control cards. Figure 31 shows the control card deck format for the data base utility.

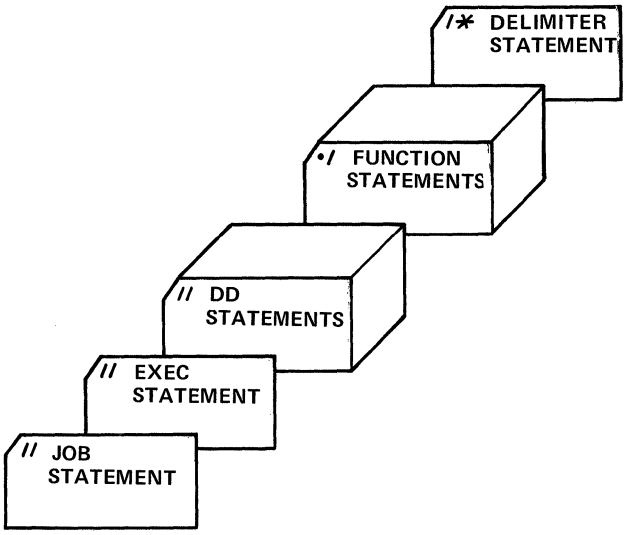


Figure 31. Data base utility control cards

Job Control Statements

The following is an example of the JCL statements required to execute the data base utility:

```
//ANYNAME      JOB     MSGLEVEL=1,REGION=96K
//JOBLIB       DD      DSN=OSRTS.JOBLIB,DISP=SHR
//STEPNAME     EXEC    PGM=DIBCADBU
//INDEX        DD      DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT     DD      SYSOUT=A
                  .
                  .
                  .
Additional DD statements
                  .
                  .
                  .
//SYSIN        DD      *
                  .
                  .
                  .
Function statements and any input program or data file in card form
                  .
                  .
                  .
/*
```

The additional DD statements required depend on the operation or function to be performed; these are summarized in the following text. The function statements and examples of their use with the appropriate DD statements are given in subsequent sections.

Whenever system group data sets are accessed or whenever the SYSLIB password is specified, one DD statement must be included for each data set in the system group in one or both clusters referenced in the job step. If only one cluster is referenced, the system group supplied must be from that cluster. The ddname is of the form:

SYSGRPnn	where nn is the relative data set number; system group data set numbering must begin with 00 in the primary cluster and 40 in the alternate cluster

Whenever user group data sets are accessed, one DD statement must be included for each of the data sets in each group accessed. The ddname is of the form:

aaabbbnn	where aaa and bbb are the first three characters of the upper and lower bound user numbers (the range of the user group) having access to the defined data set, and nn is the relative data set number; user group data set numbering must begin with 00 in the primary cluster and 40 in the alternate cluster

When certain functions are requested, one or more additional DD statements are required. The ddnames are of the form:

LIBRARY	Defines either the OS/360 partitioned data set which contains the programs to be added to the data base with the INSERT/REPLACE function or the OS/360 partitioned data set which is to be used as the output data set by the WRITE function
SYSPUNCH	Defines the punched output data set
TAPEIN	Defines the backup tape to be used as input
TAPEOUT	Defines the backup tape to be written
TAPEJ	Defines a tape journal to be written by the ACCOUNT function
PRINTJ	Defines a printed journal to be written by the ACCOUNT function
CBNDX	Defines the COBI index either when the JOBFIND function is used or when the DELETE function is used to purge COBI jobs from the data base
CBJCL	Defines the COBI JCL data set when the DELETE function is used to purge COBI jobs from the data base
name	Defines an OS/360 sequential data set to be converted to a CALL-OS data file or vice versa

An example of the job control language set up for a typical application of the data base utility is shown in Figure 32. A detailed explanation of the various statement entries can be obtained from the publication IBM System/360 Operating System: Job Control Language.

```
//JOBNAME      JOB     MSGLEVEL=1,REGION=96K
//JOBLIB       DD      DSN=OSRTS.JOBLIB,DISP=SHR
//             EXEC    PGM=DIBCADBU
//INDEX        DD      DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT     DD      SYSOUT=A
//SYSPUNCH     DD      SYSOUT=B
//SYSGRP00     DD      DSN=OSRTS.SYSGRP00,DISP=SHR
//SYSGRP01     DD      DSN=OSRTS.SYSGRP01,DISP=SHR
//AAAMMM00     DD      DSN=OSRTS.AAAMMM00,DISP=SHR
//AAAMMM01     DD      DSN=OSRTS.AAAMMM01,DISP=SHR
//NNNZZZ00     DD      DSN=OSRTS.NNNZZZ00,DISP=SHR
//NNNZZZ01     DD      DSN=OSRTS.NNNZZZ01,DISP=SHR
//LIBRARY      DD      DSN=PROGRAM.PDS,DISP=OLD,UNIT=2314,
//                     VOL=SER=MYPAK
//TAPEIN       DD      DSN=AAA.BACKUP,DISP=(,KEEP),UNIT=2400,
//                     VOL=SER=INPUT,LABEL=(,SL)
//TAPEOUT      DD      DSN=MMM.BACKUP,DISP=(,KEEP),UNIT=2400,
//                     VOL=SER=OUTPUT,LABEL=(,SL)
//TESTDATA     DD      DSN=MYDATA,DISP=OLD,UNIT=2314,
//                     VOL=SER=DATAPK
//SYSIN        DD      *
          .
          .
          .
   (function statements)
          .
          .
          .
/*
```

Figure 32. Data base utility JCL example

Utility Control Statements

Function statements are the data base utility control statements. These statements indicate which data base utility function is to be performed and specify parameters used to control processing. A function statement has up to four fields: identification, mnemonic, parameter, and comment, in that order.

The identification field is always columns 1 and 2 of the statement. These columns must contain ./ followed by at least one blank.

The mnemonic field identifies the function to be used and must contain one of the following values:

```
ACCOUNT
DELETE
INSERT
JOBFIND
RECON
REORG
REPLACE
TAPE
VALIDATE
WRITE
```

The mnemonic field must be followed by at least one blank.

The parameter field contains keyword parameters which control the specific process performed by the function. The parameters are separated by commas and may be extended up to and including column 71. At least one blank must follow the last parameter unless the last character appears in column 71.

The comment field is optional, and may not appear unless it is preceded by at least one parameter, followed by a blank. The comment field may extend up to and including column 71.

Function statements may be continued on as many cards as needed to hold the statement. The rules for continuing a function statement are:

1. The card to be continued must contain either:

   - Only the identification field (columns 1 and 2) and the mnemonic field somewhere in columns 4 through 71, or,

   - The first two fields and/or an incomplete parameter field (with or without a comment field); the last parameter must be followed by a comma, and, if the comma is not in column 71, at least one blank.

2. The continuation card must contain ./ in columns 1 and 2.

3. One or more blanks must appear between the ./ and the continued parameter field.

Note: When the PASSWORD or USERPASS parameter is specified on the control statement, printing of the associated value is suppressed unless the control statement contains an error. If the value following either the PASSWORD parameter or the USERPASS parameter contains embedded blanks and/or special characters, then the value must be enclosed in single quotes. If the value contains a single quote, the quote is represented by two consecutive single quotes. The following examples illustrate this point:

Password Value	Parameter Value
ABC,b489	'ABC,b489'
AAAb'ME'	'AAAb''ME'''
MMM#?	'MMM#?'

where b represents a blank

Analysis of the function statements involves only those parameters which are either required or optional for the specified function. If parameters other than those listed for a function are specified on a function statement, they are ignored. If parameters which are not valid for any function are specified, an error message is issued. Specific details on parameters and options for each function are given in the following sections.

ACCOUNT FUNCTION

Online accounting routines update only processor and terminal connect time; no accounting is made of disk usage by the online system. The ACCOUNT function of the data base utility gives the capability to perform offline accounting functions on the CALL-OS data base (intended for customer billing services).

This function provides for offline updates to the disk usage parameter, DISKUNIT, located in each user's equivalency entry. There are also provisions for accounting functions in which the equivalency records may be written to tape and/or the pertinent data extracted and listed on the printer.

Additional DD Statements

In addition to the required JCL statements described previously, the ACCOUNT function requires one or more system group DD statements and/or one or more user group DD statements. If a tape journal is requested, a TAPEJ DD statement is necessary. If a printer journal is requested, a PRINTJ DD statement is necessary; this statement should define the system output data set.

ACCOUNT Function Statement

The ACCOUNT function statement specifies the operations to be performed. The specific options available with the ACCOUNT function are described below.

1-2	Mnemonic	Parameters
./	ACCOUNT	USRGROUP={aaabbb|SYSGRP},CLUSTER=k,
		PASSWORD=xxxxxxxx
		[,OPTIONS=([TAPEJ,][PRINTJ,][RESET])]

Note: {} - select one from enclosed list.
[] - indicates optional parameter

USRGROUP=

aaabbb — specifies the user group on which the accounting function is to be performed; when this option is specified, one or more aaabbbnn DD statements must be supplied.

SYSGRP — specifies that the system group is to have the accounting function performed on it.

Note: Prior to this version of CALL-OS, accounting information was not maintained for the system group; instead, these fields (CPUTIME, TERMTIME, and DISKUNIT) were set to arbitrary high values. If it is desired to perform accounting for a system group created and/or used under a previous version of CALL-OS, the ACCOUNT function should be run first with OPTIONS=RESET to initialize the accounting fields in the SYSLIB equivalency entry.

CLUSTER=k

specifies the cluster number, either 1 or 2, of the user group or system group data set upon which accounting is to be done.

PASSWORD=xxxxxxxx

must specify the SYSLIB password; because this parameter is required, one or more system group DD statements must be supplied to provide password validation.

OPTIONS=( )

Notes: (1) If only one option is given, the parentheses may be omitted.

(2) These options may be used alone or in any combination.

TAPEJ	specifies that user equivalency information used for accounting is to be written to the tape journal data set defined by the TAPEJ DD statement.
PRINTJ	specifies that user equivalency information used for accounting is to be written to the printed journal data set defined by the PRINTJ DD statement.
RESET	specifies that user equivalency information used for accounting (CPUTIME, TERMTIME, and DISKUNIT) is to be reset to zero.

Accounting Options and Examples

The accounting operations performed depend on whether the OPTIONS parameter is absent or whether one or more options are specified. The options available are as follows:

- Basic accounting
- Tape journal
- Printed journal
- Resetting equivalency fields to zero

These options and suggestions for the use of the ACCOUNT function are described in greater detail in the following paragraphs.

Basic Accounting: Basic accounting is performed every time the ACCOUNT function is executed, whether or not the OPTIONS parameter is specified. Each user's catalog records are scanned for current disk usage and the total amount of space now being used is compared with the equivalency entry, DISKUNIT. If the DISKUNIT value is less than the current usage, it is set to the greater amount; purged space is not counted. The equivalency record is then rewritten.

An example of the JCL for execution of the ACCOUNT function with only the basic accounting operations to be performed follows:

```
//ACCOUNT     JOB     'J.CODER',MSGLEVEL=1,REGION=96K
//JOBLIB      DD      DSN=OSRTS.JOBLIB,DISP=SHR
//BASIC       EXEC    PGM=DIBCADBU
//INDEX       DD      DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT    DD      SYSOUT=A
//SYSGRP00    DD      DSN=OSRTS.SYSGRP00,DISP=OLD
//SYSGRP01    DD      DSN=OSRTS.SYSGRP01,DISP=OLD
//MEDSUR00    DD      DSN=OSRTS.MEDSUR00,DISP=OLD
//MEDSUR01    DD      DSN=OSRTS.MEDSUR01,DISP=OLD
//MEDSUR02    DD      DSN=OSRTS.MEDSUR02,DISP=OLD
//SYSIN       DD      *
./  ACCOUNT   USRGROUP=MEDSUR,CLUSTER=1,PASSWORD=COMMCON1
/*
```

Basic accounting operations are performed on user group MEDSUR in the primary cluster. The DISKUNIT field in the equivalency records is updated for all the users in the group. No journals are produced and no resetting is done.

Tape Journal: The tape journal option (TAPEJ) provides basic accounting, in addition to a daily journal tape. The tape journal contains the up-to-date equivalency entries for each user. When the

TAPEJ option is specified, a TAPEJ DD statement must be included in the JCL to define the data set. This DD statement may define either a tape data set or a physical sequential data set on disk.

The tape journal is written with fixed (F) format records and a blocksize of 7248 bytes. The first 36 bytes contain a copy of the key field from the updated equivalency record; note that the date on which the equivalency record was last updated reflects the date on which the ACCOUNT function was performed. The remaining 7212 bytes contain the updated equivalency information for the user or system group. For the system group, the only genuine user number is SYSLIB; however, the ACCOUNT function generates a pseudo-entry with a user number of PUBLIC. This entry contains the number of tracks used by the ** directory, with zeros in the other accounting fields. The pseudo-entry is included in the system group equivalency record written in the tape journal; however, it is not written into the equivalency file in the system group data set.

An example of the JCL for execution of the ACCOUNT function with the tape journal option follows:

```
//TAPEJOUR     JOB     'J.CODER',MSGLEVEL=1,REGION=96K
//JOBLIB       DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//STEP1        EXEC    PGM=DIBCADBU
//INDEX        DD      DSNAME=OSRTS.INDEX,DISP=SHR
//SYSPRINT     DD      SYSOUT=A
//TAPEJ        DD      DSN=DEVDEV,UNIT=2400,VOL=SER=OUTPUT,
//                     LABEL=(,SL),DISP=(MOD,KEEP)
//SYSGRP40     DD      DSN=OSRTS.SYSGRP40,DISP=OLD
//DEVDEV40     DD      DSN=OSRTS.DEVDEV40,DISP=OLD
//SYSIN        DD      *
./  ACCOUNT  USRGROUP=DEVDEV,CLUSTER=2,PASSWORD=CONSOLE1,
./           OPTIONS=TAPEJ
/*
```

The data base utility ACCOUNT function is performed on user group DEVDEV in the alternate cluster. Basic accounting is performed and a tape journal is produced from the equivalency records for the users in the group. Note that by specifying MOD on the TAPEJ DD statement, accounting information for more than one user group or more than one day's activities may be accumulated in the same data set.

Printed Journal: When the printed journal option (PRINTJ) is specified, basic accounting is performed and the equivalency entries are written out on the printer with no tape output. The printed output shows the current basic accounting information which includes updated disk usage. When the PRINTJ option is specified, a PRINTJ DD statement must be included in the JCL to define the printer data set. The format of the printed journal for a user group is as follows:

```
CALL-OS  CURRENT TOTALS  USER GROUP UCBUCB  MM/DD/YY        PAGE 1

USER NUMBER/CONNECT TIME/CPU TIME/DISK TRACKS/LAST VALID./LAST USE DATE

    UCB103      HH:MM       HH:MM:SS    TTTT.T    MM/DD/YY  MM/DD/YY
    UCB104      HH:MM       HH:MM:SS    TTTT.T    MM/DD/YY  MM/DD/YY
      .           .           .           .         .         .
      .           .           .           .         .         .
    UCB999      HH:MM       HH:MM:SS    TTTT.T    MM/DD/YY  MM/DD/YY
```

Time is reflected in the printed output in hours, minutes, and seconds. The number of disk tracks used is reflected to the smallest disk unit which is a tenth (.1) of a track.

The printed journal for a system group has the same format as for a user group. The user number for the system group is SYSLIB, and is followed by the accounting information. The ACCOUNT function generates a pseudo-entry whose user number is PUBLIC; this entry contains the number of tracks used by the ** directory and zeros in the other accounting fields. The pseudo-entry is included in the system group equivalency record written in the printed journal; however, it is not written into the equivalency file in the system group data set.

An example of the JCL for execution of the ACCOUNT function with the printed journal option follows:

```
//ACCOUNTP      JOB     'J.CODER',MSGLEVEL=1,REGION=96K
//JOBLIB        DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//STEP1         EXEC    PGM=DIBCADBU
//INDEX         DD      DSNAME=OSRTS.INDEX,DISP=SHR
//SYSPRINT      DD      SYSOUT=A
//PRINTJ        DD      SYSOUT=A
//SYSGRP00      DD      DSN=OSRTS.SYSGRP00,DISP=OLD
//SYSGRP01      DD      DSN=OSRTS.SYSGRP01,DISP=OLD
//UCBUCB00      DD      DSNAME=OSRTS.UCBUCB00,DISP=OLD
//UCBUCB01      DD      DSNAME=OSRTS.UCBUCB01,DISP=OLD
//SYSIN         DD      *
./  ACCOUNT      USRGROUP=UCBUCB,CLUSTER=1,PASSWORD=COMMCON1,
./               OPTIONS=PRINTJ
/*
```

Basic accounting is performed and a printed journal is produced for user group UCBUCB in the primary cluster.

