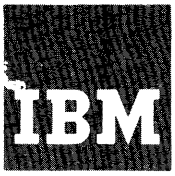
File No. S360-20
Form C28-2003-3



Systems Reference Library

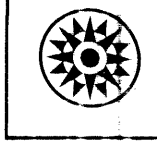
IBM System/360 Time Sharing System

Concepts and Facilities

Time Sharing System/360 is a comprehensive programming system used in conjunction with IBM System/360 computers that have time-sharing features. TSS/360 comprises a supervisory program, a group of service programs, and a group of user programs. The supervisory program controls operation of the system and provides the time-sharing environment. The service programs perform task- and data-management functions in response to user or system requests. The user programs perform language processing, linkage editing, and other work defined by the user's problem programs.

The primary purpose of TSS/360 is to provide many users with simultaneous conversational (on-line) access to a computing system that may have a single processor, or multiple processors. The combination of machine and program features gives each user the impression that he has sole possession of the system. He uses the system as if it had a directly accessible main-storage addressing space equal to the addressing capability of the system, rather than its actual main-storage capacity.

While the system is operating conversationally, for many simultaneous users, it can also operate nonconversationally, with batch-type processing jobs, in the background.



Preface

This publication is an introduction to Time Sharing System/360, and is directed to managers, administrators, operators, system monitors, and all users, including programmers.

It contains six sections:

- General description — explains TSS/360, its advantages and how it is used by system personnel.
- System summary — describes the principal components of TSS/360.
- Publications plan — gives the purpose and scope of each TSS/360 publication, and outlines the intended reading series for each type of user.
- Use of TSS/360 — illustrates how the various categories of personnel use the system, and briefly describes individuals' responsibilities.
- Task management — describes communication with the system in both conversational and nonconversational modes of operation.
- Data management — describes definition, organization, and manipulation of data in the system.

FOURTH EDITION (September 1968)

This edition, Form C28-2003-3, is a major revision of the preceding Form C28-2003-2, and obsoletes it and all previous editions, as well as Technical Newsletters N28-3005, N28-3016, and N28-3022. This edition reflects major changes in the command system, including many new commands and new names for old commands. It explains the facilities that permit the user to create new commands, rename commands, set up and maintain a unique user profile, and to use the text editor.

This edition is current with Version 3, Modification 0, and remains in effect for all subsequent versions or modifications of IBM Sytem/360 Time Sharing System unless otherwise indicated. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters. Before using this publication in connection with the operation of IBM systems, refer to the latest edition of *IBM System/360 Time Sharing System: Addendum*, Form C28-2043, for the editions of publications that are applicable and current.

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

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Department 561, 2651 Strang Boulevard, Yorktown Heights, New York 10598.

Contents

Tables and Figures

Time Sharing System/360 is a comprehensive programming system that is used in conjunction with System/360 computers having time-sharing features. The program-machine configuration is a computing system that can be employed simultaneously for either one or the combination of two processing modes: conversational, on-line processing, during which users interact with the system and their executing programs; and nonconversational or background processing.

Time and Resources Sharing

Time Sharing System/360 provides many simultaneous users with a wide variety of program services for solving individual computer problems. TSS/360 users obtain these services through commands given to the system through remote terminals. The terminals permit them to enter and construct programs and data sets, to debug programs, and to request execution of programs that use specified data sets. Furthermore, users can specify that programs and data sets are to be added to, deleted from, modified, copied, or moved to and from input/output units.

Users are in constant communication with the system, and are thus aware of the individual functions the system performs at any given instant. As requests for various services are made, the system informs individual users of the action it has taken, of additional information required to complete services requested, and of any discrepancies that may exist in service requests.

Of equal importance, all TSS/360 users have easy and open access to the central data-processing facility. Individual remote terminals provide users with their own direct means of monitoring and controlling the computers that are processing their programs. TSS/360 thus provides an easy man-machine communication that facilitates program writing and checkout, as well as the solution to complex program design problems that require immediate human judgement.

Similarly, users can initiate programs from their remote terminals, thus availing themselves of the system's nonconversational processing capabilities in a multiprogrammed, batch-processing environment.

Conversational Processing

The easy-to-operate remote terminals are typewriter-like devices; their keyboards serve as input units to the system, their printing mechanisms as the system's output units. The terminals can be located near the central computer installations, or at remote locations that extend computer access over thousands of miles.

Two types of terminals are available with TSS/360: the IBM 1050 Data Communication System, with an IBM 1052 Printer-Keyboard and an optional IBM 1056 Card Reader; and the IBM 2741 Communication Terminal.

The IBM 1050 data communication system is a multipurpose terminal that can transmit and receive information to and from a central processing unit and, also, can be used to produce documents and cards, in the off-line mode. The IBM 1050 consists of a control unit and a keyboard printer.

The 1052 printer-keyboard accepts data from the communications line, from the keyboard, or from card readers. The keyboard can send data to the communications line, to the printer, or to card punches. The 1052 also has the control switches that attach components either to the communications line or for off-line use. The optional IBM 1056 card reader sends data to the communications line. A special feature for automatic card-reread upon detection of an error is available.

The IBM 2741 communications terminal incorporates the features of the IBM SELECTRIC® typewriter in a free-standing communications terminal that is especially designed for conversational-mode processing with System/360.

Through both terminals, 1050 and 2741, system services can be made available to users at the required locations and at desired times. Furthermore, each user can employ the type of terminal that best meets his needs.

TSS/360 provides each conversational user with a certain amount of program-execution time per computer time cycle. Due to the high operating speed of

System/360, each user may consider himself as having sole possession of the system.

Minimal training is required to use the system; a user needs only to know

- the procedure for terminal operation,
- a simple control language to identify himself and his work to the system, and
- a problem-oriented language to define the desired system action, in terms familiar to the user.

The TSS/360 conversational capabilities permit the user to have dialogues with the system. Initially, the system can be considered as a teacher; it assists the novice user in developing operational skills. During a dialogue, or terminal session, the system guides the user through procedures and work, with messages that indicate errors and diagnoses. So, users can interact directly with their own programs, with other users' programs that they have been given permission to share, or with system programs during execution. They can stop programs, obtain intermediate results, introduce new data, change sequence of execution, and perform similar functions.

During any given time, TSS/360 can serve many simultaneous users with such widely varying backgrounds, occupations, and processing objectives as scientific users, commercial users, programmer users, publication personnel, programming support personnel, and others.

Nonconversational Processing

In the nonconversational mode, TSS/360 processes all programs as in the conventional batch-processing operation with a multiprogramming environment. Nonconversational processing may be performed simultaneously with conversational processing, depending upon the number of active terminal users and the total processing capacity of the system. Nonconversational tasks are normally executed with a lower priority, to avoid interference with conversational operations.

Time Sharing System/360 meets the long-felt user need for easier and direct use of computers, and the need for immediate response from the computer, thus reducing the time interval between problem definition and solution. With TSS/360, persons who do not have extensive programming background can now use a powerful computing system.

User-Oriented Design

TSS/360 efficiently serves the widely divergent requirements of many experienced users, and offers significant advantages to new users. The system accommodates professionals of various disciplines, without requiring professional programming experience; it accommodates the occasional user, the user requiring immediate results, the user who needs to interact dynamically with an executing program, and the user with production-type data-processing tasks. TSS/360 provides facilities for all these, and many other users, and allows them to economically buy time on a computer, providing the same immediate results as with conventional batch-processing systems but without prolonged waiting time.

TSS/360 helps users to concentrate on the solutions to their processing problems with no concern for conventional programming subtleties, system administration, scheduling, or maintenance. In a like manner, TSS/360 assists computer personnel in their specific work: system managers and administrators can easily maintain control over the entire system; system operators are aided in the control and maintenance of the physical devices in the system.

Assistance to Users

The time-sharing system assists users by providing

- direct communication between the user and his executing program;
- ability to share data and programs;
- problem-oriented languages for convenient user-definition of problems;
- facilities for storing programs and data within the system and for testing, modifying, and referring to programs and data during program execution;
- messages to inform the user of the system input required, and to confirm his input or inform him of the system's actions;
- direct access to the full facilities of a large computer through his terminal;
- on-line debugging aids, so that the user can correct errors in his program before actually executing it.

Typical TSS/360 Installation

The personnel in a typical TSS/360 installation are shown in Figure 1; these computer personnel assist the users in system operations.

- *System manager* — has overall responsibility for the installation; one system manager for each installation.
- *System administrator* — responsible for a group of users; one or more system administrators for an installation.
- *System operator* — responsible for operation of the computer and its peripheral devices.
- *System monitor* — maintains the system, and analyzes and evaluates system performance.

The responsibilities of these personnel are described in Section 4, "Use of TSS/360."

Assistance to Manager

The time-sharing system aids the manager of an installation by providing

- direct control over users' access to the system,
- control of user-identification assignments and of accounting procedures,
- facilities to obtain information about every user of the system,
- system administrators to assist the manager in performing his duties.

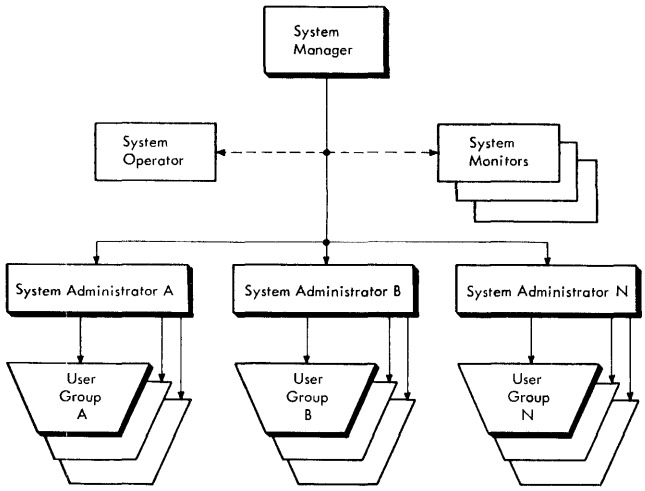


Figure 1. Organization of Typical TSS/360 Installation

Assistance to Operator

The time-sharing system assists the system operator by providing

- facilities to communicate with, and to direct the system,
- facilities to communicate directly with users of the system,
- instructions and messages to assist and direct him in his work,
- system routines to prepare I/O storage media for use by TSS/360,
- log to record data and significant events during system operation.

Basic Concepts and Facilities of TSS/360

TSS/360 provides a variety of direct user-system interfaces, to give the user more direct control of the execution of his programs and to aid him in making necessary modifications. Also, the user's program-system interfaces enable the user's program to interact with the system, at execution time, to permit problem solution without prior resolution of all program definitions by the user.

Direct User-System Communication

TSS/360 interacts directly with the conversational user through

- a separate task for each user,
- remote conversational terminals,
- separate system input and output streams for each conversational user,
- conversational command system,
- a separate profile for each user,
- user-written commands,
- conversational language processors.

Separate Task for Each User

Each TSS/360 user has a separate task established for him when he logs on. A task, which is the basic unit of work for the system, is the action and work that result from the commands issued by a user who wants to obtain the services of the system. Since each user has a separate task, each can operate independently of the others; thus, failure of one user's task does not prevent other users from continuing their communication with the system.