Resetting Equivalency Fields to Zero: When RESET is specified, the equivalency entries used for accounting (CPUTIME, TERMTIME, and DISKUNIT) are reset to zero. Resetting occurs after the updated information has been written into the tape journal and/or the printed journal. Then the equivalency record is written back on the disk.

An example of the JCL for execution of the ACCOUNT function with the tape journal, printed journal, and reset options follows:

```
//ACCNTTP       JOB     'J.CODER',MSGLEVEL=1,REGION=96K
//JOBLIB        DD      DSNAME=OSRTS.JOBLIB,DISP=SHR
//STEP1         EXEC    PGM=DIBCADBU
//INDEX         DD      DSNAME=OSRTS.INDEX,DISP=SHR
//SYSPRINT      DD      SYSOUT=A
//PRINTJ        DD      SYSOUT=A
//TAPEJ         DD      DSNAME=JOURNAL,UNIT=2400,VOL=SER=OUTPUT,
//                      LABEL=(,SL),DISP=(MOD,KEEP)
//SYSGRP00      DD      DSN=OSRTS.SYSGRP00,DISP=OLD
//SYSGRP01      DD      DSN=OSRTS.SYSGRP01,DISP=OLD
//SYSGRP40      DD      DSN=OSRTS.SYSGRP40,DISP=OLD
//MEDSUR00      DD      DSNAME=OSRTS.MEDSUR00,DISP=OLD
//MEDSUR01      DD      DSNAME=OSRTS.MEDSUR01,DISP=OLD
//MEDSUR02      DD      DSNAME=OSRTS.MEDSUR02,DISP=OLD
//DEVDEV40      DD      DSNAME=OSRTS.DEVDEV40,DISP=OLD
//UCBUCB00      DD      DSNAME=OSRTS.UCBUCB00,DISP=OLD
//UCBUCB01      DD      DSNAME=OSRTS.UCBUCB01,DISP=OLD
//SYSIN         DD      *
./  ACCOUNT   USRGROUP=SYSGRP,CLUSTER=1,PASSWORD=COMMCON1,
./            OPTIONS=(TAPEJ,PRINTJ,RESET)
./  ACCOUNT   USRGROUP=UCBUCB,CLUSTER=1,PASSWORD=COMMCON1,
./            OPTIONS=(TAPEJ,PRINTJ,RESET)
./  ACCOUNT   USRGROUP=MEDSUR,CLUSTER=1,PASSWORD=COMMCON1,
./            OPTIONS=(TAPEJ,PRINTJ,RESET)
./  ACCOUNT   USRGROUP=SYSGRP,CLUSTER=2,PASSWORD=CONSOLE1,
```

```
./            OPTIONS=(TAPEJ,PRINTJ,RESET)
./  ACCOUNT   USRGROUP=DEVDEV,CLUSTER=2,PASSWORD=CONSOLE1,
./            OPTIONS=(TAPEJ,PRINTJ,RESET)
/*
```

A tape and printed journal is produced for user group DEVDEV in the alternate cluster, and for user groups UCBUCB and MEDSUR in the primary cluster. In addition, a tape and printed journal of the system group for each cluster is also printed. All equivalency entries are reset to zero.

Suggestions for Use of the ACCOUNT Function: The available ACCOUNT function options may be used in any combination to provide a variety of accounting services. For example, the TAPEJ option could be used daily to provide basic accounting information and a daily tape journal containing month-to-date statistics. This tape would serve as a backup copy for chargeable services if system malfunctions occurred the next day and prevented accounting from taking place. The PRINTJ, RESET, and/or TAPEJ options could be used monthly to provide monthly accounting information, including a printed journal of accounting information accumulated during the month, followed by resetting of the equivalency fields to zero.

However, this method has one disadvantage: the DISKUNIT field reflects the largest amount of disk storage used on any one day. If it is desired to bill users based on the average amount of disk storage used, or in a detailed manner based on day-to-day variations in disk storage used, the ACCOUNT function can be used to obtain this information on a day-to-day basis. For example, if the TAPEJ and RESET options are used daily, then the tape journal will contain accounting information for only that day. At the end of the month, these daily journals can be used as input to an installation-written accounting rouinte. This accounting routine can then total and average the amount of CPU time, terminal time, and disk storage used by each terminal user, as well as prepare the bills.

Whatever the accounting method used, some type of accounting should be performed every day. One way to ensure this is to execute the ACCOUNT function of the data base utility as a job step immediately following the job step which initiates CALL-OS. In this way, accounting is performed as soon as CALL-OS terminates.

DELETE FUNCTION

The DELETE function of the data base utility is used to remove any portion of a user's data base through the following capabilities:

- Cancel a user.
- Cancel a range of users.
- Purge one or all programs for a user, removing all directory references.
- Purge one or all data files for a user, removing all directory references.
- Purge one or all COBI jobs for a user.
- Purge all programs and all data files for a user, removing all directory references.

- Purge all programs, data files, and COBI jobs for a user, removing all directory references.
- Purge all user catalog entries, removing all directory references, for a range of users.
- For a user or range of users, purge all programs, data files, or both that have not been used since a specified date, unless the files are locked.
- Purge one or all ***Library entries.
- Pull directory (* or ** or both) entries for one or all programs or data files for a user or range of users.
- Pull directory (* or ** or both) entries regardless of user number associated.
- Pull directory (* or ** or both) entries on a time-since-last-used basis.
- Clean a directory by pulling all directory entries for which there are no corresponding active program or data files in the user catalog (for one user, for a range of users, or for a directory without regard to associated user number).
- Write a backup tape of deleted program(s) or data file(s).

Additional DD Statements

In addition to the required JCL statements described previously, the DELETE function requires, depending on the operation, one or more system group DD statements, and/or one or more user group DD statements. If a backup tape is to be written of the deleted items, a TAPEOUT DD statement must be supplied. If COBI job entries are to be purged from the data base, the CBNDX, CBJCL, and SYSPUNCH DD statements must also be supplied.

DELETE Function Statement

The DELETE function statement specifies the operations to be performed on the data base. The options available are described below.

1-2	Mnemonic	Parameters
./	DELETE	FROMUSER={aaanmm|SYSLIB|**LIB},
		[FROMUSR2=aaanpp,]FROMCLUS=k,
		COMMAND=([*CANCEL,][PURGE|PULL]),
		PASSWORD=xxxxxxxx,
		[FILE={PROG|DATA|BOTH|NULL|JOB|EVERY},]
		[NAME={filename|xxxxxxx.|(ALL)},]
		[DATE=yyddd,]
		[LANG={BASIC|FORTRAN|PL/I|DATA},]
		[OPTIONS=([UNLOCK,][NOTAPE,][*,][**])]

Note: {} - select one from enclosed list
[] - indicates optional parameter

FROMUSER=

aaanmm — specifies the only user number or the first of a range of user numbers to be cancelled and/or whose files are to be purged or pulled. When this option is specified, one or more aaabbbnn DD statements must be supplied to define the user group to which this user number or range of numbers belongs. When PULL is specified, a user number of the form aaan00 indicates that entries are to be pulled from the *Directory without regard to the user who pooled the entry.

SYSLIB — specifies that entries are to be purged from the ***Library.

**LIB — specifies that entries are to be pulled from the **Directory without regard to the user who pooled the entry.

FROMUSR2=aaanpp

specifies the last of a range of user numbers to be acted upon. This parameter is not needed if only one user is being acted upon. When FROMUSR2 is specified, FROMUSER must be specified as aaanmm and aaan must be the same for FROMUSER and FROMUSR2 and mm must be less than or equal to pp. FROMUSR2 must not be specified if FROMUSER is of the form aaan00 and PULL is specified.

FROMCLUS=k

specifies the cluster number, either 1 or 2, of the user group to be acted upon.

COMMAND=

*CANCEL — cancels the user number(s) specified by setting the password(s) to zero. *CANCEL may not be specified if FROMUSER=SYSLIB or **LIB.

PURGE — purges the catalog entries for the user(s) specified (if the appropriate password is given) unless the specified file is locked. As an entry for a program

is purged, any directory reference to the program is also removed. If the user has specified UNLOCK in the OPTIONS parameter, the file is unlocked, then purged. A user number of the form aaan00 is ignored, but all other users in the range specified by FROMUSER and FROMUSR2 are processed. A backup tape containing all purged programs and data files is created unless OPTIONS=(NOTAPE) is specified. The backup tape must be defined with a TAPEOUT DD statement. One backup tape data set is created for each DELETE control statement; a disposition of MOD cannot be used to put multiple data sets on one tape. However, multivolume data sets are permitted. The format of the tape is the same as that for the TAPE function so that the tape can be used as input to either the INSERT/REPLACE or RECONSTRUCT function to restore the purged program(s) or data file(s).

PULL — specifies that directory entries are to be pulled.

PASSWORD=xxxxxxxx

specifies the current password of the single currently-valid user specified by FROMUSER, when FROMUSR2 is not specified and *CANCEL is not specified. The password given on the control statement must match the password in the equivalency entry for the user number; if the passwords do not match no processing is done for this control statement.

specifies the SYSLIB password under the following conditions: if *CANCEL is specified or if the user has already been cancelled; if a range of users is being acted upon; or if SYSLIB, **LIB, or a *Library (aaan00) has been specified as the user. When this option is specified, one or more SYSGRPnn DD statements must be supplied.

FILE=

PROG — specifies that source and object program entries are to be purged or pulled from the data base. If purged, any directory references are removed.

DATA — specifies that data file entries are to be purged or pulled from the data base. If purged, any directory references are removed.

BOTH — specifies that both program and data file entries are to be purged or pulled from the data base. If purged, any directory references are removed.

NULL — specifies that all directory entries for which there are no corresponding program or data files in the user's catalog are to be pulled. PULL must be specified.

JOB — specifies that COBI job entries are to be purged from the data base. One SCRATCH control statement for the OS/360 utility IEHPROGM is generated for each data set associated with the COBI job entry purged. If this option is specified, a SYSPUNCH DD statement must be supplied.

EVERY — specifies that every program file, data file, and COBI job entry is to be purged from the data base for the user(s) specified. For purged files, any directory references are removed.

Notes: (1) Specifying FILE=EVERY and OPTIONS=UNLOCK with the *CANCEL and PURGE commands removes a user from the data base.

(2) When a single user number is specified, the FILE parameter defaults to FILE=BOTH; when a range of user numbers is specified, the FILE parameter defaults to FILE=EVERY.

(3) The FILE parameter should not be specified when *CANCEL is the only command given.

(4) When FILE=JOB or EVERY, the CBNDX, CBJCL, and SYSPUNCH DD statements must be supplied.

NAME=

filename — specifies the name of the program, data file, or COBI job entry to be purged or pulled from the catalog of the user(s) specified.

xxxxxxx. — specifies that only those files whose name begins with the characters indicated are to be deleted. From zero to seven characters may precede the period. The period indicates only that the characters specified are a prefix and is not used as a part of the prefix.

(ALL) — specifies that all files of the type indicated by the FILE parameter are to be purged or pulled for the user(s) specified. NAME=. is equivalent to NAME=(ALL).

Notes: (1) The NAME parameter defaults to NAME=(ALL) under the following conditions: when FILE=BOTH, EVERY, or NULL is specified; when a range of users is being purged; or when the DATE parameter is specified.

(2) The NAME parameter should not be specified when *CANCEL is the only command given.

(3) NAME=xxxxxxxx. should not be specified for COBI job entries.

DATE=yyddd

specifies a two-digit year number and three-digit day number for purging records on a time-since-last-used basis. For example, the notation for January 1, 1971 would be 71001. All entries of the type specified by the FILE parameter that have not been accessed on or after the date are purged for the user specified (unless the file is locked, and UNLOCK has not been specified in the OPTIONS parameter). Directory references are also pulled on a time-since-last-used basis.

Notes: (1) When a DATE parameter is specified, the NAME parameter defaults to NAME=(ALL).

(2) The DATE parameter should not be specified when *CANCEL is the only command given.

(3) The DATE parameter should not be specified when FILE=JOB or EVERY.

LANG={BASIC|FORTRAN|PL/I|DATA}

specifies the programming language of the programs or data files to be deleted. If the LANG parameter is not specified, programs and data files are processed without regard to programming language.

OPTIONS=( )

Note: If only one option is given, the parentheses may be omitted.

UNLOCK — specifies that the program or data file to be purged is to be unlocked before being purged. UNLOCK may be specified only when PURGE is specified.

Note: A program or data file with the LOCK attribute is not purged unless OPTIONS=UNLOCK has been specified.

NOTAPE — specifies that no backup tape is to be written. NOTAPE may be specified only when PURGE is specified; if NOTAPE is not specified, a backup tape is written containing each purged program or data file.

* and/or ** — specifies the directory or directories from which programs are to be pulled if a range of users or a single user (other than **LIB or a *Library) has been specified. If neither * nor ** is specified, it is assumed that entries are to be pulled from both directories. PULL must be specified when * and/or ** is specified.

Example

In the following example, all the entries in the data base associated with a user number are to be deleted and a backup tape is to be written of the deleted information.

```
//DELETE      JOB     MSGLEVEL=1,REGION=96K
//JOBLIB      DD      DSN=OSRTS.JOBLIB,DISP=SHR
//DEV468      EXEC    PGM=DIBCADBU
//INDEX       DD      DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT    DD      SYSOUT=A
//DEVDEV00    DD      DSN=OSRTS.DEVDEV00,DISP=SHR
//DEVDEV01    DD      DSN=OSRTS.DEVDEV01,DISP=SHR
//CBNDX       DD      DSN=OSRTS.CBNDX,DISP=SHR
//CBJCL       DD      DSN=OSRTS.CBJCL,DISP=SHR
//SYSPUNCH    DD      SYSOUT=B
//TAPEOUT     DD      DSN=BACKUP.DEV,DISP=(NEW,KEEP),UNIT=2400,
//                    LABEL=(,SL),VOL=SER=DEV468
//SYSIN       DD      *
./  DELETE  FROMUSER=DEV468,FROMCLUS=1,COMMAND=PURGE,PASSWORD=GRP2USR4,
./          FILE=EVERY,OPTIONS=UNLOCK
/*
```

The user whose number is DEV468 and whose password is GRP2USR4 is deleted from the primary cluster of the data base. Program files and data files are unlocked before being deleted and are written onto the backup tape defined by the TAPEOUT DD statement. In addition, each COBI job entry for that user is deleted and a SCRATCH control statement is produced for each data set associated with the COBI job.

INSERT/REPLACE FUNCTION

The INSERT/REPLACE function of the data base utility is used to make additions and changes to a user catalog through card, tape, or disk input. The INSERT/REPLACE function also allows program input in the form of members of a partitioned data set, and data file input in the form of a sequential data set.

A FORTRAN, PL/I, or BASIC program or data file in OS/360 format may be entered from cards into the user's data base as either a new entry or a replacement for an existing entry. BASIC source programs are entered into CALL-OS essentially unchanged. Each card must begin with a line number, starting in the first column of the left-hand margin (usually column 1). The line numbers must be in nondescending sequence. If two or more cards have the same line number, they are combined to form a single source line in CALL-OS. In this case, the second and succeeding cards in the combination must have a blank after the line number; source columns following the blank are added to the end of the source columns on the preceding card. When a FORTRAN source program is entered, the INSERT/REPLACE function reformats the program to CALL-OS free-form format by changing FORTRAN continuation and comment characters and adding a line number to the beginning of each generated line. PL/I source programs and DATA terminal files are entered into CALL-OS with the source data unchanged. A CALL-OS line number is added to the beginning of each generated line. For FORTRAN, PL/I, and DATA, the line number is either taken from columns 76 through 80 of the card sequence number or generated as specified. The sequence numbers must be in nondescending order; if two or more cards have the same sequence number, they are combined to form a single source line in CALL-OS. However, no editing for language differences is done by the INSERT/REPLACE function.

The INSERT/REPLACE function can be used to enter programs from tape. This facilitates reentrance of programs, data files, or entire user libraries or user ranges from a backup tape as well as transferral of users from one CALL-OS data base to another by tape.

A third use of the INSERT/REPLACE function is to transfer programs or data from one user catalog to another within the same data base. Entire user libraries or user ranges may be copied from one part of the data base to another, changing the user numbers in the process. In addition, directories may be updated, either using a backup tape or another part of the same direct-access data base.