Remote Conversational Terminals

The conversational user of TSS/360 directs and controls his own use of the time-sharing system from his terminal, from which he enters commands, data, and the source statements for his programs. The system, in turn, responds to the user's requests, delivering its output at the terminal in the form of typed responses.

Separate Input and Output Streams for Each User

Each conversational task has a system input stream, and a system output stream, as shown in Figure 2. The input, called the SYSIN data set, contains the user-supplied sequence of commands and data. The output, called the SYSOUT data set, consists basically of system messages; it may also contain messages from problem programs and actual data to be printed at the user's terminal. Because the user's terminal is a combined SYSIN/SYSOUT device, it generates a mixture of the two system streams.

Conversational Command System

The command system is used to direct system operation. The user enters commands to initiate and terminate tasks, execute programs, create and modify data sets, and to perform such operations as obtaining bulk output and copying data sets.

After the command is entered, the system analyzes the user's input. If correct, the system processes the command; if one or more of the user's input operands is incorrectly entered, the system types a pertinent diagnostic message that prompts the user to enter the correct operand.

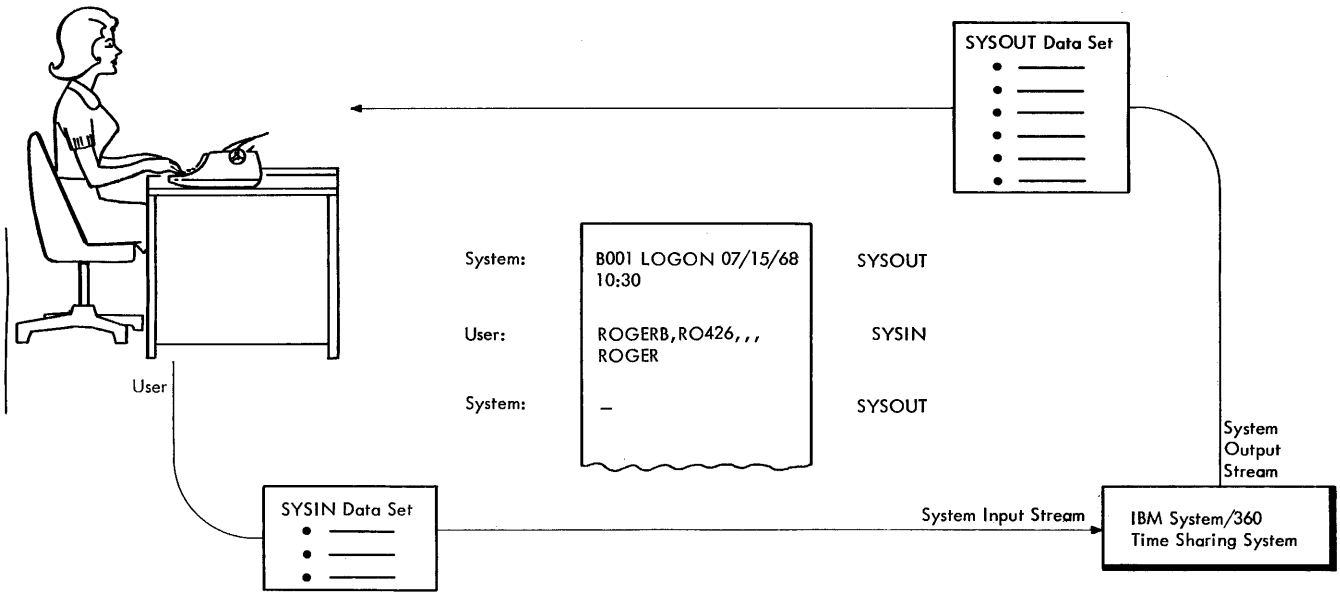


Figure 2. System Input and Output Streams

Conversational Language Processors

TSS/360 processes assembler language and FORTRAN IV source language programs. During conversational assembly or compilation, the system performs on-line syntax analysis of the source statements. Diagnostic messages are provided immediately, if an error in the source coding is detected. The user can then correct the error and modify the source coding, if necessary, before finishing assembly or compilation.

Execution-Time Program Control

In addition to source-language statements for user-program communication, the program-control commands give the user other execution-time program control. The user can insert program-control commands in source programs and thus obtain information on the progress of the program. He can specify, for example, that a program value be printed at his terminal, *or* he can specify that printout occur only under certain conditions. Based on this information, he can make any needed corrections to the program for meaningful results.

Communication Between User's Program and System

The TSS/360 facilities, which allow the user to interact with the system, also provide for the executing programs to call on these system services

- nonconversational program execution,
- on-line storage for libraries of programs and data,
- execution-time program linking,
- data management facilities.

Nonconversational Program Execution

Programs that do not require user intervention can be executed *nonconversationally,* in which mode the program can use all system facilities, except entering into user-system dialogue.

The user can execute a program nonconversationally by executing a previously stored sequence of commands, or by submitting his source statements and commands in punched card format, as a conventional batch job, to the system operator.

The user can include source statements to read records from SYSIN or write records on SYSOUT. In this instance, it is the user's responsibility to ensure that the input stream contains the required records, since he cannot interact with and modify his program during its execution.

On-Line Storage for Libraries of Programs and Data

TSS/360 provides storage and retrieval facilities for user programs. Once the programs are assembled or compiled (and, optionally, combined) they can be stored in libraries for retrieval and execution when called upon by an active program. There are three types of libraries.

- *Job libraries* — data sets created to contain user job libraries that can be used to store programs that

perform specific functions. Unless cataloged, the job library data set is erased when the task is terminated.

- *User library* — a data set for storage of programs that are used in more than one task. Programs in a user library are available to any task established by the owner of the library.
- *System library* — a data set that contains IBM-supplied programs; always available to all users of the system.

In addition to these three libraries, sequences of commands, called nonconversational SYSIN data sets, can be stored in the system as a group. Each sequence is stored in the system and given a group name. These data sets can be used only to control nonconversational tasks. When the user wants to execute a nonconversational SYSIN data set, he specifies its name; the system retrieves and executes it nonconversationally.

Execution-Time Program Linking

A user's program can refer to other programs. The system obtains a copy of the program, places it in his virtual storage (if it is not already there), and resolves linkages and references between the two programs. Thus, the user can obtain the services of the program without coding or entering it in his input stream every time it is needed.

Data Management Facilities

Data management facilities control I/O devices and provide device-independent operation for system routines and user tasks. Effective data management requires the naming of data sets. In TSS/360, a data set is a named collection of related records. Examples of data sets: the conventional input/output files used by data processing programs, a source program, a library of programs and subroutines, and FORTRAN input records.

Device Independence

When a program processes a data set, the data set is referred to by its symbolic name rather than by the name of the input/output device on which it is stored. Unless the user specifies a particular device for a data set, the system will make the assignment. For example, the system assigns an available device to a data set that is to be written on a direct-access device; the user does not specify a particular device. Therefore, a user need not be concerned with the availability of a device or a device type; a program can be executed on systems with different I/O devices or device combinations.

Data Set Names

The user must give a name to a newly created data set that is placed in external storage; the name is used when the data set is to be retrieved.

The name of a data set can be qualified to avoid ambiguity and to distinguish it from another data set. For example, if payroll records for two departments, D561 and D58, were maintained and each had the simple name PAYROLL, the department name could be prefixed to the simple data set name to produce two unique data names — D561.PAYROLL and D58.PAYROLL. Thus, each data set has a unique qualified name, even though each has the same simple name.

Data Set Cataloging

The cataloging facility of TSS/360 aids the user in referring to data sets without specifying their physical locations. A cataloged data set can be referred to by name only; the serial numbers of its volumes are associated in the catalog with the name of the data set, so that the system can locate the data set. All data sets that reside on public volumes (see "TSS/360 Volumes" below) are cataloged by the system when they are defined.

TSS/360 Volumes

Despite the variety, TSS/360 devices for collecting, storing, and distributing data have many common characteristics. For convenience, therefore, the generic term *volume*, which is used to refer to a standard unit of external storage, may be

- a disk pack,
- a drum,
- a reel of magnetic tape.

The volume serial number identifies the volume on which the data set resides. When the system is started, each direct-access device in the system and its associated volumes are designated as *public* or *private*. A public volume can be used by many users concurrently, and any user of the system has access to it; private volumes are restricted to one user. Magnetic-tape volumes are always private.

Data Set Protection and Sharing

To control access to a data set, TSS/360 allows the data set owner (i.e., the creator) to specify which users may share and have access to the data set: either all TSS/360 users, or only a selected group; no other user then has access to the data set. The owner can also specify the level of data set access. If a user has read-only access to the data set, he can read the data set but cannot change it; if he has read-and-write access

to the data set, he can change the data set, but cannot erase it; unlimited access to the data set means that the sharer can treat the data set as his own — he may even erase it. The owner can apply these restricted access qualifications to himself for protection; only he can change access qualifiers to a data set.

Data Access

The TSS/360 data access facilities efficiently schedule and control the transfer of data between storage and input/output devices. The available routines

- read data,
- write data,
- overlap reading/writing and processing operations,
- read and verify volume and data set labels,
- write data set labels,
- automatically position and reposition volumes,
- detect error conditions and correct them when possible,
- provide exits to user-written error and label routines.

Flexibility is a major design principle in the system's data access facilities. The user can employ six data access methods to obtain a group of facilities that best meet his processing requirements. Each access method supplies a comprehensive group of macro instructions that permit the user to specify input/output requests with a minimum of effort. The user need not be concerned with learning the individual access characteristics of the many input/output devices that the system supports.

TSS/360 data management facilities

- permit the user to store, modify, and refer to programs and data using the system storage facilities;
- free the user from concern with specific input/output device configurations;
- permit the user to defer such specifications as device type and length of records in the data set, until a program is submitted for execution;
- permit any desired interchange of programs and data among TSS/360 installations;
- save the time and expense involved in writing routines similar to those provided by IBM;
- allow users to concentrate their programming efforts on processing the records read and written by the data management function;
- provide standardized methods for handling a wide range of input/output and related operations;
- provide the flexibility for including new or improved devices, as they become available;
- provide comprehensive error-recovery procedures.

How Time is Shared in TSS/360

TSS/360 allows many users to have concurrent access to the system by granting each user a "piece" of computer time at intervals. During this time, called a time slice, each terminal user has all the CPU resources of the system, even though he may be physically distant from the TSS/360 installation; he operates as though he alone were using the system. These time slices are provided frequently enough that the system responds to the user's requests as rapidly as the user can respond to the system. The user is thus unaware that other users are also using the system.

The idea of providing intervals of computer time to users, one after another, is not new. Conventional multiprogramming systems allow one task to be active while a formerly active task must wait (for example, wait for completion of an I/O operation). When the event for which the first task was waiting occurs, the task can then proceed; it receives control of the central processing unit, regardless of whether the second task can still make use of the system. The first task, in a sense, has demanded the CPU. This is sharing time on a demand basis.

TSS/360 allows users to share time on both demand and scheduled bases. If a task currently in control of the system must wait for resources, the system may give control to another task, as in a conventional multiprogramming system.