Additional DD Statements

In addition to the required JCL statements described previously, the INSERT/REPLACE function requires, depending on the operation to be performed, one or more system group DD statements, and/or one or more user group DD statements. If programs from an OS/360 partitioned data set are to be used as input, a LIBRARY DD statement that defines the input data set must be supplied. If data files from a sequential data set are to be used as input, a DD statement that defines the data set must be supplied. If a backup tape is to be used as input, a TAPEIN DD statement must also be supplied; concatenation of backup tapes is not permitted. The DCB subparameter BUFNO=1 should be specified in the TAPEIN DD statement to run DIBCADBU in 96K; backup tape input with double buffering (the default) requires 100K.

INSERT/REPLACE Function Statement

The INSERT/REPLACE function statement specifies the operations to be performed. The specific options available with the INSERT/REPLACE function are described below.

1-2	Mnemonic	Parameters
./	INSERT or REPLACE	USER={aaanmm|SYSLIB|**LIB}, [USR2=aaanpp,]CLUSTER=k, [FROMUSER={bbbrqq|SYSLIB|**LIB},] [FROMCLUS=j,] [INPUT={CARD|OSDS|DISK|TAPE},] [LANG={BASIC|FORTRAN|PL/I|DATA},] PASSWORD=xxxxxxxx,[USERPASS=yyyyyyyy,] [FILE={PROG|DATA|BOTH},] [NAME={filename|xxxxxxx.|(ALL)},] [RENAME={newname|yyyyyyy.},] [MARG=(mm,nn),] [LINEGEN={SEQ,xxxxx},][LINEINC=xxxxx,] [OPTIONS=([SEQ,][LIST,][UNLOCK,][LOCK] [LONG,][PROTECT,][*,][**,] [OBJECT,][FORMDATA,][VAL]),] [SPACE=sss]

Note: {} - select one from enclosed list
[] - indicates optional parameter

INSERT
indicates that the input program or data file is to be inserted into the data base. If a program or data file exists in the user's catalog with the same name, the program or data file is not inserted.

REPLACE
indicates that the input program or data file is to be replaced in the data base. If a program or data file with the same name currently exists in the user's catalog, it is replaced. If no program or data file with the same name exists in the user's catalog, the input program or data file is inserted into the data base and a message is printed indicating that no previous file of that name existed.

USER=
aaanmm specifies the only user number or the first of a range of user numbers into whose catalog the input program or data is to go. When this option is specified, one or more aaabbbnn DD statements must be

supplied to define the user group to which this user number or range of user numbers belongs. The user number must be a validated user number unless validation is to be performed in conjunction with the INSERT/REPLACE operation in which case OPTIONS=VAL must be specified. A user number of the form aaan00 may be specified only when INPUT=DISK or TAPE; this type of user number specifies a *Directory into which entries are to be placed. These entries are to be taken from the *Directory specified by FROMUSER, which in this case must have the form bbbr00.

SYSLIB — specifies that the input program or data is to be put in the ***Library.

LIB — specifies that the **Directory entries from the opposite cluster or from tape are to be placed into the **Directory specified by the CLUSTER parameter. INPUT=DISK or TAPE must be specififed. When USER=LIB is specified, no FROMUSER or FROMCLUS should be specified.

USR2=aaanpp
specifies the last user in a range of users into whose catalogs programs and data files are to be inserted or replaced. USR2 need not be specified if only one user is intended. USR2 must not be specified if INPUT=CARD or OSDS. If USR2 is specified, USER must be specified as aaanmm and aaan must be the same for USER and USR2 and mm must be less than or equal to pp.

CLUSTER=k
specifies the cluster number, either 1 or 2, into which the program or data is to be placed.

FROMUSER={bbbrqq|SYSLIB|**LIB}
specifies the catalog or directory from which the program or data file is to be taken when INPUT=DISK or TAPE. When USER and USR2 specify a range of users, the last two digits of the FROMUSER number must match the last two digits of the USER number. When a FROMUSER is specified of the form bbbr00, the specified USER must have the form aaanmm; if mm is not equal to 00 and USR2 is not specified, then the aaan in the USER number must be the same characters as bbbr, that is, the USER must be in the subscription group served by the bbbr00 *Directory.

Note: The FROMUSER parameter must be specified when INPUT=TAPE or DISK, except when USER=**LIB.

FROMCLUS=j
specifies the cluster from which the program or data file is to be taken when INPUT=DISK.

Note: The FROMCLUS parameter must be specified when INPUT=DISK, except when USER=**LIB or when USER and FROMUSER specify the same user number (in these cases, FROMUSER is assumed to be in the opposite cluster from USER).

INPUT=

CARD — specifies that the program or data file to be inserted or replaced follows this control statement in the input stream (in card format). The delimiter for the input deck is either the next ./ control card or an EOF indication for the SYSIN data set. INPUT=CARD is the default.

OSDS — specifies, when FILE=PROG, that the input is the member whose name is given by the NAME parameter; this member is found in the partitioned data set defined by the LIBRARY DD statement. If NAME=(ALL), all the members of the data set are used as input.

specifies, when FILE=DATA, that the input is the OS/360 data set defined by the DD statement whose ddname is given by the NAME parameter. This must be a card-image data set, unless the FORMDATA option is specified, in which case it may have F, FB, V, or VB format, with logical records of 255 data bytes or less.

DISK — specifies that the programs or data files to be inserted or replaced are to be obtained from the user catalog(s) specified by FROMUSER and FROMCLUS (on CALL-OS disk data sets).

Note: When INPUT=DISK, the specified FROMUSER is not the same as the specified USER, and USR2 is not specified, a program or data file cannot be used as input unless the user in whose catalog the program or data file currently resides first releases the program or data file. However, if FROMUSER=SYSLIB or **LIB, then any program which has not been protected may be copied from the ***Library or **Library, respectively. The data base utility secures the file in the FROMUSER catalog after it has been transferred.

TAPE — specifies that the programs or data files to be inserted or replaced are to be obtained from a backup tape, under the user number specified by FROMUSER. If this option is specified, a TAPEIN DD statement must be supplied.

LANG={BASIC|FORTRAN|PL/I|DATA}
specifies the programming language applicable to the program or data file input. The default is LANG=BASIC for INPUT=CARD or OSDS. The LANG parameter may also be specified when INPUT=TAPE or DISK to indicate that only files of a specific language type are to be processed.

PASSWORD=xxxxxxxx
specifies the user password (of the user into whose catalog the program or data is to be placed) if only a single user is specified and RECORD=PROG, DATA, or BOTH.

specifies the SYSLIB password (for the specified cluster) either if OPTIONS=VAL, or if USER=SYSLIB, **LIB, or aaan00, or USER and USR2 are specified. When this option is specified, one or more SYSGRPnn DD statements must be supplied.

USERPASS=yyyyyyyy
specifies the password which is to be given to the user(s) which is to be validated as a result of specifying either OPTIONS=VAL or a range of users. The USERPASS parameter must be specified when OPTIONS=VAL or a range of users is specified.

FILE=
PROG — specifies that the input is a program file.

DATA — specifies that the input is a data file.

BOTH — specifies that the input is both program and data files.

Notes: (1) FILE=PROG is the default, unless a range of users is specified, in which case FILE=BOTH and OPTIONS=VAL is the default.

(2) FILE=BOTH may not be specified when INPUT=CARD or OSDS, since only one program or data file may be named on a single control statement.

NAME=

filename — specifies the single program or data file being input. If no RENAME parameter is specified, the filename specified by the NAME parameter is also assigned to the program or data file after it is inserted.

Notes: (1) If INPUT=CARD, this NAME is assigned to the file after it is inserted into the data base.

(2) If INPUT=OSDS and FILE=PROG, the NAME parameter specifies the member name to be inserted from the partitioned data set defined by the LIBRARY DD statement.

(3) If INPUT=OSDS and FILE=DATA, the NAME parameter specifies the name of the DD statement which defines a sequential data set to be used as input.

(4) If INPUT=DISK or TAPE, the NAME parameter specifies the file to be transferred from the FROMUSER library.

xxxxxxx. — specifies that only those files whose name begins with the characters indicated are transferred if INPUT=DISK, TAPE, or OSDS and FILE=PROG. A maximum of seven characters in addition to the period may be specified. The period indicates only that the characters specified are a prefix and is not used as a part of the prefix.

(ALL) — specifies that all records of the type indicated by the FILE parameter are to be inserted or replaced into the specified user(s) catalog. If NAME=(ALL) is specified for program input when INPUT=OSDS, all members of the partitioned data set specified by the LIBRARY DD card are inserted. NAME=(ALL) cannot be specified when INPUT=CARD or when INPUT=OSDS and FILE=DATA. NAME=. is equivalent to NAME=(ALL).

Note: NAME=(ALL) is the default when a range of users is specified; this is the only case in which the NAME parameter may be omitted.

RENAME=

newname — specifies the new name to be assigned to the program or data file being inserted. If omitted, the filename specified by the NAME parameter is assigned to the program or data file being inserted.

yyyyyyy.
: specifies a prefix of up to seven characters which is added to the name of the program or data file after it is inserted. Resulting filenames which exceed eight characters are truncated on the right. If the NAME parameter also specifies a prefix, the prefix specified by RENAME replaces the prefix specified by NAME in the transferred file; if RENAME=. is specified, the prefix specified by NAME is omitted from the result.

Notes: (1) If the NAME parameter specifies a prefix or NAME=(ALL), RENAME may be omitted or may specify a prefix, but may not specify a unique new name.

(2) If an embedded / would otherwise appear in the resulting newname, the / is removed from the name.

(3) If the resulting newname would otherwise begin with a numeric character (due to the deletion of leading characters when RENAME=. is specified), then a # is added to the beginning of the name.

MARG=(mm,nn)
: specifies the card margins for the input, where mm specifies the left margin and nn specifies the right margin. The numbers mn and nn may be one or two decimal digits. The left margin must be greater than or equal to 1 and less than the specified right margin. The right margin must be less than or equal to 80. Only the data between and including the card margins is put into the data base. The MARG parameter is applicable only when INPUT=CARD or OSDS (and in the latter case, only when the data set is in card-image format).

Notes: (1) For BASIC source programs, the default is MARG=(1,80).

(2) For FORTRAN source programs, for all object programs, and for data files which are being input without using the FORMDATA option, the standard input has MARG=(1,72) and may not be overridden by using the MARG parameter.

(3) For PL/I source programs, the default is MARG=(2,72).

(4) For DATA terminal files and data files being input in card-image format using the FORMDATA option, the default is MARG=(1,80).

LINEGEN=
SEQ
: specifies that line numbers are to be generated from the sequence numbers contained in columns 76 through 80.

xxxxx
: specifies the initial number of the line numbers that are to be generated. The default value is 00100.

LINEINC=xxxxx
: specifies the value by which the line numbers are to be incremented. The default value is 00010.

Notes: (1) The LINEGEN and LINEINC parameters are valid only for FORTRAN and PL/I source programs, and only when INPUT=CARD or OSDS is specified. Line numbers will not be generated for BASIC programs, for object programs, or for data files.

(2) If the LINEGEN and LINEINC parameters are omitted, the default is an initial line number of 00100 incremented by 00010.

(3) The LINEINC parameter should not be specified when LINEGEN=SEQ is specified.

OPTIONS=( )

Note: If only one option is given, the parentheses may be omitted.

SEQ — specifies that sequence numbers on the input statements are to be checked. The sequence numbers must be contained in columns 73 through 80 of the input card-image records. If SEQ is specified, any statement out of sequence is flagged and not entered into the data base. If SEQ is not specified, no sequence checking is performed.

Notes: (1) No sequence checking is performed when LANG=BASIC.

(2) The SEQ option may be specified only when INPUT=CARD or OSDS.

LIST — specifies that the input is to be listed on the printer as it is entered from cards or from an OS/360 data set. The LIST option may be specified only when INPUT=CARD or OSDS.

UNLOCK — specifies that if the program or data being replaced in the data base is locked, it is to be unlocked, then replaced. The UNLOCK option is ignored on an INSERT function.

LOCK — specifies that the program or data being entered is to be given the LOCK attribute (that is, the entry cannot be purged unless first unlocked).

LONG — specifies that, for the BASIC source program being entered, arithmetic is long precision.

PROTECT — specifies that the program is to be run-only (that is, it cannot be listed or saved by anyone except the owner of the program).

* and/or ** — specifies that the file being entered is to be pooled in the *Library and/or **Library, respectively.

OBJECT — specifies that the program file(s) being entered is a CALL-OS object program(s).

VAL — specifies that the user(s) indicated by the USER (and USER2) parameter is to be validated before the INSERT or REPLACE function is performed. When this option is specified, the VALIDATE function is executed before the INSERT/REPLACE function.

FORMDATA — specifies that a CALL-OS external format data file is to be generated from input data which does not already contain the various control bytes used in such files. Either LANG=FORTRAN or PL/I must be specified when the FORMDATA option is given; the generated data file conforms to the external data format used by the specified language. The FORMDATA

option should not be specified when the input data already contains all necessary control bytes to make it a valid CALL-OS data file (either internal format or external format.) The FORMDATA option may be specified only when INPUT=CARD or OSDS.

SPACE=sss
specifies the maximum number of file units (half tracks) which may be allocated to a data file which is being inserted or replaced, where 1 ≤ sss ≤ 100. Whenever INPUT=CARD or OSDS and FILE=DATA are specified, the default is SPACE=4. If the SPACE parameter is specified when INPUT=DISK or TAPE, it is used to override the maximum file space allocations of any data files being inserted or replaced whose existing allocation is smaller than sss. Only the number of file units given by the SPACE parameter (default or specified) are allocated for the input. Any remaining input is ignored.

Example

In the following example, program files from an OS/360 partitioned data set and data files from OS/360 sequential data sets are to be added to the data base.

```
//REPLACE     JOB     MSGLEVEL=1,REGION=96K
//JOBLIB      DD      DSN=OSRTS.JOBLIB,DISP=SHR
//INS305      EXEC    PGM=DIBCADBU
//INDEX       DD      DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT    DD      SYSOUT=A
//INSINS40    DD      DSN=OSRTS.INSINS40,DISP=SHR
//LIBRARY     DD      DSN=INS.PDSIN,DISP=OLD,UNIT=2314,VOL=SER=INS204
//DATAIN      DD      DSN=INS.DATA1,DISP=OLD,UNIT=2314,VOL=SER=INS204
//            DD      DSN=INS.DATA2,DISP=OLD,UNIT=2314,VOL=SER=INS204
//            DD      DSN=INS.DATA3,DISP=OLD,UNIT=2314,VOL=SER=INS204
//SYSIN       DD      *
./  REPLACE  USER=INS305,CLUSTER=2,INPUT=OSDS,LANG=PL/I,
./           PASSWORD=305MMW,FILE=PROG,NAME=(ALL),
./           OPTIONS=(SEQ,LIST,UNLOCK)
./  REPLACE  USER=INS305,CLUSTER=2,INPUT=OSDS,LANG=PL/I,
./           PASSWORD=305MMW,FILE=DATA,NAME=DATAIN,
./           OPTIONS=(LIST,UNLOCK,FORMDATA)
/*
```

Program and data file input is used to replace entries in the alternate cluster for the user whose number is INS305 and whose password is 305MMW. All of the members in the OS/360 partitioned data set INS.PDSIN are used as input; the members are PL/I programs and are sequence checked and listed as they are read in. The input OS/360 data files reside in three sequential data sets, concatenated with DD statement DATAIN; the three files are converted to one CALL-OS data file which is listed as it is read in. If the program or data file to be replaced is locked, it is unlocked and then replaced. If the program or data file being read in does not have a matching entry in the data base, it is added to the data base and a message is issued.

JOBFIND FUNCTION

When a job is submitted through COBI, an entry for that job is created in the user's catalog, as well as the COBI index. The JOBFIND function of the data base utility is used to update user catalogs with respect to current COBI job entries the user may have in the COBI index. Any user catalog entries that refer to a COBI job which is no longer

current are eliminated. Also, user catalog entries are created for current COBI jobs which did not previously have a user catalog entry.

The JOBFIND function should be used after each REORGANIZE or RECONSTRUCT function to ensure that user catalogs contain the user's current COBI job entries. Similarly, if the installation switches from a user group in the primary cluster to a backup copy of that user group in the alternate cluster (or vice versa), the JOBFIND function should be run on the backup copy; this updates the backup copy with respect to COBI jobs currently in the COBI index.

Additional DD Statements

In addition to the required JCL statements described previously, the JOBFIND function requires one or more system group DD statements, and may also require one or more user group DD statements which define the user group whose catalog records are to be updated. The COBI index data set must be defined with a CBNDX DD statement.