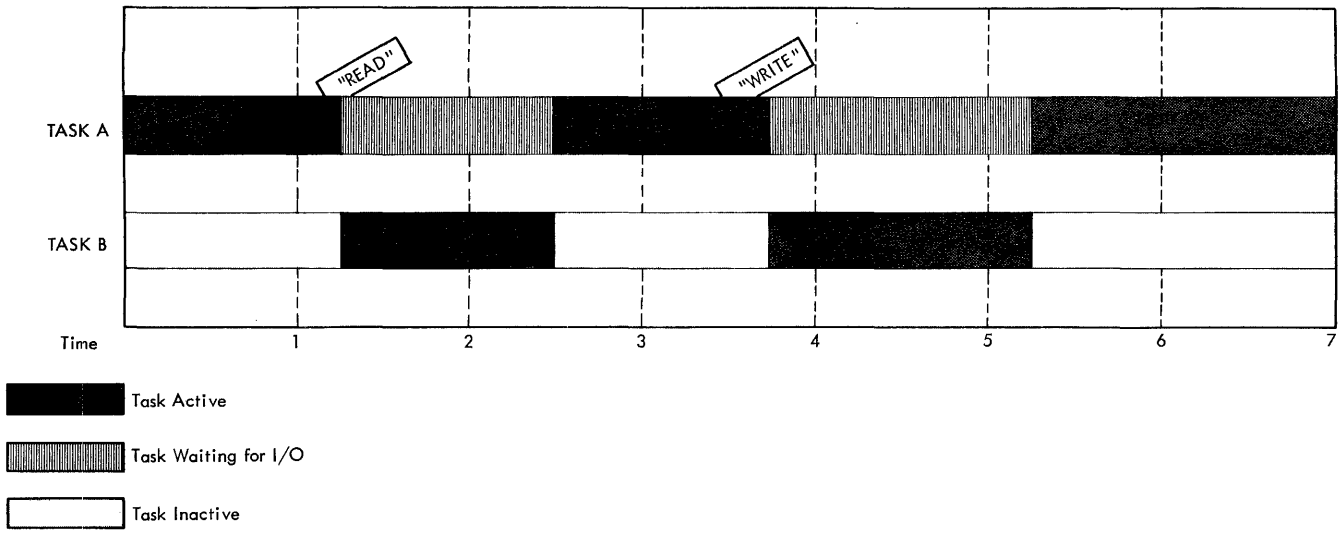
TSS/360 also provides each conversational-task user with a scheduled and specific length of CPU execution time. The computer system thus appears to be working only for each of a number of conversational users, because it can respond to each user much faster than he can communicate with the system. Of course, much more than CPU execution time is shared in TSS/360. Such resources as main storage space, I/O device space, and I/O device-access time are also shared. All these system resources, however, are shared on an essentially demand basis, as in other multiprogramming systems.

The difference between time sharing in multiprogramming and time slicing in TSS/360 can be seen by considering Figures 3 and 4.

Figure 3 illustrates how CPU time is shared between tasks A and B in a multiprogramming system; Figure 4 illustrates how TSS/360 divides time among conversational tasks.

Virtual Storage Concept

The TSS/360 virtual storage concept, a new approach to storage management, provides the most efficient system utilization. Virtual storage is the name given to the address space referenced directly by the processing



Time is shared between two tasks: Task A, a high-priority task, and Task B, a low-priority task. Task A controls the CPU whenever it is able to process data. The sharing of time is at irregular intervals, since it is governed by Task A's processing requirements.

At a point in its processing, Task A initiates a READ operation and is unable to proceed with its processing. Task B is the given execution time, while Task A waits for completion of that READ operation. When Task A's I/O operation is completed and its processing can continue, control is given back to Task A; Task B again becomes inactive. Task A later initiates a WRITE operation and must wait for its completion. Task B again receives control until the completion of Task A's operation, at which time, Task A regains control.

Note: All transfers of control between tasks are made on the basis of a demand priority; a lower-priority task receives control only when a high-priority task is unable to proceed.

Figure 3. Time Shared Between Two Tasks in a Multiprogramming System

unit of a System/360 that is equipped with time-sharing machine features. This address space is as large as the addressing capability of the system; that is, if the user can write an address, the location addressed can be included in his virtual storage. The number of addressable positions is not limited by the size of main storage. The means by which this facility is implemented is described below under "Dynamic Address Translation."

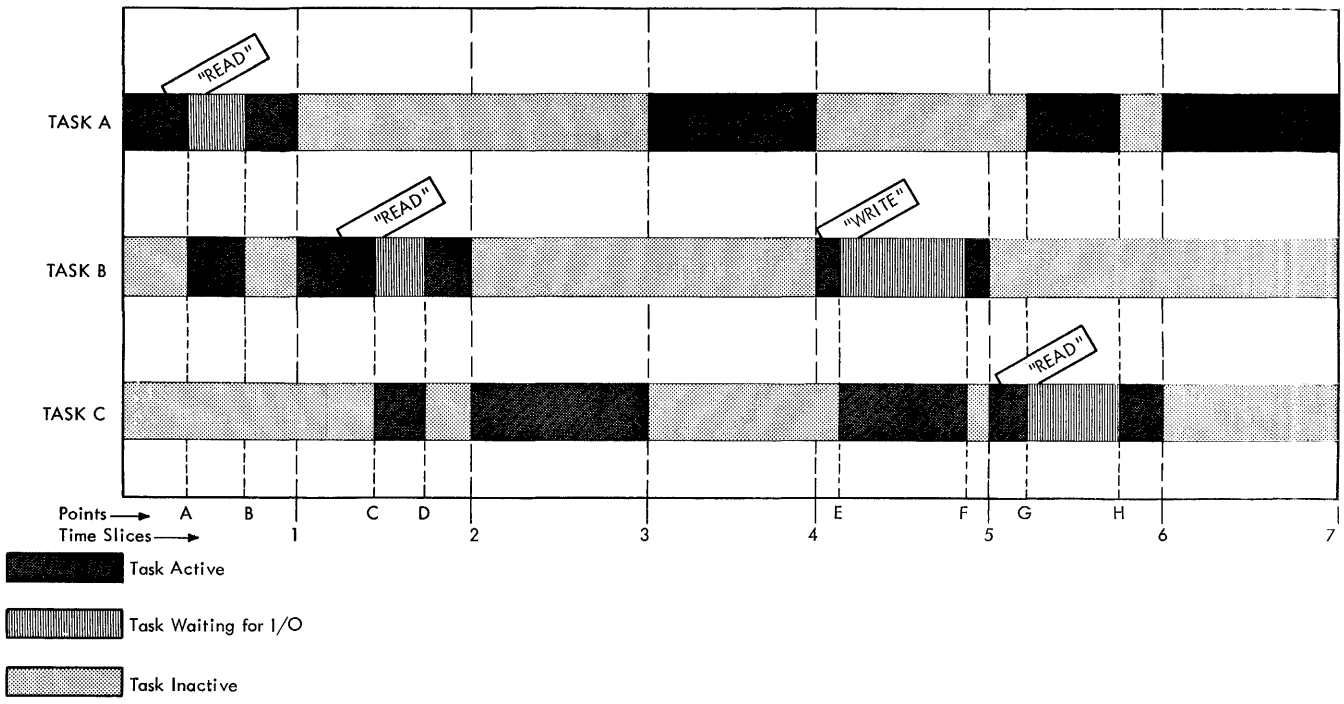
Although the addressing range of virtual storage is equal to the addressing capability of the system, the user is constrained to a virtual storage capacity that is somewhat less than that limit. The constraints are determined by the installation, on the basis of such considerations as main storage capacity, secondary storage capacity, and number of permissible users.

The virtual storage concept in TSS/360 is in marked contrast to planning and utilizing of main storage. Heretofore, storage management presented programmers with a major problem, because of these computer-organization constraints:

- A computer's processing unit can only execute programs from its main storage, usually magnetic core;
- High-speed core storage has insufficient capacity to hold many programs;
- Secondary storage provides sufficient capacity for programs, but it is slower than core storage and is addressed in a different manner.

This computer organization forced programmers to plan storage management with utmost care to use core storage as efficiently as possible, while minimizing the number of references to secondary storage — usually accomplished through program overlays, which are serial uses of a main storage area by different parts of a program. To allow for extreme cases, many programs, such as compilers, performed many overlays. Thus, the preplanning of storage utilization was often wasteful at program execution time, to say nothing of the time spent in planning and programming for two levels of storage.

Efficient use of computers was another problem; very few programs utilized an entire computing system. Multiprogramming operating systems were developed to solve this problem, but they created other



Time is divided, or sliced, in two distinct ways. If a task is unable to proceed, control is transferred to another task in essentially the same manner as in Figure 3 (i.e., on a demand basis). Time is also shared on a scheduled basis.

Task A has initial control of the CPU. At point A, Task A initiates a READ operation and waits for its completion; Task B gains control of the CPU for processing. At point B, Task A's READ operation is completed; Task A regains control. When Task A's time slice is completed, Task B receives control. Task B initiates a READ operation at point C and waits for its completion; Task C is given execution time. Task B's READ operation is completed at point D; Task B regains control. When Task B's time slice ends, Task C gains control and processes through its time slice; Task A then gains control and processes through its time slice. At the end of Task A's time slice, Task B receives control and processes until it initiates a WRITE operation; while Task B waits for completion of this I/O operation, Task C proceeds. When Task B's I/O operation is completed, Task B regains control and processes for the remainder of its time slice. Task C regains control at the end of Task B's time slice; during its processing, it initiates a READ operation. Task A processes while Task C waits for completion of the WRITE operation. When the I/O operation is completed, Task C regains control. At the end of Task C's time slice, Task A regains control and processes for duration of the time slice.

Figure 4. Time Slicing Among Three Tasks in TSS/360

problems for users: If a program uses a large amount of storage, it reduces the number of programs that can coexist with it. The limited number of alternate programs thus makes the multiprogramming ineffective. If many programs are simultaneously in main storage to enable the multiprogramming supervisor to use as much of the system as possible, the individual programs must be smaller or will require more overlays.

Dynamic Address Translation

The TSS/360 virtual storage concept is implemented through dynamic address translation. During the execution of a program that resides in, and refers to, the virtual storage address space, the system translates each virtual storage address into a corresponding real address that designates a physical location, as shown in Figure 5. The contents of virtual storage are formatted into blocks of 4096 bytes, called pages. Each user has his own separate, complete virtual storage-address space.

Before an instruction can be executed, in TSS/360, it must be brought into main storage. Each address is translated as the instruction is executed; the translation process actually amounts to relocation by a value that is a multiple of one page (4096 bytes). As pages are required, they are brought into main storage; when they are not being used and the main storage space is needed by another task, they are written out onto secondary storage unless an exact copy already exists (i.e., the page was not modified during execution). See Figure 6. Some advantages of virtual storage and dynamic address translation are:

- Only the needed pages of a user's virtual storage are in main storage. In Figure 7, the shaded areas represent that part of each user's virtual storage that is actually in main storage.