JOBFIND Function Statement

The JOBFIND function statement specifies the user group catalogs to be updated. The specific options available with the JOBFIND function are described below.

1-2	Mnemonic	Parameters
./	JOBFIND	USRGROUP={aaabbb|SYSGRP},CLUSTER=k,
		PASSWORD=xxxxxxxx

Note: {} - select one from enclosed list.

USRGROUP=

aaabbb — specifies the user group in which user catalogs are to be updated to reflect current COBI job status; when this option is specified, one or more aaabbbnn DD statements must be supplied.

SYSGRP — specifies that the system group catalog is to be updated to reflect current COBI job status.

CLUSTER=k

specifies the cluster number, either 1 or 2, of the group to be processed.

PASSWORD=xxxxxxxx

must specify the SYSLIB password; because this parameter is required, one or more SYSGRPnn DD statements must be supplied to provide validation of this password.

Example

In the following example, the user catalogs are updated with respect to current COBI job entries in the COBI index.

```
//JOBFIND     JOB     MSGLEVEL=1,REGION=96K
//JOBLIB      DD      DSN=OSRTS.JOBLIB,DISP=SHR
//IBMJOB      EXEC    PGM=DIBCADBU
```

```
//INDEX        DD      DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT     DD      SYSOUT=A
//CBNDX        DD      DSN=OSRTS.CBNDX,DISP=SHR
//SYSGRP00     DD      DSN=OSRTS.SYSGRP00,DISP=SHR
//SYSGRP01     DD      DSN=OSRTS.SYSGRP01,DISP=SHR
//IBMIBM00     DD      DSN=OSRTS.IBMIBM00,DISP=SHR
//IBMIBM01     DD      DSN=OSRTS.IBMIBM01,DISP=SHR
//SYSIN        DD      *
./  JOBFIND  USRGROUP=IBMIBM,CLUSTER=1,PASSWORD=COMMCON1
/*
```

The catalogs for user group IBMIBM in the primary cluster are updated to reflect the current status of the COBI index. The SYSGRPnn DD statements are required for validation of the SYSLIB password.

RECONSTRUCT FUNCTION

The RECONSTRUCT function of the data base utility gives the capability to recreate a system group or to selectively recreate a user group from the information recorded on a backup tape. A backup tape may be produced by the TAPE and DELETE functions of this utility, and may also be produced by the CALL/360: Standalone System backup program.

Note: Before the RECONSTRUCT function is executed, the data sets involved must be preformatted by U#UTIL1. Before U#UTIL1 can function properly, entries for the data sets must be removed from the index by deleting them with UTILX. These utilities are described earlier in this chapter.

Additional DD Statements

In addition to the required JCL statements described previously, the RECONSTRUCT function requires one or more DD statements that define either a system group or a user group which is to be recreated from the backup tape. The backup tape itself must be defined by a TAPEIN DD statement; concatenation of backup tapes is not permitted.

RECONSTRUCT Function Statement

Either a system group or a user group is recreated depending on the specification in the RECONSTRUCT function statement. The specific options which are available with the RECONSTRUCT function are described below.

1-2	Mnemonic	Parameters
./	RECON	USRGROUP={aaabbb|SYSGRP},CLUSTER=k

Note: {} - select one from enclosed list

USRGROUP=

aaabbb — specifies the user group which is to be recreated from the backup tape; when this option is specified, one or more aaabbbnn DD statements must be supplied. The RECON function statement may be followed by range cards that specify which user numbers are to be selected from the backup tape for the new user group. If the range cards are omitted, a default range is used; this range includes the entire group specified by the USRGROUP parameter.

SYSGRP
: specifies that the system group is to be recreated from the backup tape; when this option is specified, one or more SYSGRPnn DD statements must be supplied. The RECON function statement may be followed by range cards that specify the user numbers whose pooled programs and data files are to be included in the reconstructed **Directory. If the range cards are omitted, the **Directory contains no entries.

CLUSTER=k
: specifies the cluster number, either 1 or 2, of the user group or system group data set which is to be produced by reconstruction. This does not have to be the same cluster number as the cluster from which the backup tape was made.

Range Cards

User numbers within a range of user numbers may be selected for transfer from a backup tape to the data base. One execution of the RECONSTRUCT function accepts up to 200 ranges. These ranges are kept in a sorted table, and each tape record is checked against this table to see if it should be moved into the user group.

The ranges are specified on one or more range cards with the following format.

1-2	4-15	17-28	30-41	43-54	56-67	69-80
/$	[range]	[range]	[range]	[range]	[range]	[range]

Note: [] indicates optional parameter.

range
: specifies a twelve-character user number range of the form aaammmbbbnnn, where aaammm and bbbnnn are user numbers, with aaammm less than or equal to bbbnnn in the collating sequence. A maximum of 200 ranges may be specified on as many cards as are needed. There must be one blank between consecutive range fields and any field may be left blank. If a twelve-byte range field is blank, it is ignored. The ranges specified must be nonoverlapping ranges within the user group specified on the RECON control statement.

Another ./ function statement or an end-of-file ends range card reading. If no range cards are provided, a range equal to the output user group range is put into the range table, except for SYSGRP runs, in which case the range table is left empty.

Using a Backup Tape

The following text describes the format and processing of a backup tape; specific differences between the backup tape formats for CALL-OS and the CALL/360: Standalone system are noted at the end of this section.

Backup Tape Format: The backup tape is written in a user-number oriented sequence. Therefore, system group (SYSGRP) records have special pseudo user numbers which begin with three asterisks (***). All *** material is located in the first portion of the backup tape.

The tape is written in a variable-length blocked format, where the maximum block size is 7404 bytes, and the maximum logical record is 7400 bytes. Each logical record corresponds to a record from the data base. Only the useful information from each disk record is transferred to tape, so the tape records are variable in length. Equivalency records are a full track; catalog, directory, object program, and data records are a half track; source program records vary in size from a tenth track to a full track. All records except the equivalency records contain a 36-byte key field.

Records on a backup tape appear in the order shown in Figure 33.

```
User No.    Records

***010      CONTROL RECORD (PRESENT ON STANDALONE SYSTEM BACKUP
            TAPES ONLY)
***700      ALL EQUIVALENCY RECORDS
***800      PUBLIC (**) DIRECTORY
***900      SYSTEM (***) CATALOG
***900      SYSTEM (***) PROGRAMS AND DATA FILES
AAA000      FIRST POSSIBLE SUB GROUP DIRECTORY
AAA001      CATALOG                          FIRST POSSIBLE USER
AAA001      PROGRAMS AND DATA FILES
.
.
.
ZZZ999      CATALOG                          LAST POSSIBLE USER
ZZZ999      PROGRAMS AND DATA FILES
END OF FILE
```

Figure 33. Order of records on a backup tape

Backup Tape Processing: Two modes of operation are used during backup tape processing: system or user, depending upon the USRGROUP specification on the RECON function statement.

In system mode, the backup tape need only be read until the first user number for any user appears. In other words, the processing is ended when a user number without *** in the first three characters is encountered.

In user mode, processing ends when a user number higher than the highest user number in the output group range is read. For example, if an output user group had a range of TTT to ZZZ, the complete backup tape (or set of tapes) would have to be scanned and processed.

Notes: The following notes are made with reference to the RECONSTRUCT function as performed on a backup tape:

1. Equivalency records on the backup tape do not have a disk key field. They are up to a full track in length and contain up to 130 entries, each 56 bytes long. The first entry starts at byte 20.

2. In system mode, the system directory is created in the output group. However, the catalog pointers within this directory are not initialized. They are set up during the startup of CALL-OS.

   In user mode complete directories are built. When a directory is read, it is structured into an output buffer. It stays there until the next directory is read, at which time all of the catalogs which should be pointed to by the first directory have

been processed. The catalog processing has set the disk addresses for the catalogs into the equivalency file. The equivalency file is scanned for user numbers which appear in the directory, and the catalog disk addresses are transferred from the equivalency file to the directory. The directory is written to disk, and the next directory is moved into the output buffer to await catalog addresses.

3. The user number in a directory entry must be within one of the ranges in the range table, or it will not be moved to CALL-OS. This applies to both system and user runs. In user mode, the range table is also used to determine if equivalency entries, catalog records, program records, and data records should be included in the reconstructed user group.

   Files other than the user (sub group) directories are not as complex to handle. They are read from the tape and moved to disk with only certain flags and lengths in their catalog entries altered to be compatible with CALL-OS, if necessary.

4. The backup tape has no file descriptor records since all records for a given program or data file are adjacent on the tape. The RECONSTRUCT function creates file descriptors for all object programs and for any data files containing more than four records.

5. The CALL/360: Standalone System provides BASIC, PL/I, and FORTRAN language time sharing and is similar in some ways to the CALL-OS system. However, the two data bases are somewhat different in nature. The CALL/360: Standalone system has a data base made up of full tracks for equivalency files and half tracks for the remainder of the data base. CALL-OS has a data base made up of tenth, fifth, half, and full tracks. Therefore, while the maximum source program record on the standalone system is a half track, a source program record for CALL-OS may be up to a full track in length.

Example

In the following example, a backup tape is used to recreate a user group data set in the primary cluster.

```
//JOBRECON JOB      'J.CODER',MSGLEVEL=1,REGION=96K
//JOBLIB   DD       DSN=OSRTS.JOBLIB,DISP=SHR
//STEP1    EXEC     PGM=DIBCADBU
//INDEX    DD       DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT DD       SYSOUT=A
//TAPEIN   DD       DSNAME=BACKUP,DISP=OLD,VOLUME=SER=(DAY1,DAY1A),
//                  UNIT=2400
//DEVDEV00 DD       DSNAME=OSRTS.DEVDEV00,DISP=OLD
//SYSGRP00 DD       DSN=OSRTS.SYSGRP00,DISP=OLD
//CBNDX    DD       DSN=OSRTS.CBNDX,DISP=OLD
//SYSIN    DD       *
./  RECON  USRGROUP=DEVDEV,CLUSTER=1
/$ DEV222DEV333
/$ DEV770DEV772 DEV780DEV785 DEV790DEV792 DEV795DEV795 DEV710DEV720
/$              DEV733DEV733 DEV550DEV560
/$ DEV055DEV063 DEV010DEV010
./ JOBFIND  USRGROUP=DEVDEV,CLUSTER=1,PASSWORD=SECURITY
/*
```

This example assumes that data set OSRTS.DEVDEV00 is in the preformatted state. If this is not the case, then JCL similar to that

shown in the examples for the REORGANIZE function must be used to execute UTILX and U#UTIL1.

REORGANIZE FUNCTION

The REORGANIZE function of the data base utility is used to reorganize the system group or one or more user groups of one cluster of the data base into the system group or a user group (respectively) of the other cluster. In this reorganization, purged space is eliminated, and the new system or user group may have a different number of data sets of different sizes from the original group. Also, a new user group may be an extraction from, or a combination of, the original user group(s).

Note: Before the REORGANIZE function is executed, the data sets involved must be preformatted by U#UTIL1. Before U#UTIL1 can function properly, entries for the data sets must be removed from the index by deleting them with UTILX. These utilities are described earlier in this chapter.

Additional DD Statements

In addition to the required JCL statements described previously, the REORGANIZE function requires one or more DD statements (for both clusters) for each system or user group to be reorganized.

REORGANIZE Function Statement

The new system or user group which is to result from the reorganization is specified on the REORGANIZE function statement, along with the cluster to which the new group belongs. The input for each reorganization may contain more than one system or user group; however, only one group may be specified on the REORGANIZE function statement. A complete reorganization of the "to" cluster requires as many function statements as there are system and user groups in that cluster.

The specific options which are available with the REORGANIZE function are described below.

1-2	Mnemonic	Parameters
./	REORG	USRGROUP={aaabbb|SYSGRP},CLUSTER=k

Note: {} - select one from enclosed list.

USRGROUP=

aaabbb — specifies the new user group which is to result from the processing; when this option is specified, one or more aaabbbnn DD statements must be supplied.

SYSGRP — specifies that a new system group is to result from the processing; when this option is specified, one or more SYSGRPnn DD statements must be supplied.

CLUSTER=k

specifies the cluster number, either 1 or 2, of the new user group or system group data set.

Examples

The following example reorganizes a user group with four data sets into a user group with only one data set. The reorganization is from the primary cluster to the alternate. The four input data sets are REGREG00 through REGREG03, since, by convention, the data sets of the primary cluster have ddnames which end with numbers in the range 00 through 39. REGREG40 designates the single data set in the alternate cluster. The user number range for both the input user group and the output user group is REG000 to REG999.

```
//JOBREORG JOB      'J.CODER',MSGLEVEL=1,REGION=96K
//JOBLIB   DD       DSN=OSRTS.JOBLIB,DISP=SHR
//STEP1    EXEC     PGM=DIBCADBU
//INDEX    DD       DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT DD       SYSOUT=A
//REGREG00 DD       DSN=OSRTS.REGREG00,DISP=OLD
//REGREG01 DD       DSN=OSRTS.REGREG01,DISP=OLD
//REGREG02 DD       DSN=OSRTS.REGREG02,DISP=OLD
//REGREG03 DD       DSN=OSRTS.REGREG03,DISP=OLD
//REGREG40 DD       DSN=OSRTS.REGREG40,DISP=OLD
//SYSIN    DD       *
./ REORG  USRGROUP=REGREG,CLUSTER=2
/*
```

The next example shows system group reorganization. In this case, the reorganization is from the alternate cluster to the primary. The data is copied from the single data set SYSGRP40 to the SYSGRP00 and SYSGRP01 data sets.

```
//JOBREORG JOB      'J.CODER',MSGLEVEL=1,REGION=96K
//JOBLIB   DD       DSN=OSRTS.JOBLIB,DISP=SHR
//STEP1    EXEC     PGM=DIBCADBU
//INDEX    DD       DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT DD       SYSOUT=A
//SYSGRP00 DD       DSN=SYSGRP00,DISP=OLD
//SYSGRP01 DD       DSN=SYSGRP01,DISP=OLD
//SYSGRP40 DD       DSN=SYSGRP40,DISP=OLD
//SYSIN    DD       *
./REORG  USRGROUP=SYSGRP,CLUSTER=1
/*
```

The two preceding examples assume that the system and user group data sets are in the preformatted state. The following example shows how to preformat data sets with the UTILX and U#UTIL1 utility programs. In addition, the REORGANIZE function is used to backup and recover data sets.

Backup is defined as copying the data base from the primary cluster to the alternate cluster. Recovery is defined as copying the data base from the alternate cluster to the primary cluster. In the execution of UTILX in STEP1, note that no index deletion is specified for the IBMIBM user group. This is because the data set for the alternate cluster is newly created by U#UTIL1 in STEP2 (the disposition parameter on the IBMIBM DD statement specifies NEW).

The following example shows the backup of the IBMIBM and JBMJBM user groups, followed by the recovery of the reorganized data sets immediately.

```
//BACKUP   JOB      'J.CODER',MSGLEVEL=1,REGION=96K
//JOBLIB   DD       DSN=OSRTS.JOBLIB,DISP=SHR
//STEP1    EXEC     PGM=UTILX
//INDEX    DD       DSN=OSRTS.INDEX,DISP=SHR
//SYSPUNCH DD       DUMMY
//SYSPRINT DD       SYSOUT=A
//SYSIN    DD       *
DEL
   180,040,JBMJBM,JBMJBM40,OSRTS.JBMJBM40
/*
//STEP2    EXEC     PGM=U#UTIL1,PARM='USRGROUP'
//SYSPRINT DD       SYSOUT=A
//INDEX    DD       DSN=OSRTS.INDEX,DISP=SHR
//IBMIBM40 DD       DSN=OSRTS.IBMIBM40,DISP=(NEW,CATLG),UNIT=2314,
//                  VOL=SER=RTOS01,SPACE=(CYL,(30)),DCB=DSORG=DA
//JBMJBM40 DD       DSN=OSRTS.JBMJBM40,DISP=OLD
//STEP3    EXEC     PGM=DIBCADBU
//INDEX    DD       DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT DD       SYSOUT=A
//IBMIBM00 DD       DSN=OSRTS.IBMIBM00,DISP=OLD
//IBMIBM01 DD       DSN=OSRTS.IBMIBM01,DISP=OLD
//IBMIBM02 DD       DSN=OSRTS.IBMIBM02,DISP=OLD
//JBMJBM00 DD       DSN=OSRTS.JBMJBM00,DISP=OLD
//JBMJBM01 DD       DSN=OSRTS.JBMJBM01,DISP=OLD
//JBMJBM02 DD       DSN=OSRTS.JBMJBM02,DISP=OLD
//IBMIBM40 DD       DSN=OSRTS.IBMIBM40,DISP=OLD
//JBMJBM40 DD       DSN=OSRTS.JBMJBM40,DISP=OLD
//SYSIN    DD       *
./  REORG  USRGROUP=IBMIBM,CLUSTER=2
./  REORG  USRGROUP=JBMJBM,CLUSTER=2
/*
//STEP4    EXEC     PGM=UTILX
//INDEX    DD       DSN=OSRTS.INDEX,DISP=SHR
//SYSPUNCH DD       DUMMY
//SYSPRINT DD       SYSOUT=A
//SYSIN    DD       *
DEL
   160,000,IBMIBM,IBMIBM00,OSRTS.IBMIBM00
   160,001,IBMIBM,IBMIBM01,OSRTS.IBMIBM01
   160,002,IBMIBM,IBMIBM02,OSRTS.IBMIBM02
   160,000,JBMJBM,JBMJBM00,OSRTS.JBMJBM00
   160,001,JBMJBM,JBMJBM01,OSRTS.JBMJBM01
   160,002,JBMJBM,JBMJBM02,OSRTS.JBMJBM02
/*

//RECOVERY JOB      'J.CODER',MSGLEVEL=1,REGION=96K
//JOBLIB   DD       DSN=OSRTS.JOBLIB,DISP=SHR
//STEP5    EXEC     PGM=U#UTIL1,PARM='USRGROUP'
//SYSPRINT DD       SYSOUT=A
//INDEX    DD       DSN=OSRTS.INDEX,DISP=SHR
//IBMIBM00 DD       DSN=OSRTS.IBMIBM00,DISP=OLD
//IBMIBM01 DD       DSN=OSRTS.IBMIBM01,DISP=OLD
//IBMIBM02 DD       DSN=OSRTS.IBMIBM02,DISP=OLD
//JBMJBM00 DD       DSN=OSRTS.JBMJBM00,DISP=OLD
//JBMJBM01 DD       DSN=OSRTS.JBMJBM01,DISP=OLD
//JBMJBM02 DD       DSN=OSRTS.JBMJBM02,DISP=OLD
//STEP6    EXEC     PGM=DIBCADBU
//INDEX    DD       DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT DD       SYSOUT=A
//IBMIBM00 DD       DSN=OSRTS.IBMIBM00,DISP=OLD
//IBMIBM01 DD       DSN=OSRTS.IBMIBM01,DISP=OLD
//IBMIBM02 DD       DSN=OSRTS.IBMIBM02,DISP=OLD
//IBMIBM40 DD       DSN=OSRTS.IBMIBM40,DISP=OLD
//JBMJBM00 DD       DSN=OSRTS.JBMJBM00,DISP=OLD
//JBMJBM01 DD       DSN=OSRTS.JBMJBM01,DISP=OLD
```

```
//JBMJBM02 DD      DSN=OSRTS.JBMJBM02,DISP=OLD
//JBMJBM40 DD      DSN=OSRTS.JBMJBM40,DISP=OLD
//SYSGRP00 DD      DSN=OSRTS.SYSGRP00,DISP=OLD
//CBNDX    DD      DSN=OSRTS.CBNDX,DISP=OLD
//SYSIN    DD      *
./  REORG  USRGROUP=IBMIBM,CLUSTER=1
./  REORG  USRGROUP=JBMJBM,CLUSTER=1
/*
```

If at a later date, it becomes necessary to recover the data base, the job consists of STEP4 from the BACKUP job and the entire RECOVERY job with the addition of the following JOBFIND control statements after the REORG statements:

```
./  JOBFIND  USRGROUP=IBMIBM,CLUSTER=1,PASSWORD=SECURITY
./  JOBFIND  USRGROUP=JBMJBM,CLUSTER=1,PASSWORD=SECURITY
```

TAPE FUNCTION

The TAPE function of the data base utility is used to write a backup tape of a user, subscription group, user group, system group, or user group/system group cluster. Each backup tape contains all appropriate user numbers and passwords, catalogs, directories, file descriptors, programs, and data files. The format of the backup tape is compatible with the backup tapes obtained from the CALL/360: Standalone System. These backup tapes can be used as input to the INSERT/REPLACE function and the RECONSTRUCT function of the data base utility.

Additional DD Statements

In addition to the required JCL statements described previously, the TAPE function requires, depending on the operation to be performed, one or more system group DD statements and/or one or more user group DD statements. The output tape must be defined with a TAPEOUT DD statement; a separate backup tape data set is created for each TAPE control statement. A disposition of MOD cannot be used to put multiple data sets on one tape; however, multivolume data sets are permitted.

TAPE Function Statement

The output operations to be performed are described on the TAPE function statement. The specific options which are available with the TAPE function are described below.

1-2	Mnemonic	Parameters
./	TAPE	[FRMGROUP={aaabbb|SYSGRP}, FROMUSER={aaanmm|SYSLIB|**LIB},]
		[FROMUSR2=aaanpp,]FROMCLUS=k,
		PASSWORD=xxxxxxxx,
		[FILE={PROG|DATA|BOTH},]
		[NAME=filename|xxxxxxx.|(ALL),]
		[LANG={BASIC|FORTRAN|PL/I|DATA},]

Note: {} - select one from enclosed list
[] - indicates optional parameter

FRMGROUP=

aaabbb — specifies the user group which is the only user group to be contained on the backup tape. When this option is specified, one or more aaabbbnn DD statements must be supplied.

SYSGRP — specifies that the backup tape is to contain the system group only. When this option is specified one or more SYSGRPnn DD statements must be supplied.

Note: The **Directory is in the system group, but the programs and data files it references are in other groups. Directory entries are not included unless they refer to user groups for which DD statements have been provided.

FROMUSER=

aaanmm — specifies the only user number or the first of a range of user number which the backup tape is to contain. When this option is specified, one or more aaabbbnn DD statements must be supplied to define the user group to which this user number or range of numbers belongs. A *Directory is always written on the tape for the specified user or range of users; consequently, it is not necessary to start a range of users with a FROMUSER of the form aaan00. If FROMUSER is of the form aaan00 and FROMUSR2 is not specified or specifies the same user number, no backup tape is written.

SYSLIB — specifies that the backup tape is to contain the ***Library only.

**LIB — specifies that the backup tape is to contain the **Library only. Only the directory is included; the pooled programs and data files themselves are not. Directory entries are not included unless they refer to user groups for which DD statements have been provided.

Note: If both the FRMGROUP and FROMUSER parameters are omitted, the backup tape covers the system group from the cluster specified and all user groups for which the user has supplied DD statements.

FROMUSR2=aaanpp

specifies the last number of a range of user numbers which the backup tape is to contain. This parameter is not needed if only one user to be included on the tape. The FROMUSR2 parameter may be specified only when the FROMUSER parameter has been specified as aaanmm. When FROMUSR2 is specified, aaan must be the same for FROMUSER and FROMUSR2, and mm must be less than or equal to pp, so that FROMUSER and FROMUSR2 define a range of users within a subscription group.

FROMCLUS=k

specifies the cluster number, either 1 or 2, from which the backup tape is to be written.

PASSWORD=xxxxxxxx

specifies the current password of the single, currently valid user specified by FROMUSER. The password given on the control card must match the password in the user's equivalency entry or the tape is not written.

specifies the SYSLIB password under the following conditions: if the user has been cancelled; if a range of users, user group, or entire cluster has been specified; or if FROMUSER=SYSLIB or **LIB, or FRMGROUP=SYSGRP has been specified. When this option is specified, one or more SYSGRPnn DD statements must be supplied.

FILE=

PROG	specifies that the backup tape is to include only program files.
DATA	specifies that the backup tape is to include only data files.
BOTH	specifies that the backup tape is to include both program and data files for the specified user(s).

Note: FILE=BOTH is the default.

NAME=

filename	specifies that the backup tape is to include only program and/or data files that have this name.
xxxxxxx.	specifies that the backup tape is to include only those files whose name begins with the characters indicated. A maximum of seven characters in addition to the period may be specified. The period indicates only that the characters specified are a prefix and is not used as a part of the prefix.
(ALL)	specifies that all program and/or data files for the specified user(s) are to be included on the backup tape. NAME=. is equivalent to NAME=(ALL).

Note: NAME=(ALL) is the default.

LANG={BASIC|FORTRAN|PL/I|DATA}
specifies the programming language of the programs or data files being output. If the LANG parameter is not specified, programs and data files are processed without regard to programming language.

Example

In the following example, a backup tape is created of two user groups and the system group for the primary cluster.

```
//TAPE       JOB    MSGLEVEL=1,REGION=96K
//JOBLIB     DD     DSN=OSRTS.JOBLIB,DISP=SHR
//BACKUP     EXEC   PGM=DIBCADBU
//INDEX      DD     DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT   DD     SYSOUT=A
//TAPEOUT    DD     DSN=BACKUP.DEVMMB,UNIT=2400,DISP=(NEW,KEEP),
//                  VOL=SER=(DM0001,DM0002)
//SYSGRP00   DD     DSN=OSRTS.SYSGRP00,DISP=SHR
//SYSGRP01   DD     DSN=OSRTS.SYSGRP01,DISP=SHR
//DEVDEV00   DD     DSN=OSRTS.DEVDEV00,DISP=SHR
//DEVDEV01   DD     DSN=OSRTS.DEVDEV01,DISP=SHR
//MMBMMB00   DD     DSN=OSRTS.MMBMMB00,DISP=SHR
//MMBMMB01   DD     DSN=OSRTS.MMBMMB01,DISP=SHR
//MMBMMB02   DD     DSN=OSRTS.MMBMMB02,DISP=SHR
//SYSIN      DD     *
./  TAPE  FROMCLUS=1,PASSWORD=COMMCON1
/*
```

Since neither FRMGROUP nor FROMUSER is specified, the backup tape contains information from the system group for the cluster and those user groups for which DD statements are provided, in this case user groups DEVDEV and MMBMMB.

VALIDATE FUNCTION

The VALIDATE function of the data base utility gives the capability for offline validation of one user and the assignment of a password of his choice. A range of user numbers may also be validated and assigned a common password which may be changed as soon as the user signs on to the system the first time. If an already validated user number is specified, an error message is issued.

Additional DD Statements

In addition to the required JCL statements described previously, the VALIDATE function requires one or more system group DD statements and one or more user group DD statements.

VALIDATE Function Statement

The user(s) to be validated and the password to be assigned is specified on the VALIDATE function statement. The specific options which are available with the VALIDATE function are described below.

1-2	Mnemonic	Parameters
./	VALIDATE	USER=aaanmm,[USR2=aaanpp,]CLUSTER=k,
		PASSWORD=xxxxxxxx,USERPASS=yyyyyyyy

Note: [ ] indicates optional parameter.

USER=aaanmm
: specifies the only user number or the first of a range of user numbers to be validated. One or more aaabbbnn DD statements must be supplied to define the user group to which the user number or range of numbers belongs.

USR2=aaanpp
: specifies the last number of a range of user number(s) to be validated. This parameter is not needed if only one user is being validated. When USR2 is specified, aaan must be the same for USER and USR2, and mm must be less than or equal to pp.

Note: Any user number that is within the specified range and already validated is not processed, but a message is printed.

CLUSTER=k
: specifies the cluster number, either 1 or 2, of the user group to be acted upon.

PASSWORD=xxxxxxxx
: must specify the SYSLIB password; because this parameter is required, one or more SYSGRPnn DD statements must be supplied to provide password validation.

USERPASS=yyyyyyyy
specifies the password to be assigned to the user(s) validated. In the case of validation of a range of users, this password is assigned to all users validated.

Note: The USERPASS parameter is not required for validation of the single user number that represents a directory.

Example

In the following example, a range of users numbers is validated and assigned a single password.

```
//VALID        JOB     MSGLEVEL=1,REGION=96K
//JOBLIB       DD      DSN=OSRTS.JOBLIB,DISP=SHR
//CCC          EXEC    PGM=DIBCADBU
//INDEX        DD      DSM=OSRTS.INDEX,DISP=SHR
//SYSPRINT     DD      SYSOUT=A
//SYSGRP40     DD      DSN=OSRTS.SYSGRP40,DISP=SHR
//CCCDDD40     DD      DSN=OSRTS.CCCDDD40,DISP=SHR
//CCCDDD41     DD      DSN=OSRTS.CCCDDD41,DISP=SHR
//SYSIN        DD      *
./  VALIDATE  USER=CCC101,USR2=CCC199,CLUSTER=2,PASSWORD=CON1,
./            USERPASS=CCC12345
/*
```

The user numbers in the range CCC101 through CCC199 are validated for the alternate cluster. All the users are assigned a password of CCC12345 which may be changed as soon as the user signs on the system.

WRITE FUNCTION

The WRITE function of the data base utility provides the following capabilities:

- Print (formatted or dumped in hexadecimal) the equivalency entry (excluding the password), catalog record and summary, program file, data file, or file descriptor record, for a user.
- Print (formatted or dumped in hexadecimal) every record pertaining to a given user.
- Print one or all programs or data files in the ***Library.
- Print the *Directory or **Directory.
- Print the allocation record or all equivalency entries for a user group.
- Print statistics on disk space usage, saved and stored programs, and saved data files.
- Punch program or data files in card format, with source program files converted to a format compatible with OS/360 batch compilers. BASIC source programs are punched with the same content as in CALL-OS, with the line number at the beginning of each line. The right margin must be greater than or equal to the left margin plus six. FORTRAN source programs are converted to standard fixed-form FORTRAN format. The CALL-OS line number is converted to an eight-digit sequence number and punched in columns 73 through 80. Comment lines located immediately after a continued line are treated as continuation lines rather than comments. PL/I source programs are converted to the standard 60-character PL/I character set. The

CALL-OS line number is converted to an eight-digit sequence number and punched in columns 73 through 80 (if the right margin is 72 or less). DATA terminal files are punched with the same content as in CALL-OS, except that the line number is removed from the front of each line. If the right margin is 72 or less, the line number is converted to an eight-digit sequence number and punched in columns 73 through 80.

- Convert CALL-OS program or data files to OS/360 data sets; source program files are members of partitioned data sets in card-image format compatible with OS/360 batch compilers, and data files are sequential data sets in card-image format suitable for reloading (by the INSERT/REPLACE function) or reformatted to F, FB, V, or VB format (or FA, FBA, VA, or VBA format if printer control characters are desired).

The WRITE function is the only function that may be executed at the same time that the online CALL-OS system is operating against the same data sets. Thus, control information, programs, and data files may be listed, and programs and data files may be converted to OS/360 format, all while CALL-OS is executing. The online system and the WRITE function may use the same system group and/or user group data sets; these common data sets must be defined for both jobs with DD statements that contain DISP=SHR. The WRITE function executes in a task area size of 68K.

Additional DD Statements

In addition to the required JCL statements described previously, the WRITE function requires, depending on the operation to be performed, one or more system group DD statements and/or one or more user group DD statements. If a program file is to be written into an OS/360 partitioned data set, a LIBRARY DD statement that describes the output data set must be supplied. If a data file is to be written into an OS/360 sequential data set, a DD statement that describes the output data set must also be supplied. If a program or data file is to be punched, a SYSPUNCH DD statement must be supplied.

WRITE Function Statement

The output produced depends on the specifications made in the WRITE function statement. Specific options available with the WRITE function are described below.

```
┌──────┬──────────┬──────────────────────────────────────────────┐
│ 1-2  │ Mnemonic │              Parameters                      │
├──────┼──────────┼──────────────────────────────────────────────┤
│ ./   │ WRITE    │ {FRMGROUP={aaabbb|SYSGRP}                    }
│      │          │ {FROMUSER={aaanmm|SYSLIB|**LIB}              }
│      │          │                                              │
│      │          │ FROMCLUS=k,[OUTPUT={PRINT|CARD|OSDS},]       │
│      │          │                                              │
│      │          │ FILE={PROG|DATA|BOTH|DES|CAT|DIR|EQUIV|      │
│      │          │       ALLOC|EVERY|STAT},                    │
│      │          │                                              │
│      │          │ [NAME={filename|xxxxxxx.|(ALL)},]            │
│      │          │                                              │
│      │          │ [RENAME={newname|yyyyyyy.},]                 │
│      │          │                                              │
│      │          │ [LANG={BASIC|FORTRAN|PL/I|DATA},]            │
│      │          │                                              │
│      │          │ [MARG=(mm,nn),]                              │
│      │          │                                              │
│      │          │ [OPTIONS=([HEX,][OBJECT,][LIST,]             │
│      │          │           [FORMDATA]),]                      │
│      │          │                                              │
│      │          │ [PASSWORD=xxxxxxxx]                          │
└──────┴──────────┴──────────────────────────────────────────────┘
```

Note: {} - choose one from enclosed list
[] - indicates optional parameter

FRMGROUP=

aaabbb — specifies the user group from which records are to be written; when this option is specified, one or more SYSGRPnn DD statements and one or more aaabbbnn DD statements must be supplied.