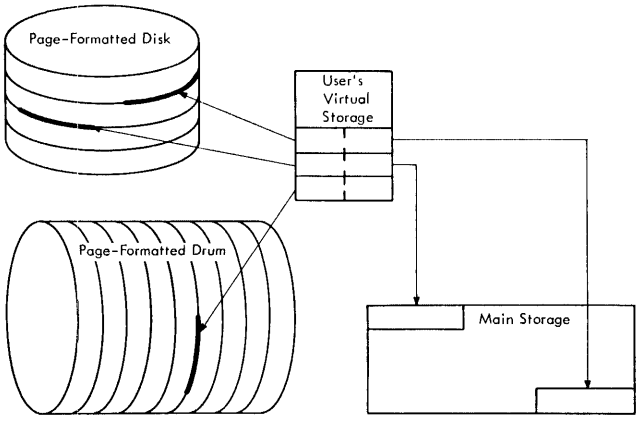


Figure 5. Dynamic Address Translation

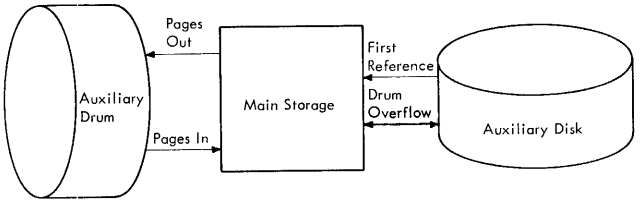


Figure 6. Paging Procedure

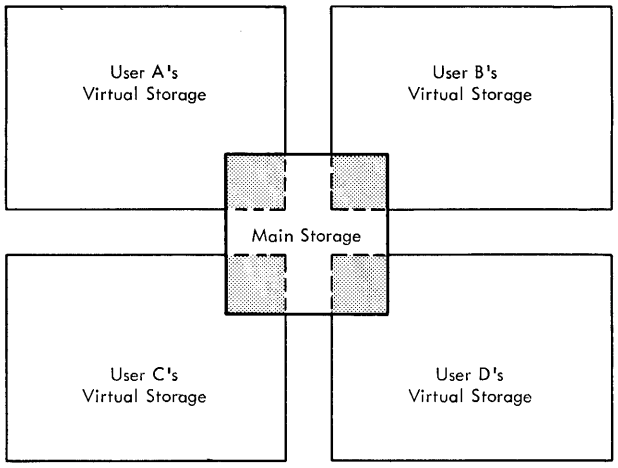


Figure 7. Sharing of Main Storage

- Very large programs can be designed with less concern for the size of the computer's main storage.
- If a user needs a routine (the FORTRAN compiler, for example), a copy of which is already in main storage, he can use this copy, with his dynamic address-translation tables being adjusted to the main storage location of the routine. See Figure 8.
- If programs are subdivided, the individual portions, not the entire program, are all that need to be in main storage at any given time.

As pages that are not in main storage are referred to, the system interrupts the task (the interruption is not evident to the task), reads the page in, and initializes the mapping mechanism to establish the proper relationship between the virtual storage location and the real storage location. This mapping mechanism is implemented by a combination of hardware and programming, which minimizes the overhead caused by address mapping.

Since each task resides in its own virtual storage, each task has an independent addressing structure; i.e., tasks may use the same set of addresses, because the system maps the same address values into different main storage locations for each task. With the map-

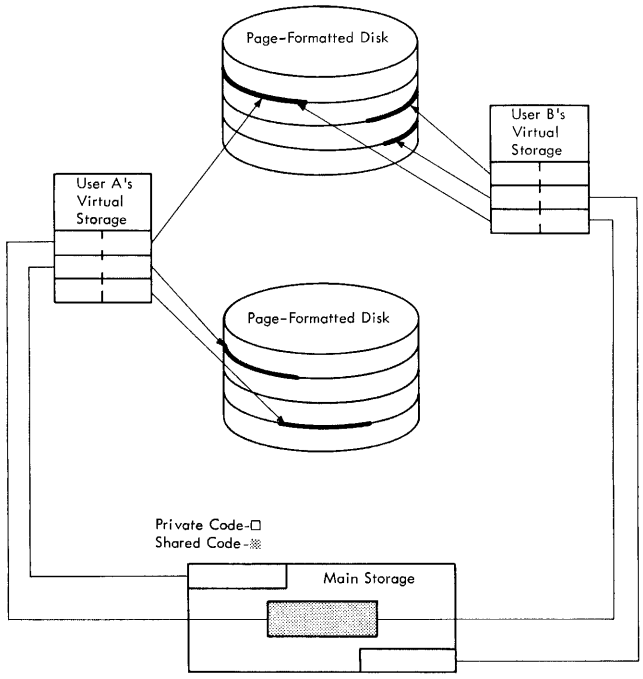


Figure 8. Private Code and Shared Code

ping mechanism, the user is able to design his programs somewhat independently of main-storage addressing constraints; i.e., the address range available to him can be considerably larger than that of main storage. This addressing method simplifies program design, in that the address range available to the user is large enough so that he need not be concerned with the overlay problem. However, he must be aware of storage layout so that paging is minimized. Variable-size data structures are easily handled by managing virtual space in a manner that leaves enough contiguous virtual space to accommodate expansion.

In TSS/360, the supervisor program controls the use of main storage by various users. Users do not, as in operating systems, need to preplan the allocation of main storage.

Multiprocessing Concept

The number of tasks that can be serviced efficiently in a time-sharing system may be increased by adding a central processing unit (CPU) in parallel; the system can contain one or two CPU's.

Section 2: System Summary

TSS/360 is composed of four sets of programs; each set is designed to perform different functions in the system. See Figure 9.

- Supervisor program;
- Privileged service routines;
- Nonprivileged routines, some IBM-supplied (as the language processors), others user-written;
- Supporting programs.

Supervisor	Service Routines	Problem Programs	Supporting Programs
Privileged	Privileged	Nonprivileged	Independent
Permanently in Main Storage	Resides in Virtual Storage	Resides in Virtual Storage	Resides in Virtual Storage
Task Management	Task Management Data Management	Source Language Processors FORTRAN Assembler Service Program Linkage Editor Library Programs User-Written Programs	System Generation and Maintenance Independent Utilities

Figure 9. Categories of Programs in Time Sharing System/360

Supervisor Program

The supervisor program controls the execution of tasks in the system and the hardware environment in which they operate. Although its operations are not apparent to most users of the system, the supervisor creates the time-sharing environment of the system. By means of time slicing, it provides rapid and complete service to a number concurrent users.

The supervisor's basic function is to respond to interruptions (i.e., requests for services) by sorting them as to type and function, and initiating the appropriate routines to respond to each. Supervisor routines also control I/O activity and allocation of machine resources, such as CPU time, main storage space, channels, control units, and devices.

The supervisor, which also provides user-accounting statistics, and performs reconfiguration and recovery functions when necessary, has these important characteristics:

- It resides permanently in main storage.
- Its locations are not addressable by programs operating in virtual storage.
- It executes in the supervisor state.
- It is not time-sliced.
- There is only one copy (with the exception of certain recovery programs); it is used by all central processing units.

Privileged Service Routines

Execution of each task in the system requires a number of service programs; for example, once a task is initiated, routines must be made available to read and interpret its input stream, and to process the task's output stream. These services, as well as those required for execution of programs, are provided by a set of reenterable routines that are made a part of the task's virtual storage, either when the service programs are requested by the user, or when they are indirectly specified by some explicit service request, such as a request for additional virtual storage. Because these routines are reenterable, they can be placed in main storage once and then be shared by all tasks in the system. These routines, however, do not reside in main

storage; they are moved in and out as required. They are privileged, in that they have access to all virtual storage and can use certain supervisor services not available to nonprivileged routines. Routines are provided to aid in management of tasks, data, and programs.

Task Management Routines

Six TSS/360 privileged service routines provide task management functions.

- *Command system* — consists of an interpreter for commands the user has included in his system input stream and for initiating the action requested by each command; also, the command routines to perform the requested actions. The interpreter provides diagnostic messages that inform the user of errors in his commands, prompting messages requesting the user to enter operands, and response messages informing the user of the system's actions.
- *Task monitor* — provides an interface with the resident supervisor for receiving and analyzing task-oriented interruptions, and provides linkage to the required processing service routines.
- *Batch monitor* — controls nonconversational tasks in the system and allocates resources to them.
- *Communication services* — allow the user, during task execution, to communicate with his terminal, the system, the system operator, and the system log.
- *Virtual storage allocation* — allocates virtual storage when needed for a user's task and releases it when no longer needed. The virtual storage may be requested directly by the user or by other routines whose services the user has requested.
- *Device management* — allocates all devices necessary for task execution.

Data Management Routines

Four TSS/360 privileged service routines perform data management functions.

- *Access methods* — provide efficient scheduling and control of the transfer of data between storage and input/output devices.
- *Catalog services* — allow the user to store data so that he can subsequently retrieve it by name only. Also, facilities are provided for sharing data sets.
- *External storage allocation* — processes a user's request for space on a direct-access volume. These routines keep track of the space allocated to a data set and maintain the volume table of contents.
- *Bulk I/O facilities* — provide facilities for writing a data set on magnetic tape or printer, or punching it on cards. Facilities are also provided for reading in a data set from punched cards or magnetic tape, and storing it in the system.

Program Management Routines

Two TSS/360 privileged service routines provide program management functions.

- *Program control system* — allows the user to obtain, dynamically, intermediate results of a program.
- *Dynamic loader* — assigns virtual storage to a task's program modules and adjusts address constants to reflect the module's relocation in virtual storage.

Nonprivileged Routines

There are two categories of nonprivileged routines in TSS/360.

- IBM-supplied processing programs
  Language processors (FORTRAN compiler, assembler)
  Service program (linkage editor)
  Library programs
- User-written problem programs

IBM-supplied and user-written programs are set up and executed in basically the same way by the user.

Language Processors

The language processors have four characteristics.

- They are reenterable, and a single copy in the system can be used by many users; however, the required work areas will be provided by each task.
- They use only direct-access storage for work space; sequential storage (e.g., tape) is not used.
- They include special options that assist in checking out programs.
- Their principal output, called an object module, has a common format that is structured so it can be loaded and executed as a program. An object module can be linked to other object modules either statically or dynamically. The linkage editor can be used to statically link two or more object modules prior to execution; during execution, one object module can be made to dynamically call others for execution.

The operation of each language processor is governed by control information and source statements that the user includes in his system input stream while the task in which assembly or compilation is being performed.

Linkage Editor

The linkage editor, a service routine, is used in association with language processors to perform any of three functions.

- Combine two or more object modules into one object module, prior to execution.
- Define a new entry point for an object module, without reassembly or recompilation.
- Edit control sections, the building blocks of object modules, in any of these ways, without reassembly or recompilation:
  replace, delete, or rename the control sections;
  rename their entry points and external references;
  delete their entry names;
  change their attributes;
  combine one or more into a single control section.

Operation of the linkage editor is governed by control statements which the user includes in his system input stream.

Library Programs

Two types of FORTRAN IV-supplied subprograms are available with TSS/360: mathematical and service. These subprograms are permanently stored in the system, in a library called SYSLIB, and are available to both FORTRAN and assembler language users. The user can provide his own version of a subprogram that would replace the FORTRAN IV-supplied subprogram for that user only.

The user passes one or more parameters to the subprogram, which then performs its operation, using the supplied parameters, and returns a single value to the user.

User-Written Programs

In addition to the programs available with TSS/360, the user may incorporate his own programs into the time-sharing system. Such programs, which may perform specialized operations for an installation, can be assembled or compiled and stored in the system library. They are then available for any user who calls on their services. (For example, an installation might design its own programming language and associated compiler.) The installation can also replace an IBM-supplied program with one of its own (e.g., a different assembler or FORTRAN compiler).