SYSGRP — specifies that records are to be written from the system group data sets, which contain the ***Catalog and the **Directory; when this option is specified, one or more SYSGRPnn DD statements must be supplied.

Note: When FRMGROUP is specified, only the allocation record, equivalency file, or group statistics may be obtained.

FROMUSER=

aaanmm — specifies the user number from which records are to be written. When this option is specified, one or more aaabbbnn DD statements must be supplied to define the user group to which this user number belongs. If mm is equal to 00, the FROMUSER is the *Library for a subscription group.

SYSLIB — specifies that records are to be written from the ***Library.

**LIB — specifies that records are to be written from the **Library.

FROMCLUS=k

specifies the cluster number, either 1 or 2, from which the indicated records are to be written.

OUTPUT=

PRINT — specifies that the records indicated by the FILE and NAME parameters are to be written on the output device specified by the SYSPRINT DD statement.

CARD — specifies that the programs and/or data files indicated by the FILE and NAME parameters are to be written on the output device specified by the SYSPUNCH DD statement.

OSDS — specifies that the programs and/or data files indicated by the FILE and NAME parameters are to be written in an OS/360 data set. Source (or object) program files are written in the partitioned data set defined by the LIBRARY DD statement. They are given a member name which matches the newname of the program. Data files are written in the OS/360 physical sequential data set defined by the DD statement whose name matches the newname of the data file. Existing members in the LIBRARY partitioned data set may be replaced by the WRITE function by specifying DISP=OLD on the LIBRARY DD statement. If DISP=NEW or MOD is specified, new members are stowed into the data set, but existing members are not replaced; an OS/360 abnormal termination message is issued if an attempt is made to stow a member whose name already exists in the data set.

The required DCB parameters must be specified, either if SYSOUT is specified on a data file DD statement, or if DISP=NEW is specified on the LIBRARY DD statement or on a data file DD statement. For example,

```
//DATAF1    DD  DSN=AAA001.DATAF1,DISP=(NEW,KEEP),
//              UNIT=2314,VOL=SER=MYPACK,
//              SPACE=(TRK,10),DCB=(RECFM=VB,
//              BLKSIZE=2594,LRECL=259)
//DATAF2    DD  SYSOUT=A,DCB=(RECFM=VBA,BLKSIZE=1254,
//              LRECL=125)
//LIBRARY   DD  DSN=AAA001.SOURCE,DISP=(NEW,CATLG),
//              UNIT=2314,VOL=SER=MYPACK,
//              SPACE=(CYL,(15,,10)),DCB=(DSORG=PO,
//              RECFM=FB,BLKSIZE=3200,LRECL=80)
```

Note that when a new partitioned data set is allocated on the LIBRARY DD statement, the DCB parameter must specify DSORG=PO; in addition, the SPACE parameter must specify the number of directory blocks to be used.

New LIBRARY partitioned data sets and new data files without the FORMDATA option must have card-image records; that is, a record format (RECFM) of FB and a logical record length (LRECL) of 80, or a record format of F and a block size (BLKSIZE) of 80. Regardless of the RECFM parameter specified on the DD statement, a default of FB is used. The BLKSIZE and LRECL parameters are optional; if omitted, defaults of BLKSIZE=800 and LRECL=80 are used.

For data files with OPTIONS=FORMDATA, the RECFM parameter may be F, FA, FB, FBA, V, VA, VB, or VBA and must be specified in either the data set label (for existing data sets) or on the DD statement (for new data sets). The BLKSIZE and LRECL parameters are optional; if omitted, the following defaults are used:

RECFM	Default LRECL	Default BLKSIZE
F	80	80
FA	121	121
FB	80	800
FBA	121	1210
V	259	259
VA	125	125
VB	259	2594
VBA	125	1254

Note: If the OUTPUT parameter is omitted, OUTPUT=PRINT is assumed.

FILE=

PROG — specifies that program(s) are to be written from the library indicated by FROMUSER.

DATA — specifies that data files are to be written from the library indicated by FROMUSER.

BOTH — specifies that either programs or data files or both are to be written from the library indicated by FROMUSER.

DES — specifies that the file descriptor record(s) for the data file(s) and/or object program(s) specified by the NAME parameter are to be listed for the user indicated by the FROMUSER parameter. OUTPUT must not be CARD or OSDS.

CAT — specifies that the catalog for the user specified by the FROMUSER parameter is to be listed. When the FROMUSER is a *Library or the **Library, the catalog entries corresponding to the directory entries are printed. OUTPUT must not be CARD or OSDS.

Note: When the entire catalog record is being listed for an individual user, unless OPTIONS=HEX, a catalog summary giving usable space due to purging or replacing, space in use for programs, and space in use for data files is also printed.

DIR — specifies that the directory is to be listed for a *Library of a subscription group or for the **Library, as indicated by the FROMUSER parameter. OUTPUT must not be CARD or OSDS.

EQUIV — specifies that the equivalency entry or entries are to be listed for the user or user group specified by the FROMUSER or FRMGROUP parameter. OUTPUT must not be CARD or OSDS.

ALLOC — specifies that the allocation record for the user group specified by the FRMGROUP parameter is to be printed. OUTPUT must not be CARD or OSDS.

EVERY — specifies that every applicable type of record for the specified FROMUSER is to be listed. For an individual user, EVERY is equivalent to specifying EQUIV, CAT, DES, and BOTH. When FROMUSER is a *Library or the **Library, EVERY is equivalent to EQUIV, DIR, and BOTH. OUTPUT must not be CARD or OSDS.

STAT — specifies the listing of statistics concerning allocated, unallocated, and purged space, number and size of saved programs and data files, for the user group specified by the FRMGROUP parameter. OUTPUT must not be CARD or OSDS.

NAME=

filename — specifies the name of the file or entry to be written out.

xxxxxxx. — specifies that only those files whose name begins with the characters indicated are to be written out. A maximum of seven characters in addition to the period may be specified. The period indicates only that the characters specified are a prefix and is not used as part of the prefix.

(ALL) — specifies that all records or entries of the type named by FILE are to be written out. NAME=. is equivalent to NAME=(ALL).

Note: The NAME parameter may be specified only when FILE=PROG, DATA, BOTH, DES, CAT, DIR, or EVERY. When FILE=PROG, DATA, BOTH, DES, or EVERY, the NAME parameter must be specified. For FILE=CAT or DIR, the default is NAME=(ALL).

RENAME=

newname — specifies the name of the converted output version of the program or data file. If this parameter is omitted, the newname is assumed to be identical to the filename; in this case, the name on the DD statement that defines the output data set must also be identical to filename.

yyyyyyy. — specifies a prefix of up to seven characters which is to be added to the name of the converted output version of the program or data file. Resulting filenames which exceed eight characters are truncated on the right. If the NAME parameter also specifies a prefix, the prefix specified by RENAME replaces the prefix specified by NAME in the result; if RENAME=. is specified, the prefix specified by NAME is omitted from the result.

Notes: (1) If the NAME parameter specifies a prefix or if NAME=(ALL) is specified, RENAME may be omitted or may specify a prefix, but may not specify a unique newname. RENAME may be specified only for OUTPUT=CARD or OUTPUT=OSDS.

(2) If an embedded / would otherwise appear in the resulting newname, the / is removed from the name.

(3) If the resulting newname would otherwise begin with a numeric character (due to the deletion of leading characters when RENAME=. is specified), then a # is added to the beginning of the name.

LANG={BASIC|FORTRAN|PL/I|DATA}

specifies the programming language of the programs or data files being output. If the LANG parameter is not specified, programs and data files are processed without regard to programming language. The LANG parameter may be specified only when FILE=PROG, DATA, BOTH, DES, CAT, DIR, or EVERY.

MARG=(mm,nn)

specifies the card column margins of the converted output (program or data file) for OUTPUT=CARD or OSDS, where mm specifies the left margin and nn specifies the right margin of the card-image output. The numbers mm and nn may be one or two decimal digits. The left margin must be greater than or equal to 1 and must be less than the specified right margin. The right margin must be less than or equal to 80; if it is less than or equal to 72, a sequence number is generated in columns 73 through 80 of the output card-image. In the case of source programs, this sequence number is of the form 000ddddd, where ddddd is the source program line number. For object programs and data files, the sequence numbers start with 00000010 and are incremented by 10. When OUTPUT=OSDS, data files are not given margins or sequence numbers if the output data set does not contain card-image records.

Notes: (1) For BASIC source programs, the default is MARG=(1,80). The same line number is generated on each card punched for a BASIC statement longer than 80 characters.

(2) For FORTRAN source programs, for all object programs, and for data files for which the FORMDATA option is not specified, the standard output is MARG=(1,72) and may not be overridden by using the MARG parameter.

(3) For PL/I source programs, the default is MARG=(2,72).

(4) The same sequence number is generated for each card punched for the same FORTRAN or PL/I statement.

(5) For DATA terminal files and data files for which the FORMDATA option is specified and whose output data set is card-image, the default is MARG=(1,80).

OPTIONS=( )

Note: If only one option is given, the parentheses may be omitted.

HEX — specifies that the records are to be listed in hexadecimal dump format. This option may be specified only when OUTPUT=PRINT. If HEX is not specified, the records are formatted, and listed in character form with heading, unless the records are those of a stored object program, in which case the listing is always in hexadecimal dump format.

OBJECT — specifies that object programs are the only programs which may be output. If OBJECT is specified, the FILE parameter must specify PROG, BOTH, or EVERY. If the OBJECT option is not specified, only source programs are written.

LIST — causes the printed listing of a program or data file which has been converted for OUTPUT=CARD or OSDS. This printed listing is in output record EBCDIC character format for source programs and for data files for which the FORMDATA option has been specified; in all other cases, the listing is in output record hexadecimal format. The LIST option may not be specified when OUTPUT=PRINT.

FORMDATA — specifies that the records in any CALL-OS data file are to be stripped of certain control bytes, and, where appropriate, split into logical records in the output data set. Internal format data files are

converted to external format in the output data set. The OUTPUT parameter must specify CARD or OSDS and FILE must specify DATA or BOTH.

Note: When OUTPUT=OSDS, the record format (RECFM) DCB parameter must be specified for the output data set. The logical record length (LRECL) and the blocksize (BLKSIZE) may be specified. If omitted, defaults are assigned as described under the OSDS output option.

PASSWORD=xxxxxxxx

specifies the SYSLIB password if the FRMGROUP parameter is specified or if the FROMUSER is canceled; when this option is specified, one or more SYSGRPnn DD statement must be supplied.

specifies the FROMUSER's password. If the FROMUSER is a * Library, then the PASSWORD parameter must specify the password of some currently valid user in the subscription group. For example, if FROMUSER=AAA100, then the password of any valid user from AAA101 to AAA199 may be used. If the FROMUSER is canceled, the SYSLIB password must be specified. If FROMUSER=**LIB, the PASSWORD parameter should be omitted. Protected programs and data files can not be output when the FROMUSER is a *Library or the **Library.

Note: When FROMUSER=SYSLIB, unprotected programs and data files may be written out even if the PASSWORD parameter is omitted. Protected programs and data files and equivalency records require the SYSLIB password to be specified.

Example

In the following example, the program and data files from the ***Library for the entire system are to be printed.

```
//WRITE       JOB     MSGLEVEL=1,REGION=68K
//JOBLIB      DD      DSN=OSRTS.JOBLIB,DISP=SHR
//THREEST     EXEC    PGM=DIBCADBU
//INDEX       DD      DSN=OSRTS.INDEX,DISP=SHR
//SYSPRINT    DD      SYSOUT=A
//SYSGRP00    DD      DSN=OSRTS.SYSGRP00,DISP=SHR
//SYSGRP01    DD      DSN=OSRTS.SYSGRP01,DISP=SHR
//SYSGRP02    DD      DSN=OSRTS.SYSGRP02,DISP=SHR
//SYSGRP40    DD      DSN=OSRTS.SYSGRP40,DISP=SHR
//SYSGRP41    DD      DSN=OSRTS.SYSGRP41,DISP=SHR
//SYSIN       DD      *
./  WRITE  FROMUSER=SYSLIB,FROMCLUS=1,FILE=BOTH,NAME=(ALL)
./  WRITE  FROMUSER=SYSLIB,FROMCLUS=2,FILE=BOTH,NAME=(ALL)
/*
```

Both program files and data files are written from the primary and alternate ***Library.

WRITE Function Output

The following text describes the output of the WRITE function when OUTPUT=PRINT is specified and OPTIONS=HEX is not specified. The utility control statement is printed preceding the output associated with the statement. A header line (RECORD KEY:) appears before the file or record to be printed. The contents of this header depend on the type of file or record being printed. The rest of this section describes the output format for the types of files and records in the data base: equivalency, directory, catalog, source program, object program, and

data files; allocation and file descriptor records. In addition, user and group statistics may be obtained with the WRITE function; the format of these statistics is described in "User and Group Statistical Reports".

Note: The description of the directory and catalog file output applies to a request for the printing of the entire file. If selective printing is requested, this output is somewhat simplified.

Allocation Record: The header for an allocation record indicates the type of output (ALOC), the group and relative data set number to which the allocation information applies, and the date on which the record was last written. The header is followed by the total number of tracks in the data set, the number of unused tracks, the address of the next unused full track, and the addresses of the last used half, fifth, and tenth tracks. The allocation record for each data set in the group is printed.

Equivalency File: The header for each link of an equivalency file indicates the type of output (EQU.), the address of the next link in the file, the group to which the equivalency information applies, and the date on which this link was last written. The header is followed by a list of the equivalency entries in the link. The listing for each entry contains the following:

- The user number
- The date on which the user number was validated
- Either the password or an indication that the user number has been canceled
- The address of the first catalog link associated with the user number; for user numbers of the form aaan00, the address of the first directory link
- The amount of CPU time accumulated by the user
- The amount of time the user was connected to the system
- The date on which the user last used CALL-OS
- The number of disk tracks associated with the user

If the equivalency file has more than one link, each link is printed in the format described. The header line precedes each link. For the last (or only) link in the file, the next link field in the header contains zeros.

Directory file: The header for each link of a directory file indicates the type of output (DIR.), the address of the next link in the directory file, the user (either PUBLIC for the **Directory or a user number of the form aaan00 for a *Directory), and the date on which this link was last written. The directory entries are printed in two columns. Each entry contains the name of the file, the user number of the pooling user, the date on which the file was pooled, and the address of the catalog link which contains the file. Null entries appear when a file name has been pulled from the directory.

If a directory file has more than one link, each link is printed in the format described. The header line precedes each link. For the last (or only) link, the next link field in the header contains zeros.

Note: For the **Directory, the catalog link addresses may not be accurate. These addresses reflect the location of the links the last time CALL-OS was initialized. If a user group was not initialized, the catalog link addresses for all files pooled by users in that group will be zero.

Catalog File: The header for each link of a catalog file indicates the type of output (CAT.), the address of the next link in the file, the user whose catalog is being printed, and the date on which this link was last written. This header is followed by a list of the entries in the catalog link. The listing for each entry contains the following:

- The type of entry
- The language
- In some cases, the maximum size of the program or data file: for source programs which have just been run and then saved, an estimate of the number of 2K blocks required to run the program; for object programs, the number of 2K blocks of object code; for data files, the maximum number of units (half tracks) which can be allocated to the file
- A flags field which indicates special conditions associated with the program or data file, as follows:

  L the file is locked

  P the file is protected

  E the object program was allocated excess file units which were not used; the number of excess units appears at the right of the entry

  R the file has been released

  When present, these flags may appear singly or in combination.

- The name of the file
- The date on which the file was created
- The date on which the file was last accessed
- The number of lines in the file
- The size of the file: for source program files, the number of bytes; for object program and data files, the number of file units
- The location of the file: for source program files and for object programs and data files occupying four or less file units, from one through four disk addresses; for object program files and data files occupying more than four file units, the address of a file descriptor record, which in turn gives the location of the file

If a catalog file has more than one link, each link is printed in the format described. The header line precedes each link. For the last (or only) link, the next link field in the header contains zeros. The catalog listing is followed by a catalog summary. This summary gives the number of programs, data files, COBI jobs, and purged entries in the catalog, as well as the amount of storage used and the amount of storage available because of purging.

Source Program File: The header for a source program file indicates the type of output (PROG), the name of the file, the user number of the user who created the file, and the date on which the file was created or last modified. The header is followed by the source program. Each line in the source program is printed, preceded by the number of bytes in the line. This number appears in parentheses unless the program is continued to another file unit. In this case, the number appears between asterisks. When a source program exceeds one file unit, the header line precedes each subsequent file unit.

Object Program File: The header for an object program file indicates the type of output (OBJ.), the name of the file the user number of the user who stored the object program, and the date on which the file was last stored. The header is followed by a hexadecimal dump of the object code. If an object program file exceeds one file unit, the header line precedes each subsequent file unit.

Data File: The header for a data file indicates the type of output (DATA), the name of the file, the user number of the user who created the file, and the date on which the file was last written. A second header line indicates the number of data bytes in the file unit, the last file unit in the file (otherwise blank), and the type of data in the file. This is followed by the data itself. If a data file exceeds one file unit, the two header lines precede each subsequent file unit.

File Descriptor Record: The header for a file descriptor record indicates the type of output (FILE), the catalog entry which refers to the record, the user number, and the date on which the record was last written. The header is followed by a list of file unit addresses, in groups of eight, which give the location of file associated with the record.

USER AND GROUP STATISTICAL REPORTS

During the processing for the ACCOUNT, RECONSTRUCT, and REORGANIZE functions, statistics are automatically printed for each user processed. Statistics may also be requested independently by specifying FILE=STAT in the WRITE function statement.

User Statistics

For each user number, the following line or lines are printed:

- One directory line is printed when a directory is involved in the processing. This line identifies the directory (either aaan00 DIRECTORY for each sub group directory or **DIRECTORY for the system directory) and indicates the number of half tracks, pooled entries, and null (pulled) entries.

- Four user lines are printed for each user number involved in the processing. The first line gives the user number (either SYSLIB for the system group or aaannn for each user in each user group), the number of half tracks and number of null entries in the catalog, and either the number of days since any disk activity on the catalog or an indication that the user has been canceled. The second line indicates the number of source programs, object programs, data files, purged entries, and COBI jobs in the catalog. The last two lines give the amount of used and purged disk space, in tenth, fifth, half, and full track amounts.

Group Statistics

At the end of the processing for the function, group statistics are produced. The statistics can be of use to the computer center in planning expansion of the data base and in judging the frequency of reorganization required. The group statistics are a detailed breakdown of storage use and activity analysis on the output user group. This report is produced on three pages.

The first page of the group statistics indicates the group (either SYSGRP for the system group or aaabbb for a user group), the cluster to which the group belongs, and a summary of disk space used by the group. The disk summary lists the total tracks in the group, the number of tracks available, in use, purged, and lost. This summary is followed by a breakdown of the records in the group and the amount of space in use and purged for the entire group.

The second page provides an analysis by size and programming language for all files associated with the group. This analysis is divided into source program, object program, and data file statistics. The analysis for source programs gives the number of programs in each disk size as well as the number of lines and bytes per line. For object programs and data files, the analysis is by half tracks, with an indication of the number of programs or files in each of several ranges.

The last page provides an activity analysis. The breakdown is by number of days (several ranges are used) and is categorized by the number of source programs, object programs, data files, and COBI jobs in each range. The number of days since last used, run, read, or submitted provides an indication of the currency of group activity; the number of days since last saved, stored, or created provides an indication of the growth activity.

As described in the system build process, the material received by the installation consists of from two to four release tapes, depending on the number of compilers selected. Each release tape contains source and macro libraries in the form of unloaded data sets. These libraries, together with source modifications, are used by the installation to apply fixes to the system.

Before updates can be made, the respective source and macro libraries must be loaded from the release tape onto disk. Then the source module is updated with the source modifications and the altered source module is assembled to obtain an object deck containing all the changes. This object deck is then relink-edited and a new system load module is produced. Note that upon completion of a relink-edit of a compiler module, the utility U#UTIL1 must be run to convert the new version of the compiler to the fast-load-and-go format and to update the compiler size in the index.

The examples given below illustrate the various methods that can be used to move the data sets provided from tape to disk and to update the system. For detailed information of block sizes and data set members, the reader is referred to the appropriate CALL-OS Application Directory which is supplied with each release tape.

LOADING EXECUTIVE AND UTILITIES SOURCE AND MACRO LIBRARIES

To load the source and macro libraries for the executive and the utilities, punch and execute the following JCL, supplying the necessary information. If the macro library was not deleted after the system build process, the COPY statement for the macro library may be omitted.

```
//LOAD     JOB     ---
//         EXEC    PGM=IEHMOVE
//SYSPRINT DD      SYSOUT=A
//SYSUT1   DD      UNIT=2314,DISP=OLD,VOL=SER=scrvol
//DD1      DD      UNIT=2314,DISP=OLD,VOL=SER=volid1
//TAPE     DD      UNIT=2400,DISP=OLD,VOL=SER=RTOSYS,
//                 LABEL=(,NL),DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSIN    DD      *
   COPY    PDS=OSRTS.EXEC.MACLIB,TO=2314=volid1,                X
               FROM=2400=(RTOSYS,3),FROMDD=TAPE,                X
               RENAME=qualifier.MACLIB
   COPY    PDS=OSRTS.EXEC.SOURCE,TO=2314=volid1,                X
               FROM=2400=(RTOSYS,4),FROMDD=TAPE
/*
```

Parameter	Meaning
scrvol	Volume identification of scratch volume
volid1	Volume identification of volume on which qualifier.MACLIB resides
qualifier	Index level qualifier for CALL-OS chosen by user at system build time

LOADING COMPILER SOURCE AND MACRO LIBRARIES

The following example loads the source and macro libraries for all three compilers. The statements for those compilers not in the system should be omitted.

```
//LOAD          JOB     ---
//              EXEC    PGM=IEHMOVE
//SYSPRINT      DD      SYSOUT=A
//SYSUT1        DD      UNIT=2314,DISP=OLD,VOL=SER=scrvol
//DD1           DD      UNIT=2314,DISP=OLD,VOL=SER=volid1
//BASTAPE       DD      UNIT=2400,DISP=OLD,VOL=SER=BASIC,LABEL=(,NL),
//                      DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//FORTAPE       DD      UNIT=2400,DISP=OLD,VOL=SER=FORT,LABEL=(,NL),
//                      DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//PLITAPE       DD      UNIT=2400,DISP=OLD,VOL=SER=PLI,LABEL=(,NL),
//                      DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//SYSIN         DD      *
   COPY         PDS=OSRTS.BASIC.MACLIB,TO=2314=volid1,          X
                    FROM=2400=(BASIC,2),FROMDD=BASTAPE,         X
                    RENAME=qualifier.MACLIB
   COPY         PDS=OSRTS.BASIC.SOURCE,TO=2314=volid1,          X
                    FROM=2400=(BASIC,3),FROMDD=BASTAPE
   COPY         PDS=OSRTS.FORTRAN.MACLIB,TO=2314=volid1,        X
                    FROM=2400=(FORT,2),FROMDD=FORTAPE,          X
                    RENAME=qualifier.MACLIB
   COPY         PDS=OSRTS.FORTRAN.SOURCE,TO=2314=volid1,        X
                    FROM=2400=(FORT,3),FROMDD=FORTAPE
   COPY         PDS=OSRTS.PLI.MACLIB,TO=2314=volid1,            X
                    FROM=2400=(PLI,2),FROMDD=PLITAPE,           X
                    RENAME=qualifier.MACLIB
   COPY         PDS=OSRTS.PLI.SOURCE,TO=2314=volid1,            X
                    FROM=2400=(PLI,3),FROMDD=PLITAPE
/*
```

Parameter	Meaning
scrvol	Volume identification of scratch volume
volid1	Volume identification of volume on which qualifier.MACLIB resides
qualifier	Index level qualifier for CALL-OS chosen by user at system build time

Note that the executive and utilities macro library is required in order to assemble compiler modules. The compiler macro libraries and the executive and utilities macro library should be combined into one data set as indicated by the RENAME parameter in the examples given. If they are not so combined, the update and assemble example must be modified to concatenate the libraries on the SYSLIB DD statement for the assembler step.

OBTAINING A MODIFIED OBJECT DECK

The source module is updated with the changes and reassembled to produce an object deck which contains the changes. The following example updates a source module with changes and assembles the modified module.

```
//UPDTASM  JOB      ---
//UPDT     EXEC     PGM=IEBUPDTE,PARM=MOD
//SYSUT1   DD       DSN=OSRTS.component.SOURCE,DISP=OLD,
//                  UNIT=2314,VOL=SER=volid1
//SYSUT2   DD       DSN=OSRTS.component.SOURCE,DISP=OLD,
//                  UNIT=2314,VOL=SER=volid2
//SYSPRINT DD       SYSOUT=A
//SYSIN    DD       *
./ CHANGE           NAME=modname

   (source modification cards)

/*
//ASM      EXEC     PGM=IEUASM,PARM='LINECOUNT=50,DECK',
//                  REGION=80K,COND=((0,NE,UPDT))
//SYSLIB   DD       DSN=qualifier.MACLIB,DISP=OLD,UNIT=2314,
//                  VOL=SER=volid
//         DD       DSN=SYS1.MACLIB,DISP=OLD
//SYSUT1   DD       UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSUT2   DD       UNIT=SYSDA,SPACE=(1700,(400,50))
//SYSUT3   DD       UNIT=(SYSDA,SEP=(SYSUT2,SYSUT1,SYSLIB)),
//                  SPACE=(1700,(400,50))
//SYSPRINT DD       SYSOUT=A
//SYSPUNCH DD       SYSOUT=B
//SYSIN    DD       DSN=OSRTS.component.SOURCE(modname),
//                  DISP=OLD,UNIT=2314,VOL=SER=volid2
```

Parameter	Meaning
component	Name of the CALL-OS component to be updated; it must be one of the following: EXEC, BASIC, FORTRAN, or PLI
volid1	Volume identification of volume on which OSRTS.component.SOURCE resides
volid2	Volume identification of volume on which OSRTS.component.SOURCE is to reside
modname	Name of the module (or subroutine) to be updated
qualifier	Index level qualifier for CALL-OS chosen by user at system build time
volid	Volume identification for volume on which qualifier.MACLIB resides

OBTAINING A MODIFIED SYSTEM LOAD MODULE

A system load module is link-edited using an object deck and linkage editor control statements. The object deck is obtained as in the preceding example. The control statements used depend on the module being updated. The following text shows an example of the JCL required to execute the linkage editor, followed by the linkage editor control statements.

LINKAGE EDITOR JCL REQUIREMENTS

The following example shows the JCL required to execute the linkage editor:

```
//LKED     JOB      ---
//LINK     EXEC     PGM=IEWL,PARM='XREF,LIST,LET,NCAL',REGION=96K
//SYSLIN   DD       DDNAME=SYSIN
//SYSPRINT DD       SYSOUT=A
//SYSLMOD  DD       DSN=qualifier.JOBLIB,DISP=OLD
//SYSUT1   DD       UNIT=(SYSDA,SEP=(SYSLMOD,SYSLIN)),
//                  SPACE=(1024,(200,20))
//SYSIN    DD       *

   (object deck and linkage editor control statements)

/*
```

LINKAGE-EDITOR CONTROL STATEMENTS

The following sections show the linkage editor control statements required to link edit system modules. The statements required depend on the module.

Control Statements for RTOS1

```
INSERT  T#GTAB
INSERT  C#CPID      (C#CPIDV for MVT user)

(object deck obtained from update and assembly)

INCLUDE SYSLMOD(RTOS1)
ENTRY   N#LINIT
NAME    RTOS1(R)
```

Control Statements for U#UTIL1

```
(object deck obtained from update and assembly)

INCLUDE SYSLMOD(U#UTIL1)
ENTRY   U#UTIL1
NAME    U#UTIL1(R)
```

Control Statements for DIBCADBU

```
(object deck obtained from update and assembly)

INCLUDE SYSLMOD(DIBCADBU)
ENTRY   DIBINIT
NAME    DIBCADBU(R)
```

Control Statements for Other Modules

The following control statements are to be used for potentially nonresident modules and utility modules other than those referred to previously:

```
(object deck obtained from update and assembly)

NAMEmodname(R)
```

Control Statements for the BASIC Compiler

Note that the module NUCLEUS must be the first module in the composite load module (BASIC).

1. For NUCLEUS

   (object deck obtained from update and assembly)

```
INCLUDE SYSLMOD(BASIC)
NAME    BASIC(R)
```

2. All other BASIC modules

```
REPLACE csect[,csect,...]   (Replace all csects in assembled
                             module)
INCLUDE SYSLMOD(BASIC)
```

   (object deck obtained from update and assembly)

```
NAME    BASIC(R)
```

Control Statements for the FORTRAN Compiler

Note that the module FORCOMP must be the first module, followed by the eight modules listed below in item 2, in the composite load module (FORTRAN).

1. For FORCOMP

   (object deck obtained from update and assembly)

```
INCLUDE SYSLMOD(FORTRAN)
NAME    FORTRAN(R)
```

2. Any of the following: FORALC1,FORALC2,FORINIT,FORPCHK, FORPOPS,FORREM,FORSCN,FORSCNSR.

   (object deck of FORCOMP)
   (object deck obtained from update and assembly)

```
INCLUDE SYSLMOD(FORTRAN)
NAME    FORTRAN(R)
```

3. All other FORTRAN modules

```
REPLACE csect[,csect,...]   (Replace all csects in assembled
                             module)
INCLUDE SYSLMOD(FORTRAN)
```

   (object deck obtained from update and assembly)

```
NAME    FORTRAN(R)
```

Control Statements for the PLI Compiler Phase

Note that the module $CCONT must be the first module in the composite load module (PLI).

1. For $CCONT

   (object deck obtained from update and assembly)

```
INCLUDE SYSLMOD(PLI)
NAME    PLI(R)
```

2. All other PLI phase modules

```
REPLACE csect[,csect,...]   (Replace all csects in assembled
                             module)
INCLUDE SYSLMOD(PLI)

(object deck obtained from update and assembly)

NAME    PLI(R)
```

Control Statements for the PL2 Compiler Phase

Note that the module $WCONT must be the first module in the composite load module (PL2).

1. For $WCONT

```
(object deck obtained from update and assembly)

INCLUDE SYSLMOD(PL2)
NAME    PL2(R)
```

2. All other PL2 phase modules

```
REPLACE csect[,csect,...]   (Replace all csects in assembled
                             module)
INCLUDE SYSLMOD(PL2)

(object deck obtained from update and assembly)

NAME    PL2(R)
```

DIAGNOSTIC AIDS

CALL-OS provides several debugging facilities which can be used in conjunction with, and in addition to, the service aids provided by OS/360. The global table and the user terminal table are the primary sources of information concerning the status of the CALL-OS program at the time of an error. For information regarding the events which immediately preceded the error, the CALL-OS entries in the OS/360 trace table are of great assistance. For online debugging and error analysis, the commands *STATUS and *REPORT can provide useful information.

GLOBAL TABLE AND USER TERMINAL TABLE

Efficient debugging requires knowledge of system control information contained in the form of control blocks and tables. The CALL-OS global table and the user terminal table provide important sources of information concerning status of both the system and the individual user.

The global table can be located by reference to the link-edit map generated at system build time. Its symbolic name is T#GTAB. When CALL-OS is in control, register 12 contains the address of the global table. Constants, important addresses, and canned messages (referenced by more than one module or subroutine) are placed in this table.

The user terminal table, assigned to each communications line, contains information concerning each individual user. Channel program areas and OS/360 input/output blocks are located within the UTT, together with information needed by the compilers and the executive routines. The upper and lower limits of the list of UTT's can be found in the global table.

CALL-OS TRACE ENTRIES

If the CALL-OS user includes the trace table option in his OS/360 system generation, he also has the option of placing special CALL-OS entries in that same table. These entries are made automatically unless NOTRACE has been included in the PARM field on the EXEC statement of the startup deck. These trace entries are very important in any error analysis within CALL-OS, and the user should create a trace table sufficiently large to ensure its usefulness.

*REPORT COMMAND

This operator command prints the contents of various statistical counters, including I/O activity counts, and hardware-oriented terminal information. These reports provide valuable data to help diagnose communications line problems and other errors that do not normally cause a core dump to be printed. If a dedicated printer is not specified on the SYSPRINT DD statement at system initialization, the report is printed at job end. An example of a statistical report and the definition of fields for the *REPORT command are presented in Appendix A.

*STATUS COMMAND

This operator command obtains current status information from the designated UTT and prints it on the command console. This command is a useful debugging aid where a quick status check is needed for a user experiencing difficulties with a particular system process. The definition of codes for the *STATUS command is given in Appendix B.