Supporting Programs

There are three groups of TSS/360 supporting programs.

- System generation and maintenance
- Time Sharing Support System
- Independent utilities

These programs operate independently of TSS/360 and perform services for time-sharing operations.

System Generation and Maintenance

In the process of system generation, a time-sharing system is adapted to the needs of a particular installation, including specifications of machine configuration, task management requirements, and command defaults. Performed under a basic time-sharing system, the process allows the user to modify and update TSS/360 and to include installation-supplied programs in the TSS/360 library.

TSS/360 maintenance facilities allow the installation, subsequently, to incorporate both IBM-supplied maintenance distributions and installation-supplied system modifications. System maintenance routines are also used to incorporate the assembled system generation macro instructions into the system to adapt the time-sharing system to the installation's requirements.

Time Sharing Support System

Properly authorized system programmers can use the facilities of the Time Sharing Support System (TSSS) for on-line error analysis and program modification. The principal purpose of TSSS is to allow system programmers to selectively gather data for analysis of system program errors and to dynamically correct such errors while TSS/360 is operating. TSSS provides access to all real, virtual, and secondary storage and to machine registers. TSSS commands may be entered directly from a terminal or may be implanted in system programs and executed when predetermined points are reached during TSS/360 execution. TSSS has been designed to rely as little as possible on other parts of TSS/360; its presence cannot be detected by TSS/360 users when TSSS has not been activated.

Independent Utilities

The four TSS/360 independent utility programs support the time-sharing system, although they operate independently of it. They provide facilities that aid the operator or user to initialize, copy and recopy direct-access volumes, and to copy the contents of all, or part of, main storage onto a direct-access device.

- *Direct-access storage device initialization* (DASDI) — initializes disk and drum storage units to the proper format for TSS/360 use.
- *Dump/restore* (DASDDR) — dumps (copies) the contents of a direct-access volume onto a tape volume or another direct-access volume, and restores (recopies) the contents of a direct-access or magnetic tape volume onto another direct-access volume.
- *Direct-access print* (DADUMP) — prints the contents of direct-access devices.
- *System/360 Model 67 Core Dump* — prints all information pertinent to each CPU in the system and the contents of main storage.

Section 3: Time Sharing System/360 Publications

The publications for Time Sharing System/360, shown in Figure 10, are described briefly in this section. The publications are grouped according to the readership for which they are intended:

- managers and administrators
- operators
- users
- system programmers

This publication, *IBM System/360 Time Sharing System: Concepts and Facilities,* explains the basic concepts of TSS/360, and summarizes the facilities of the system for managers, administrators, operators, and users.

Abstracts of all System/360 publications are contained in *IBM System/360 Bibliography,* Form A22-6822.

Publications for Managers and Administrators

IBM System/360 Time Sharing System: Manager's and Administrator's Guide, Form C28-2024

IBM System/360 Time Sharing System: Terminal User's Guide, Form C28-2017

IBM System/360 Time Sharing System: System Messages, Form C28-2037

Manager's and Administrator's Guide describes the TSS/360 command system facilities that are available to managers and administrators. The manager and administrator use an identical set of commands, although they have different administrative functions. The publication explains TSS/360 system administration; typical terminal session examples are given.

Terminal User's Guide gives instructions for operating the IBM 2741 Communication Terminal and the IBM 1050 Data Communication System in TSS/360. For each terminal, procedures are given for setting up the terminal, entering and cancelling lines, correcting lines, and communicating with the system. Error conditions and termination procedures are also explained. Appendixes explain the character sets applicable to each terminal, and various terminal servicing operations such as ribbon changing, paper insertion, and margin- and tab-stop settings.

System Messages is a compilation of all system messages, completion codes, and storage dumps. This information is given for each system message: message ID, message text, explanation (if not self-explanatory), user response, and system action with and without user response. Completion codes indicate the reasons for abnormal termination; each code is explained, and system action, as well as user response, are indicated where applicable. An explanation of storage dumps and their interpretation is given.

Publications for Operators

IBM System/360 Time Sharing System: Operator's Guide, Form C28-2033

IBM System/360 Time Sharing System: Terminal User's Guide, Form C28-2017

IBM System/360 Time Sharing System: System Messages, Form C28-2037

IBM System/360 Time Sharing System: Independent Utilities, Form C28-2038

Operator's Guide describes the facilities of the TSS/360 command system available to the system operator. It contains an overview of system operations, including functions of the system operator. Available commands, operating procedures, system messages and responses, and typical terminal sessions are given. Nontime-sharing operations are also presented: startup, shutdown, automatic restart, and hardware shutdown.

Terminal User's Guide and *System Messages,* described under "Publications for Managers and Administrators," are also of interest to operators.

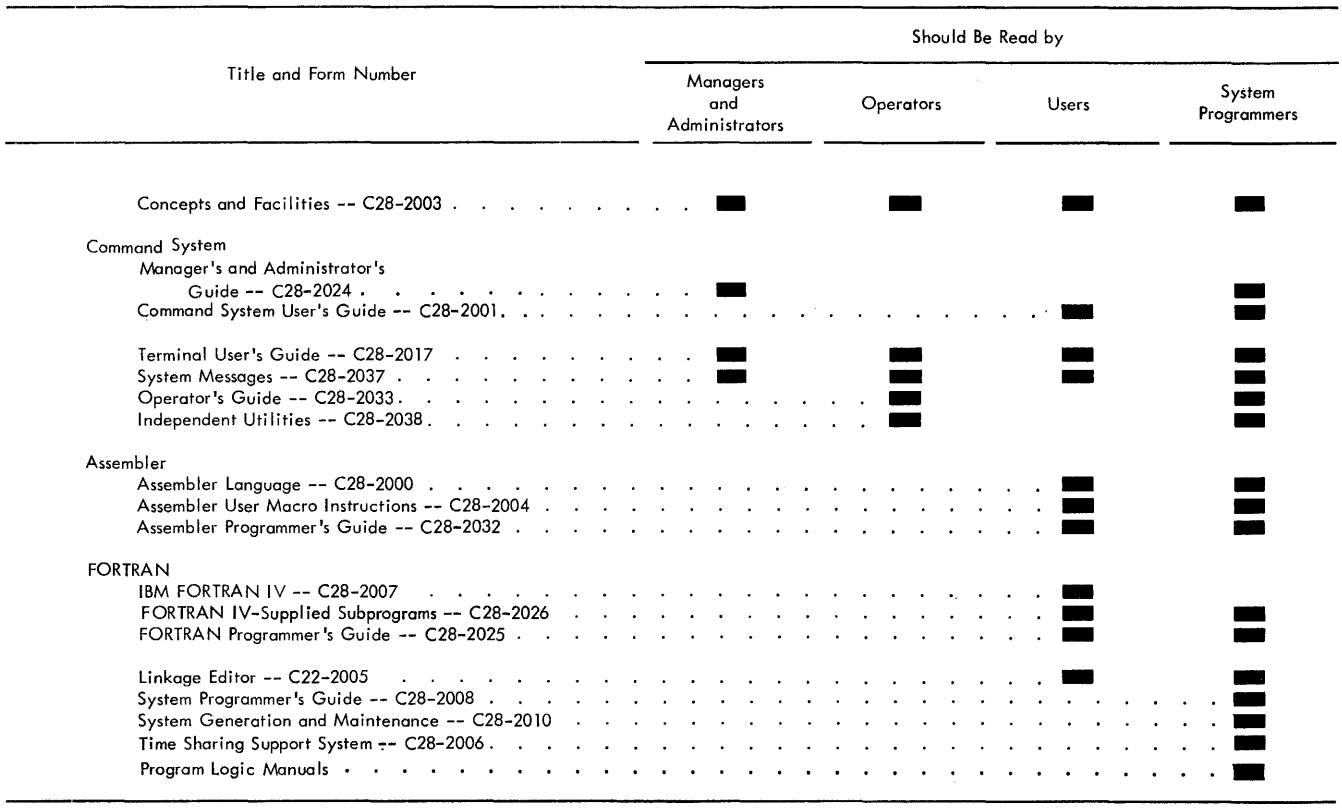


Figure 10. Time Sharing System/360 Publications Plan

Independent Utilities describes the facilities provided by the independent utility programs (dump/restore, volume initialization, and the main storage and drum dumps). It also describes the procedures required to run these programs as stand-alone programs.

Publications for Users

The four publications in this group, intended for FORTRAN and assembler-language programmers, are in addition to the language-dependent publications.

IBM System/360 Time Sharing System: Command System User's Guide, Form C28-2001

IBM System/360 Time Sharing System: Linkage Editor, Form C28-2005

IBM System/360 Time Sharing System: Terminal User's Guide, Form C28-2017

IBM System/360 Time Sharing System: System Messages, Form C28-2037

Command System User's Guide describes the facilities of the TSS/360 command system. It includes a brief description of conversational and nonconversational modes of operation, a detailed explanation of each command, typical terminal sessions, and command procedures. The rules for forming command statements are given.

The use of the TSS/360 linkage editor is explained in *Linkage Editor.* Its functions, data sources and destinations, and system inputs (commands and operands) required for a linkage editor run in both conversational and nonconversational modes of operation are given.

Terminal User's Guide and *System Messages,* described under "Publications for Managers and Administrators," are also of interest to users.

Publications for FORTRAN Programmers

IBM System/360 Time Sharing System: IBM FORTRAN IV, Form C28-2007

IBM System/360 Time Sharing System: FORTRAN IV-Supplied Subprograms, Form C28-2026

IBM System/360 Time Sharing System: FORTRAN Programmer's Guide, Form C28-2025

IBM FORTRAN IV describes the TSS/360 FORTRAN IV language. It includes FORTRAN coding conventions, a discussion of the elements of the language, and a detailed explanation of each of the types of FORTRAN statements. Examples are used to clarify programming rules and to illustrate the various ways in which FORTRAN statements can be written.

FORTRAN IV-Supplied Subprograms describes the subprograms in the IBM-supplied TSS/360 FORTRAN IV library and their use in either a FORTRAN-written or an assembler-written program. The subprograms are divided into two groups: mathematical and service. Library-list control is explained. The concepts of user-written programs (stored in user libraries or job libraries) and IBM-supplied subprograms (described in this publication; stored in SYSLIB), and static and dynamic module combinations are explained.

FORTRAN Programmer's Guide provides tutorial and reference material about TSS/360 from the standpoint of a FORTRAN programmer. Both conversational and nonconversational operations are discussed; rules and conventions are given that must be observed at source coding time to use TSS/360 efficiently. Compiler examples and typical coding sequences are shown.

Publications for Assembler Programmers

IBM System/360 Time Sharing System: Assembler Language, Form C28-2000

IBM System/360 Time Sharing System: Assembler User Macro Instructions, Form C28-2004

IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032

Assembler Language describes the TSS/360 assembler language — its coding conventions and basic statements. The macro language and procedures for its use are described.

Assembler User Macro Instructions describes the TSS/360 macro instructions available to the assembler problem programmer. The types and forms of system macro instructions are explained; each macro instruction is described in detail.

Assembler Programmer's Guide provides tutorial and reference material about TSS/360 as viewed by an assembler-language programmer. Conversational and nonconversational operations are discussed; the rules and conventions are given that must be observed at source coding time to use TSS/360 efficiently. Assembler examples and typical coding sequences are shown.

Publications for System Programmers

IBM System/360 Time Sharing System: System Programmer's Guide, Form C28-2008

IBM System/360 Time Sharing System: System Generation and Maintenance, Form C28-2010

IBM System/360 Time Sharing System: Time Sharing Support System, Form C28-2006

IBM System/360 Time Sharing System: Program Logic Manuals (PLM's)

System Programmer's Guide describes the facilities available to system programmers for modification and extension of TSS/360. It also includes programming guidelines and examples of how to make specific TSS/360 modifications, and a summary of the conventions that were used in developing the programs that are already part of the system.

System Generation and Maintenance explains how to define and generate a time-sharing system adapted to the requirements of a specific installation. It also explains how to incorporate both IBM-prepared maintenance distributions and user modifications into existing systems.

Time Sharing Support System describes the Time Sharing Support System (TSSS) and the commands used to operate it. TSSS is an on-line facility for finding and correcting errors in or damage to TSS/360. The support system can be used only by properly authorized system programmers; therefore, this publication contains no information required by users other than such system programmers.

Program Logic Manuals (PLM's) describe the internal TSS/360 design. They are used for maintaining, updating, and modifying the system components, and for analyzing and correcting programming errors. The manuals are organized to allow a reader to locate quickly, and examine in detail, any sequence of coding in a component. Accordingly, the PLM's serve as guides to the program listing.

The TSS/360 Program Listing

The TSS/360 program listing is a source-language listing of the instructions that make up the TSS/360 system routines. The listing is the most current, up-to-date representation of TSS/360 at a particular installation, since each instruction on the listing has a one-for-one correspondence with the instructions in the routines. If a change in a system routine is to be made, the listing is consulted to see where the change is to be made, and what effect it will have. Comments are inserted throughout the listing to serve as a guide to the functions of each routine.

Section 4: Use of TSS/360

TSS/360 has assigned certain responsibilities to the personnel at an installation so that each individual user can concentrate on his own needs and responsibilities; each user can employ TSS/360 differently from other users. This section describes how managers and administrators, the system operator, system monitors, and users employ TSS/360.

Access to the System

A person who has access to the system is said to be "joined." The procedure used to grant individual access to the system is illustrated in Figure 11.

1. When the system is generated, the system manager, the system operator, and a system programmer with the user identification of TSS***** are automatically joined to the system; they are authorized to use the system.
2. Through JOIN commands, the system manager explicitly joins any system administrators who are to have access to the system. He may also join users and system monitors.
3. The system administrators, in turn, use JOIN commands to grant system access to users.

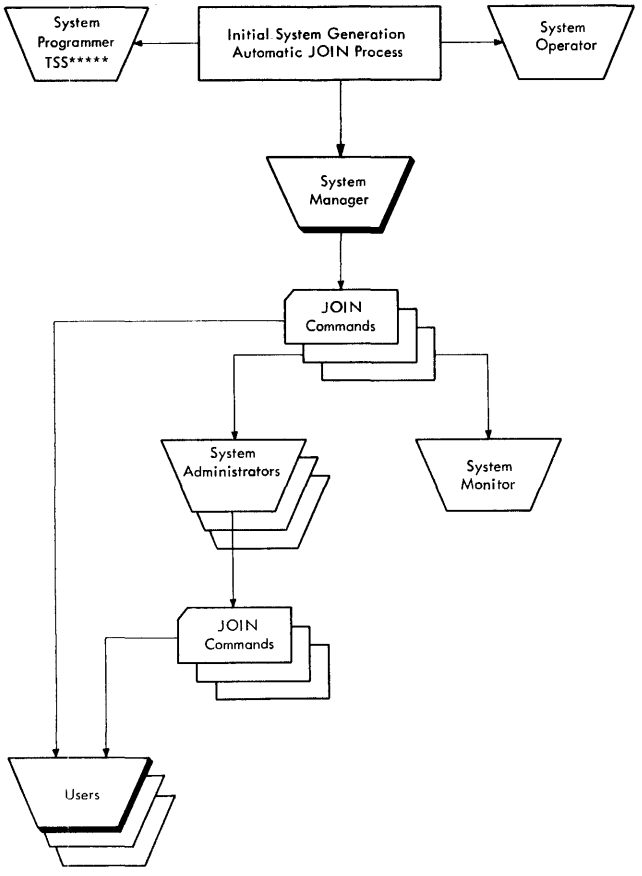


Figure 11. JOIN Procedure

Granting access to use the system amounts to defining the individual to the system. The following information is supplied to the system by the JOIN process:

User identification: Each individual has a unique identification code. In the case of users, the first two characters of the identification of the administrator who joined the user are inserted as the first two characters of the user's identification code by the system. The user must thereafter refer to himself by his complete identification code, which consists of 3 to 8 characters, the first of which must be alphabetic. The set of identification codes is determined by each installation.

Password: A code word containing 1 to 8 nonblank characters. Alphabetic, numberic, and certain special characters (but not right and left parentheses, blank, backspace, comma, equal sign, and tab) are valid. (The character set available depends on the terminal being used. See *Terminal User's Guide.*) The password validates the individual to the system. Passwords are designed within each installation. An individual is not recognized if he cannot supply the current password associated with his identification code.

Charge numbers: The user's numbers; charges for central processing unit usage during a task are accrued against the account number specified by the user for that task.

Priority: A one-digit code, 0 through 9, that specifies the relative priority to be assigned to a person's work; highest priority, 0; lowest, 9. Currently, the system does not use this information.

Privilege Class: Five one-letter codes indicate the specific sets of commands that are available to the person who has been joined to the system.

F: System manager
A: System operator
B: System administrator
D: User
E: System monitor

More than one privilege class may be assigned to certain personnel. If a person, in conversational mode, attempts to issue a command for which he is not privileged, the system issues a diagnostic message and ignores that command; in nonconversational mode, the system terminates his task.

User Authority: A code indicating the level of system protection, for class D users only, as user, privileged system programmer, or nonprivileged system programmer.

From the information given by the administrator when he granted system access to a user, the system can subsequently match that user's identification when he wants to begin using the system, thus validating his right to use the system. JOIN information also establishes the basic accounting records for each user.

NOTE: The system operator is joined at initial system generation time. Subsequently, when he starts the system, he is automatically active and can use the system directly. All other personnel must log-on and validate their identification before they can use the system. Similarly, these individuals must log-off when they have completed their use of the system.

The right of any person (except the manager, the system operator, and the prejoined system programmer) to communicate with the system can be retracted by his administrator or the manager if one of them issues a QUIT command, which has the effect of denying a user access to the system.

Conversational Use of the System

Once a person is granted access to the system, he can have on-line communication through his terminal to all facilities of the system that are applicable to his function as a manager, administrator, operator, or user.

Conversational Task Initiation

To begin using the system in conversational mode, all persons (except the system operator) must log-on.

The procedure varies slightly, depending on whether the terminal has a permanently assigned subchannel for communication with the system, or must be connected via dial telephone.

Hard-wired terminals	*Dial-up terminals*
1. Power switch ON	1. Power switch ON
2. Press ATTENTION button	2. Dial up system (so system can allocate subchannel for terminal's use)

When a person presses the ATTENTION button on his terminal or dials up the system, he begins his system LOGON procedure and initiates a conversational task, which is the work associated with him, specifically, and defined by commands issued from his terminal during a session. A session is the interval between the user's LOGON and LOGOFF. If he has been granted access to the system, and he identifies himself in his LOGON operands, which were set up for him at JOIN time, the system completes the initiation of his task.

When a person's task has been initiated,

- he can converse with the system as if he alone were using it; unique communication paths are set up and routines are loaded that permit the system to read from and write to his terminal;
- he can define work for the system by issuing commands; the required programs and data will be loaded into main storage and processed, as he specified.

Each conversational task has separate system input and output streams. The system input stream contains the sequence of commands, called the SYSIN data set, issued by the user (user's input streams can also include data to be entered into the system). The terminal is the SYSIN device; it is also the SYSOUT device that receives the system output stream. The output (called the SYSOUT data set) consists basically of system messages; it may also contain messages from problem programs and/or actual data to be printed at the user's terminal. Because the terminal is a combined SYSIN/SYSOUT device, it generates a mixture of the two system streams, as shown in Figure 2, in Section 1.

Nonconversational Use of the System

In conversational mode, the person using the system gives commands to the system and can analyze the system's response to one command before issuing another. Many users like this because it permits them to direct the system as they proceed. They can change tactics in solving a problem and possibly modify their approaches as they examine the results of previous actions.

There are many applications, however, where dynamic communication with the system is not required. In these, the nonconversational, or background, operation of TSS/360 can be used. Nonconversational tasks can be set up in the system, and then be executed

concurrently with other tasks, in accordance with predetermined priorities.

These are typical nonconversational uses of TSS/360:

1. Execution of programs in which there are no unpredictable variations in processing.
2. Execution of programs that require a known set of parameters as input, and which then become examples of 1, above.
3. I/O operations involving unit-record devices.

Nonconversational Task Initiation

Nonconversational tasks can be initiated by users, by the system operator or by other nonconversational tasks.

Users issue EXECUTE, PRINT, PUNCH, or WT commands in their conversational tasks.

- EXECUTE names a user-defined nonconversational SYSIN data set that is to be executed as the nonconversational task. It must begin with a LOGON command, end with LOGOFF; it must be prestored in the system so that it can be retrieved merely by naming it.
- PRINT, PUNCH, and WT (Write Tape) function as one-command procedures that initiate nonconversational tasks, which transfer data from a direct-access device to a printer, card punch, or tape unit.

The user can also issue a BACK command to convert a task from conversational to nonconversational mode. This procedure is described in "Mixed Mode Use of the System," below.

User-requested nonconversational tasks are queued and a batch-sequence number is assigned for identification. They are executed when the necessary resources are available.

The system operator issues an RC (Read Cards) or an RT (Read Tape) command from his terminal to initiate a nonconversational task. These commands transfer data from a card reader or a tape unit to a direct-access device.

If the RC command is used to read in a nonconversational SYSIN data set, the nonconversational task is established in the system and executed as soon as resources are available in the system.

The system designates the card reader or tape unit to be used when the system operator's request to initiate a nonconversational task is accepted. The system also provides him with a batch-sequence number to identify the nonconversational task.

Nonconversational tasks can only request the initiation of other nonconversational tasks to transfer data to a printer, card punch, or tape unit. These tasks are initiated by PRINT, PUNCH, or WT commands included in the SYSIN data set of the initiating nonconversational task.

Nonconversational Processing

When a nonconversational task is executed, the command statements are taken, one at a time, from the nonconversational SYSIN data set to direct the required program execution. Data associated with commands can also be read from the SYSIN data set, if the user properly positioned the data in the SYSIN data set.

The system automatically defines a SYSOUT data set for each nonconversational task. When the task is completed, the system prints out that data set. For nonconversational tasks, the SYSOUT data set consists of diagnostic messages, the commands from SYSIN that were executed, and any data that the user, in his program, wrote to SYSOUT.