APPENDIX A: EXAMPLE OF STATISTICAL REPORT (*REPORT)

```
                                CALL-OS STATISTICAL REPORT                              15:11

      TIME ON 15:10       ELAPSED TIME=    67 SEC.                                      PAGE    1

                  TOTAL REQUESTS QUEUED REQUESTS MAXIMUM QUEUE  CURRENT QUEUE     DISK READ   OLD REQUEST

60 BYTE BUFFER              0              0             0             0
256 BYTE BUFFER             3              0             0             0
SYSTEM BUFFER              10              0             0             0
SORT BUFFER                 0              0             0             0
OVERLAY BUFFER            123              0             0             0            58           58
CPID LEVEL 1              346             29             1             0
CPID LEVEL 2              166             29             1             0
CPID LEVEL 3                0              0             0             0

COMMAND TOTALS

RUN      =     2  LIST     =     1  LOAD     =     4  SAVE     =     3  *BATCH   =     0  *CANCEL  =     0  *COBI    =     0
*DATE    =     0  *DISABLE =     0  *ENABLE   =     0  *IGNORE  =     0  *MESSAGE =     0  *OFF     =     0  *REPORT  =     2
*STATUS  =     0  *TELL     =     0  *USERS    =     0  *VALIDATE=    0  *WARN    =     0  ADD      =     0  ALLOW    =     0
CANCEL   =     0  CATALOG   =     0  CLEAR    =     0  DELETE   =     0  DSSTATUS =     0  ECHO     =     0  ENTER    =     0
EXTRACT  =     0  FILE      =     0  FIND     =     0  HELP     =     0  INSERT   =     0  JOBSTATUS=    0  KEY      =     0
LOCK     =     0  LOGON     =     0  MERGE    =     1  MOVE     =     0  NAME     =     0  NOTIFY   =     0  OFF      =     0
PASSWORD =     0  POOL      =     0  PROTECT  =     0  PULL     =     0  PUNCH    =     0  PURGE    =     0  RELEASE  =     0
RENUMBER =     1  REPLACE   =     0  SCAN     =     0  SCRATCH  =     0  SECURE   =     0  STATUS   =     0  STORE    =     0
SUBMIT   =     0  TAPE      =     0  TIME     =     0  UNLOCK   =     0  WEAVE    =     0  WIDTH    =     0

TOTAL    =    14

TOTAL     RUN      TOTAL     TTY      INPUT    INPUT      OUTPUT    OUTPUT   256 BYTE   TOTAL      TOTAL     CEN. CYL.
BREAKS    BREAKS   ERASES    ERASES   LINES    REQUESTS   LINES     MESSAGES MESSAGES   MESSAGES   DISK I/OS DISK I/OS
    4         3        0         0       15         0         19        9        1          20        115        73

M#DISP    NO SWAP   NEW JOB   OLD JOB  JOBS NEED MAX. NEW  MAX. OLD  NEW JOB  NEW JOB   OLD JOB   TENRFL OLD JOB AREA
CALLED    COUNTER   QUEUE     QUEUE    COMPILER  JOB QUEUE JOB QUEUE AREA FULL HOLE SMALL AREA FULL  SET    EXTENDED
    5         0         2         0        2         1         0        0        0          0         0          0

BASIC     PL/I      FORTRAN EXCES I/O JOBS USING DATA FILE NUMBER OF  RESTARTED  APPENDS   ADD. CORE COMPILE  COMP. & GO
LOAD      LOAD      LOAD    JOB SWAP  FILES      I/OS      OPENS     EDITS                 REQUESTS  (SEC)    (SEC)
    1         0         0         0        0         0         0        0        0          0         0          0

MASTER MODE SORT    NOSORT    OBJECT    CORI     SUBMITTED SUBMITTED MAX SUB   M#JCL     # DSN     DSN       VAR TAB
 REQUESTS                     RUNS      DISK I/OS JOBS      LINES     QUEUE     RESETS    OPENS     OPENS     TRACKS
    8         0         6         0        22        0         0         0        0          0         0          0

 SCAN DS   SCAN JCL SCRATCHED MAX IOREQ   TEMP1     TEMP2     TEMP3     TEMP4     TEMP5
OUT LINES OUT LINES DATA SETS  QUEUE
    0         0         0         0        0         0         0         0        0
```

Figure 34. Statistical report example (part 1 of 3)

CPU TIME (SEC)	NO. OF COMPUTE INTERACTION
0.0 - 0.2	2
0.2 - 0.4	0
0.4 - 0.6	0
0.6 - 0.8	0
0.8 - 1.0	0
1.0 - 1.2	0
1.2 - 1.4	0
1.4 - 1.6	0
1.6 - 1.8	0
1.8 - 2.0	0
2.0 - 2.2	0
2.2 - 2.4	0
2.4 - 2.6	0
2.6 - 2.8	0
2.8 - 3.0	0
3.0 - 3.2	0
3.2 - 3.4	0
3.4 - 3.6	0
3.6 - 3.8	0
3.8 - 4.0	0
4.0 - 4.2	0
4.2 - 4.4	0
4.4 - 4.6	0
4.6 - 4.8	0
4.8 - 5.0	0
OVER	0
TOTAL	2

Figure 34. Statistical report example (part 2 of 3)

```
                                CALL-OS STATISTICAL REPORT                      15:11

TIME ON 15:10       ELAPSED TIME=    67 SEC.                                    PAGE    3

DISK USAGES

USER GROUPS          TOTAL          AVAILABLE
                     TRACKS         TRACKS

SYSLIB                100              86
BUT TO BUT            100              84


LINE #      OFF        CALLS        2741        USER
                       RECEIVED     PARITY      ON

   1          0           1          0          1

                                                 1  TOTAL
```

Figure 34. Statistical report example (part 3 of 3)

DEFINITION OF FIELDS (STATISTICAL REPORT)

FIELD NAME	DEFINITION
60 BYTE BUFFER	Summary of the use of the 60-byte buffers
256 BYTE BUFFER	Summary of the use of the 256-byte buffers
SYSTEM BUFFER	Summary of the use of the system buffer
SORT BUFFER	Summary of the use of the sort buffer
OVERLAY BUFFER	Summary of the use of the overlay buffer
CPID LEVEL 1	Summary of the level 1 interrupts
CPID LEVEL 2	Summary of the level 2 interrupts
CPID LEVEL 3	Summary of the level 3 interrupts
COMMAND TOTALS	This section indicates the number of times a specific command has been made. A field at the end of the command totals indicates the total number of all commands made.
TOTAL BREAKS	Total number of times the BREAK key on the TTY or the ATTN key on the 2741 has been depressed
RUN BREAKS	Total number of times the BREAK key or the ATTN key has been depressed while under control of the RUN command
TOTAL ERASES	Total number of erases requested
TTY ERASES	Total number of TTY erases requested (number of characters (erase messages) received)
INPUT LINES	Total number of input lines received
INPUT REQUESTS	Total number of input requests received
OUTPUT LINES	Total number of output lines transmitted
OUTPUT MESSAGES	Total number of output messages translated
256 BYTE MESSAGES	Total number of 256-byte buffer type messages
TOTAL MESSAGES	Total number of all messages transmitted
TOTAL DISK I/OS	Total number of disk I/O operations
CEN. CYL. DISK I/OS	Total number of central cylinder disk I/O operations
M#DISP CALLED	Total number of times that the routine M#DISP was entered
NO SWAP COUNTER	Number of times any job was dispatched with L#NSOB set
NEW JOB QUEUE	Number of current new job queue entries
OLD JOB QUEUE	Number of current old job queue entries

FIELD NAME	DEFINITION
JOBS NEED COMPILER	Total number of jobs requiring a compiler
MAX. NEW JOB QUEUE	Maximum number of new jobs queued at any time on this run
MAX. OLD JOB QUEUE	Maximum number of old jobs queued at any time on this run
NEW JOB AREA FULL	Number of times a new job was ready to use the new job area but the new job area was full, and the waiting new job must be queued
NEW JOB HOLE SMALL	Number of times a new job area had less than two jobs, but the hole was too small to start the second job
OLD JOB AREA FULL	Number of times an old job is ready to use the old job area, but the old job area is already being used, and the waiting old job must be queued
TENRFL SET	A count of the number of times that the second new job area is restrained from accepting a new job, in order to permit the top job in the new job queue to have sufficient space in the program area; the purpose of this field is to permit an installation to monitor the number of requests of this type, so that it may know that performance is suffering because sufficient space for user programs was not allocated
OLD JOB AREA EXTENDED	Total number of times the old job area had to be extended
BASIC LOAD	Total number of times the BASIC compiler was loaded
PL/I LOAD	Total number of times the PL/I compiler was loaded
FORTRAN LOAD	Total number of times the FORTRAN compiler was loaded
EXCES I/O JOB SWAP	Count of the swaps due to excessive I/O operations
JOBS USING FILES	Number of current jobs using data files
DATA FILE I/OS	Number of I/O operations due to the processing of data files
NUMBER OF OPENS	Total number of opens requested during execution of user program
RESTARTED EDITS	Number of editing commands which had to be restarted because the initial core allocation was too low
APPENDS	Total number of appends
ADD. CORE REQUESTS	Total number of times additional core has been requested for restart
COMPILE (SEC)	Total time in seconds spent in compilations
COMPILE AND GO (SEC)	Total time in seconds spent in compilations and execution

FIELD NAME	DEFINITION
MASTER MODE REQUESTS	A count of the number of times when a system process, such as SORT, requires program area sorting. This information is useful in determining an installation's sort buffer requirements.
SORT	Incremented by M#RDSO if sort is needed (M#RDSO reads in user requested source program)
NOSORT	Incremented by M#RDSO if sort is not needed
OBJECT RUNS	Total number of stored object programs executed
TEMP1 - TEMP5	Used during development of additional features
COBI DISK I/OS	Number of disk I/O requests issued by COBI module M#CBIO
SUBMITTED JOBS	Number of jobs written on the COBI input data sets
SUBMITTED LINES	Number of submitted lines (80 characters each)
MAX. SUB QUEUE	Maximum number of jobs in queue between M#ISUB and M#SUB
M#JCL RESETS	Number of jobs which M#JCL reset from the COBI output class to the OS/360 output class
# DSN OPENS	Number of scannable data sets generated by COBI jobs
DSN OPENS	Number of scannable data sets which were actually scanned
VAR TAB TRACKS	Number of tracks used in building the variable tables required to access variable-length records
SCAN DS OUT LINES	Number of output lines generated for scan requests for scannable SYSOUT data sets
SCAN JCL OUT LINES	Number of output lines generated for scan requests for JCL data sets
SCRATCHED DATA SETS	Number of scannable data sets scratched
MAX IOREQ QUEUE	Maximum number of queued I/O requests
COMPUTE INTER-ACTION	This table indicates the number of jobs which have executed for the specified amount of time before being swapped out as a result of time slicing, I/O request, or request of terminal input. It is normal to have a high count in the field which is compatible with the old and new time-slice values.
DISK USAGE	Specifies disk used by present run, total number of tracks, and total number of available tracks
LINE NUMBER	Specifies all the lines included in the JCL deck. It will indicate: (1) the number of times the line has received the OFF command, (2) the number of calls received on the line, (3) the number of 2741 parity errors which have occurred, and (4) the line now active.

APPENDIX B: DEFINITION OF CODES FOR *STATUS COMMAND

USER'S STATUS CODES (STAT)

Code	Associated Command	Code	Associated Command
01	RUN without parameters	1E	WIDTH
02	RUN with parameters	1F	CATALOG
03	LIST	20	ALLOW
04	LOAD	21	PROTECT
05	SAVE	22	POOL
06	RENUMBER, DELETE, EXTRACT, ADD REPLACE, MOVE, FIND, INSERT	23	PULL
		24	*STATUS
07	MERGE, WEAVE	25	*USERS
08	TIME	26	*DATE
09	*CANCEL	27	Not used
0A	*VALIDATE	28	STATUS
0B	CLEAR	29	*REPORT
0C	*TELL	2A	KEY
0D	LOCK	2B	TAPE
0E	LOGON	2C	*BATCH
0F	OFF	2D	Not used
10	FILE	2E	STORE
11	PASSWORD	2F	CANCEL
12	PURGE	30	DSSTATUS
13	UNLOCK	31	JOBSTATUS
14	*DISABLE	32	Not used
15	*ENABLE	33	NOTIFY
16	*MESSAGE	34	SCAN
17	*WARN	35	SCRATCH
18	ECHO	36	SUBMIT
19	ENTER	37	*COBI
1A	HELP	38	PUNCH
1B	*OFF	39	SECURE
1C	NAME	3A	RELEASE
1D	*IGNORE		

TERMINAL CHANNEL STATUS CODES (TCHST)

Code	Meaning
00	Idle (no command outstanding)
04	Writing text
08	Writing marks
0C	Reading text
10	Reading skip (ignore PCI)
14	Disable outstanding
18	Enable outstanding
1C	Prepare issued for break test
20	Disable outstanding (trouble) - message next
24	Issue disable next (normal)
28	Ignore interrupts (line out of service)
2C	Paper tape is halting
80	Recursive entry state to appendage--error recorded

TERMINAL FLAG BYTE CODES (TFLG1)

Code	Meaning
80	Error has occurred in the enable sequence
40	Do not enable
20	Information message queued for this line
10	Line break has been received
08	Message routine is performing line folding
04	Program lines lost when inputting from paper tape
02	Warning message ready to print
01	Flag to get special sort error message

OS/360 IOS TERMINAL COMMUNICATIONS SWITCH CODES (IOSW)

Code	Meaning
00	Exit from start I/O appendage
04	Set up to issue halt I/O for break test
08	Switch for start I/O to post appendages for busy test
08	Last legal I/O vector. Must equal last legal value.

APPENDIX C: NONRESIDENT MODULE NUMBERS

Hexadecimal Number	Module Name
070	M#ACCT
080	M#STOR
090	M#CAT
0A0	M#CCDA
0B0	M#CCDI
0C0	M#CCME
0D0	M#CCOF
0E0	M#CCRE
0F0	M#CCST
100	M#CCTE
110	M#CCUS
120	M#CCVA
130	M#CCWA
170	M#DIR
1B0	M#ECHO
1C0	M#EDIT
200	M#HELP
230	M#LDRD
240	M#LIB
260	M#LIST
270	M#LOAD
280	M#LOG
2A0	M#MWSC
2B0	M#NAME
2F0	M#PASS
300	M#ESCN
310	M#RDSO
330	M#RUN
340	M#SAVE
350	M#SORT
380	M#STAT
3A0	M#TIME
3C0	M#WID
3D0	M#WRSO
550	M#CCBA
570	M#OBJR
590	M#CANCL
5B0	M#CBST
5C0	M#CCCO
5D0	M#IJCL
5E0	M#ISCAN
5F0	M#ISUB
600	M#JCL
610	M#NOTFY
620	M#SCAN
630	M#SCR
640	M#SUB
690	M#MREM
6A0	M#WEAV
6B0	M#ABSUB

GH20-0786-3

CALL—OS V2 Executive & Utilities PDM Printed in U.S.A. GH20-0786-3

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

READER'S COMMENT FORM

CALL—OS V 2 Executive & Utilities
Program Description Manual

GH20-0786-3

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

COMMENTS

fold fold

fold fold

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A. FOLD ON TWO LINES, STAPLE AND MAIL.

GH20-0786-3

YOUR COMMENTS PLEASE...

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
1133 Westchester Avenue
White Plains, N.Y. 10604

Attention: Technical Publications

fold fold

CALL—OS V 2 Executive & Utilities PDM Printed in U.S.A. GH20-0786-3

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

IBM

Technical Newsletter

Base Publ. No. GH20-0786-3

This Newsletter No. GN20-2780

Date July 31, 1972

Previous Newsletter Nos. None

CALL-OS
Executive and Utilities
Program Description Manual
Program Number 360A-CX-42X

This Technical Newsletter is intended only for those who wish to make the base publication apply to Version 1, Modification Level 2. It provides replacement pages for the subject manual. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are listed below:

Preface – First page of Contents
1–2
13–16
19–20
25–30

A vertical rule in the left margin indicates a change. Absence of a vertical rule on a page bearing a "revised" notice means only that existing copy has been moved or that a minor typographical error has been corrected.

Please file this cover letter at the back of the manual to provide a record of changes.

Note: The IBM Operating System is commonly referred to as OS. For consistency, the term OS/360 has been replaced by the term OS on all changed pages of this manual. Users of the manual should regard OS and OS/360 as synonymous.

IBM Corporation, Technical Publications Dept., 1133 Westchester Avenue, White Plains, N. Y. 10604

Printed in U.S.A.