Nonconversational Task Termination

Execution of a nonconversational task initiated by EXECUTE, RC, or BACK (see "Mixed Mode Use of the System," below) is terminated when the LOGOFF command in its SYSIN is executed. The RT, RC (for data), PRINT, PUNCH, and WT tasks terminate when the data transfer is completed. Nonconversational tasks can also be terminated by use of the CANCEL command.

Mixed Mode Use of the System

A user can begin a task, conversationally, at his terminal and later issue a BACK command to have the task's execution completed in nonconversational mode. He must previously have prestored a SYSIN data set for that nonconversational portion of the task. That data set must not contain a LOGON command (because the user has already logged-on), but it must end with a LOGOFF command.

After the BACK command has been executed, the user must log-on again at his terminal if he wants to initiate a new conversational task, after waiting several seconds.

Use of TSS/360 by Managers and Administrators

The system manager is responsible for the overall management of a TSS/360 installation. He was automatically joined at system-generation time, so he can proceed to direct the system once he has logged-on. In large installations, the system manager may join one or more system administrators to aid him in his administrative duties.

The system administrator's duties are similar to those of the system manager. Whenever the time-sharing system is operating, the system administrator can log-on and perform the functions that the system manager delegated to him, by issuing the appropriate commands that are reserved for system administrators.

The system manager and the system administrators can also have the privileges of the users by joining themselves to the system under the user-privilege class. Operating in this capacity, they have the same facilities available to them as the users have.

System Manager's Functions

1. Directing initial system generation. The manager's user identification, charge number, priority, privilege class, and authorization are preset by the system; however, he must establish his own password.
2. Joining one or more persons to the time-sharing system, after it has been generated. The manager may join system administrators and system monitors, as well as users. If the manager wants to assume some other responsibility (e.g., the functions of a user, to run an installation's accounting program), he may join himself with that privilege class and assign a different user identification to himself.
3. Withdrawing other users' permissions to use the system. He may withdraw this permission from anyone but himself, the system operator, and the prejoined system programmer.

An example of a typical terminal session for a system manager is presented in Figure 12. For details on the commands used to direct the system, refer to *Manager's and Administrator's Guide.*

System Administrators

A system administrator exercises the authority delegated to him after being joined by the system manager. Thereafter, he can log-on and perform his allocated functions by issuing the appropriate commands to the system.

A system administrator has two primary functions.

1. Grants users, under his control, access to the time-sharing system. (Note that, unlike the system manager, the system administrator can only join users and system programmers.) The system administrator allocates charge numbers to the users, against which their CPU time is billed.
2. Denies further access to the time-sharing system by one or more users previously joined. The system administrator can deny access only to users who are under his administration — those he has joined.

LOGGING ON	
Manager:	Identifies himself as system manager; he cannot start a task until he properly identifies himself. He must log-on with the user identification, charge number, and password assigned to him at system-generation time.
System:	Confirms the manager's entries.
USER CONTROL	
Manager:	He can join or quit users who are under his control. When he joins a user, he assigns a user identification, password, charge number, priority, privilege class, and authorization (as either user, system programmer, or privileged system programmer). When he quits a user, only the user's identification code is given; he can quit anyone except himself, the system operator, and the prejoined system programmer.
System:	Establishes control information for a user who is joined so that he can subsequently log-on. When he has been denied access to the system, his data sets are disposed of; then, he cannot log-on until he has been rejoined.
DATA SET INQUIRY	
Manager:	Can request the status of any data set in the system, or that a line from any data set be printed at his terminal.
System:	Prints the requested data-set status or line, at the manager's terminal.
LOGGING OFF	
Manager:	Informs system that he no longer wants to use it.
System:	Removes control information for manager's task from the system. If manager was under the user-privilege class, the system requests the disposition of all uncataloged data sets.

Figure 12. Example of System Manager's Terminal Session

Because managers and administrators must direct and control the time-sharing system, they usually remain on-line to it (i.e., operate in conversational mode). The JOIN and QUIT commands are not valid in nonconversational mode.

For some applications, however, the manager or administrator may want to operate nonconversationally. *Example:* If the manager wants to obtain a printout of a large data set, he could submit his request (as a card deck) to the system operator to enter the deck as a nonconversational task.

The manager or administrator terminates his conversational tasks by entering a LOGOFF command. Later, if he wants to initiate a new task, he must log-on again.

Use of TSS/360 by System Operator

The system operator is responsible for operation of the TSS/360 computer and its peripheral equipment. He communicates with the system by typing commands at his terminal, to

- Start the system and shut it down,
- Adjust the system configuration,
- Enter bulk input from cards or tape,
- Terminate tasks,
- Communicate with users and the system,
- Maintain the operator log.

An example of a system operator's terminal session is presented in Figure 13.

System Startup and Shutdown

The system operator is responsible for starting up the system; he selects the basic components – central processing units, main storage units, channel controllers, and I/O channels.

The system operator is responsible for system shutdown. Usually, he informs all conversational users in advance, so they can terminate their own tasks before the system is shut down. He makes this announcement by issuing a BCST command, which sends a message to all active conversational users. At the specified time, he issues the SHUTDOWN command to terminate all tasks – both conversational and nonconversational – that are still active in the system. Then he shuts down the system.

Configuration Control

The system operator is responsible for establishing and maintaining the configuration of the components of the time-sharing system. At startup time, he selects the basic configuration of the system; thereby, he partitions

BULK INPUT INITIATION

Operator:	Initiates bulk input operations, used to enter data into the time-sharing system. The input may be on punched cards or magnetic tape; it may be data or (for punched cards only) a sequence of commands to be executed as a nonconversational task.
System:	Designates the card reader or tape device to be used as the input device. Any errors detected in the input are listed on the SYSOUT for the bulk input operation, not at the operator's console.

DEVICE CONTROL

Operator:	Can specify that a device be disassociated from the system, making it unavailable for use. This allows limited maintenance to be done on the malfunctioning device without affecting TSS/360. At a later time, he can make the device available again.
System:	Confirms the request and disassociates the device or reinstates it.

COMMUNICATION

Operator:	Can send messages to all active conversational users or to specific users.
System:	Suffixes the current time and date to the message and transmits it.

USER TASK CONTROL

Operator:	Can request cancellation of any conversational or nonconversational task currently in the system.
System:	Cancels the specified task and informs the operator; no message is sent to the user whose task was canceled.

SYSTEM LOG PRINTOUT

Operator:	Can request that the contents of the system log for any previous session be printed out.
System:	The specified session's log is printed out at the operator's terminal.

SYSTEM SHUTDOWN

Operator:	Enters the SHUTDOWN command.
System:	Terminates all conversational and nonconversational tasks in the system, and prohibits the starting of new tasks. A message explaining the reason for cancellation is printed at every active terminal, and on the SYSOUT of every nonconversational task.

Figure 13. Example of a System Operator's Terminal Session

the equipment available at the installation. Components can then be made available for other than time-sharing operations. (Example: IBM System/360 Operating System can use the part of the equipment not used by TSS/360.)

The HOLD and DROP commands are available to the system operator for control of I/O device assignments. HOLD disassociates one or more devices from the time-sharing system; it can be issued at any time after system startup. DROP cancels the effects of a previous HOLD command; it makes the device available again for system use. Thus, a malfunctioning device does not require system shutdown; instead, the device can be made inactive by HOLD, investigated, and later returned to active status by DROP.

Bulk Input Initiation

The system operator is responsible for initiation of bulk input operations, which are used to introduce data into the time sharing system from punched cards or magnetic tape. He starts a bulk input operation from punched cards by issuing an RC command. Of course, he is not responsible for the correctness of the input; the card deck must contain all the information needed by the system to execute the operation properly. The information entered by RC can be data, or a SYSIN data set that is to be executed as a nonconversational task. He starts bulk input from magnetic tape by using an RT command, giving the information supplied by the owner of the tape as operands. Unlike RC, RT can be used only to enter data. An example of input entered through RC is presented in Figure 14.

The system operator enters an RC command from his terminal, as part of his SYSIN (Task 1 in Figure 14). When the system designates a card reader for the task, the cards are read in *by a separate nonconversational task* (Task 2 in the figure), and a virtual sequential or virtual index sequential data set is created. This task (Task 2) has its own system output stream; errors arising from the bulk input operation are sent to the SYSOUT of the nonconversational task, not to the system operator's terminal. If RC is used to read in a data set, the data set is cataloged for subsequent retrieval by the user. If a SYSIN data set is read in, a batch-sequence number is assigned to it, and it is executed as a nonconversational task as soon as the required resources become available. This nonconversational task (Task 3 in the figure) has its own SYSIN and SYSOUT, separate from the other tasks associated with the RC operation.

A separate data set is created for every task, from LOGON to LOGOFF, that is in the input stream, to allow the input to be stacked for submission.

Terminating Users' Tasks

The system operator can cancel any task in the system prior to its normal end. Any user may request that his conversational task be canceled if, for example, he loses control of it and is unable to reestablish contact; then, the system operator issues a FORCE command to cancel the task. The system operator might cancel a nonconversational task if it seems to be in an infinite loop; he issues a CANCEL command to terminate the task.

Communication with Users and the System

The system operator is responsible for replying to messages that are printed out at his terminal and that demand a response. If the message is prefixed by a four-digit identifying number, he responds by means of a REPLY command; otherwise, he merely types his response.

The system operator can also initiate messages to other conversational users. He uses the BCST command to send a message to all active conversational users, and the MESSAGE command, to a specific conversational user.

Maintaining the Operator Log

The operator log, which is a record of all communications between the operator and the system, is maintained by the system:

- All messages issued to the system operator.
- All replies to these messages.
- All messages that the system operator sends via MESSAGE and BCST commands.

A listing of the log of the previous session, from startup to shutdown, is printed automatically each time the system is started up. If the system operator wants the log of any other session, he can issue a PRINT command to obtain that listing.

Utility Functions

The TSS/360 independent utilities allow the system operator to initialize volumes for system use, copy the contents of one volume onto another, and obtain printouts of the contents of a direct-access volume or of main storage.

Use of TSS/360 by Users

TSS/360 supports many different types of concurrent users. Unlike earlier systems, which were geared primarily to programmers, TSS/360 provides services to users who may have little or no knowledge of pro-

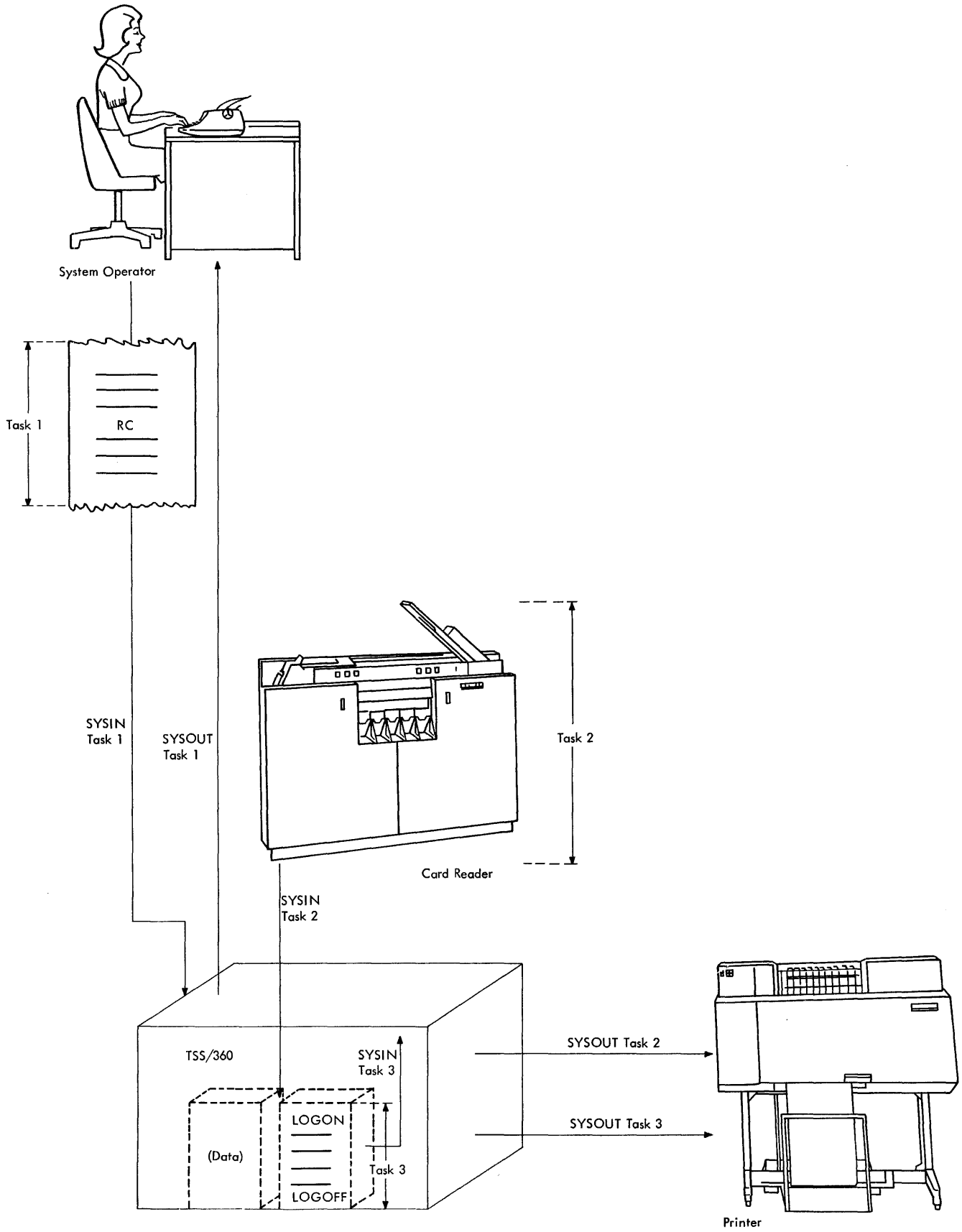


Figure 14. Bulk Input Initiation – RC (Read Cards)

gramming. Thus for TSS/360, the term "user" means anyone who potentially could communicate with the system. Active users are currently working at their terminals or have tasks that are being run in nonconversational mode.

Typical users who could be concurrently using the system are

- programmers,
- students,
- publications personnel,
- information-retrieval users,
- programming support personnel.

Programmers who use a computer source language to formulate problems that are to be solved by the system are either problem programmers or system programmers.

- Problem programmers are concerned with getting applications work done by the system, not with administering, modifying, or maintaining the system. A problem programmer designs programs for processing data or for solving specific computational problems. He may be a professional programmer, or an engineer or scientist who engages in programming only when it is required to obtain results related to his job.
- System programmers are more experienced professional programmers who may design problem programs but, generally, are concerned with maintenance and modification of the system. A system programmer must be acquainted with the internal operations of the system, as well as its use. Typically, he would use the computer to design programs to replace or modify existing system programs, or to add new facilities to the system.

Other users do not write programs; in fact they may not have any knowledge of programming. They use simplified languages to define their requirements to the system; then special routines (written by programmers) automatically process those requirements. The only knowledge of computers needed by these users is the requirements for information by the special programs they use.

Students are examples of the nonprogrammer user. They can use the system's special installation-supplied programs as their instructor. Also, they can use TSS/360 to solve mathematical and engineering problems.

A student user does not need to know the details of TSS/360 operation. In Figure 15, after logging on, the student requests that the teaching program (supplied by his installation) be initiated. The student then engages in a dialogue with the system: The system types the first question at the student's terminal and requests

LOGGING ON	
Student:	Identifies himself to the system so he can start his work.
System:	Confirms the student's entries and establishes a task for him.
INITIATE INSTALLATION-SUPPLIED TEACHING PROGRAM	
Student:	Enters a command to start execution of a teaching program prepared by his installation.
System:	The specified program gets control when it is placed in the student's virtual storage.
USER/SYSTEM DIALOGUE	
System:	Begins its dialogue with the student by printing the first question in the programmed instruction course at the student's terminal, and requests him to enter his answer.
Student:	Enters his answer.
System:	If the student entered the correct answer, the next question will be typed out. If the student entered an incorrect answer, he will be informed and requested to answer the question correctly. When the answer is correct, the next question will be typed out.
Student:	Continues to answer system's questions.
LOGGING OFF	
Student:	Informs the system that he no longer wants to communicate with it.
System:	Terminates the student's task.

Figure 15. Example of Student-User's Terminal Session

him to enter his answer. If the student enters the correct answer, the system types the next question. If the student enters the wrong answer, the system indicates his error or, possibly, gives him a hint to enable him to find his error. Then he is again requested to enter the correct answer. When the student enters the correct answer, the system prints out the next question. The student and system continue in his manner until the student either completes the course or logs off.

Manuals or other documents, prepared at the installations, will describe the details of their own programs; this student example merely illustrates a typical nonprogrammer's use of TSS/360.

Publications personnel can employ user-supplied routines in the system to store text, modify it, print out all

or part of it. The format of the printed material can be varied to suit individual requirements. The user need not be concerned with the effects of text modification on output format, since the routines adjust the format accordingly.

To information-retrieval users, TSS/360 has the effect of a large library that can be searched to produce answers to specific inquiries. A user-supplied data base (e.g., a research, engineering, legal, or medical library) is stored in the system. Users supply the system with selected keywords describing their inquiries. The system then employs a user-supplied program to search the data base for answers to the inquiries.

Programming support personnel can also use the system to perform specialized functions that can assist other users in their activities. Example: keypunch operators or typists can insert (prestore) data and programs in the system for later use by programmers or other users. In many applications such as entering data, facilities available in TSS/360 can be used directly; in other applications, user-supplied problem programs may be required.

Programmers have the full facilities of TSS/360 available to them. They use the time-sharing system to translate their source-language programs into machine language, test their programs, correct any errors in them, and execute the programs to obtain the desired results. Programmers can make use of all the data management facilities in the system.

An example of a programmer's terminal session is presented in Figure 16. The details of task and data management are explained in the following sections.

LOGGING ON	
User:	Identifies himself to the system.
System:	If the user has been validated to the system, control information is established for him so he can start his work.
COMPILE	
User:	Calls for the services of the FORTRAN compiler; then he enters his source statements from the terminal. He must now define any data sets that will be used by his program.
System:	Informs the user of syntax errors in his source statements, and tells him if the source statements can be compiled. The user is asked if he wants to make any modifications to his source statements. After all have been made, the source program is compiled.
DEBUG	
User:	Can insert statements in his source program to display intermediate results, or point out error conditions.
System:	Displays any requested information and allows the user to make modifications, if required.
EXECUTE	
User:	After the program has been debugged, it is ready for execution to obtain the desired results.
System:	Executes the program, but still allows the user to interrupt with changes.
DISPOSITION OF DATA SETS	
User:	When program execution is completed, the data sets may be erased or private data sets may be cataloged.
System:	Displays the name of each data set and the dispositions.
LOGGING OFF	
User:	Informs the system that he no longer wants to use it.
System:	Requests the dispositions of any private data sets that are still uncataloged; control information for the user's task is removed from the system.

Figure 16. Example of Programmer-User's Terminal Session

TSS/360 tasks may be executed in either of two modes: conversational or nonconversational. During conversational task execution, the user remains in communication with his task, obtaining intermediate results and modifying his program while it is executing. There is no communication, however, between the user and a nonconversational task; no task output is available until the task has been completed or is terminated. TSS/360 also allows the user to switch a conversational task to nonconversational, when user-task communication is no longer needed.

Conversational Mode

In conversational mode, the user communicates with the system through a terminal, such as the IBM 1050 System (with or without an IBM 1056 Card Reader) or the IBM 2741. The terminal may be located within the area that houses the computing System/360 Model 67, or it may be located in another area that is remote to the central computer complex. The user, no matter where he is located, communicates with the system through a typewriter-type terminal keyboard or, if his terminal equipment includes an IBM 1056, through a card reader. The system communicates with the user by printing out messages and data on his terminal printer. The system can, therefore, turn to the user to resolve any problems that arise during conversational task execution. Similarly, the user is free to change the activities he requested of the system (including the execution of his problem programs), as his needs vary or as he receives processing results during task execution.

Conversational Task Initiation

Before he can initiate a conversational task, the user must be granted authorization to use the system by his administrator or manager (refer to the description of the JOIN procedure in Section 4). The user initiates a conversational task by turning on his terminal and dialing up the system or pressing the ATTENTION button. Then,

1. A conversational task is initiated for him, on the assumption that he wishes to log-on, and the system I/O streams are set up as required for user-system communication.
2. The system prompts the user to enter his LOGON command operands.
3. After the LOGON procedure is completed, the system automatically executes the ZLOGON command. This command is provided so that the user can augment the system's initialization of his task. Originally it is defined to do nothing. The user can redefine it with the SYNONYM, PROCDEF, or BUILTIN commands (refer to "Tailoring TSS/360" at the end of this section) to perform the functions of another command, a sequence of commands, or a user-written command.

Conversational Task Execution

After the user has logged-on, the system asks him to enter his next command and, in effect, enters into a conversation with him. The user's portion of this dialogue consists of his commands and any source language statements that he enters during execution of his task, and his replies to the messages issued by the system. The system's contribution to this dialogue consists of messages, responses to commands, and requests for next commands.

Requests for Next Command

The system informs the user that it is ready to accept his next command by printing, at his terminal, an underscore character (_) beneath the first character position of a new line. (The same indication is given even when the user is entering his commands through the terminal card reader.)

Entering Commands

Every command entered from the terminal keyboard starts on a new line. It may begin above the underscore

that requests its entry or at any other point on the line. The end of the command is indicated by pressing the RETURN key. If a command cannot be entered on a single line at the terminal, it may be broken and entered on more than one line. A continuation character informs the system that the command is continued on the next line. (The RETURN key must still be pressed to mark the end of the line.)

Commands entered through the card reader attached to the IBM 1052 can be punched in any column. Continuation conventions are the same as those for the keyboard.

SYSIN

The user's series of commands, issued to direct the system's execution of his task, is called the SYSIN data set, which is not recorded by the system in any form other than as a listing printed at the terminal. In addition to commands, the SYSIN data set may include data.

Command Execution

Every command entered by the user is executed interpretively. Each one is analyzed to determine if it is valid; if it is, all the actions requested by a command are performed before the user is requested to enter the next command. Conversely, if a command is not complete or valid as entered, the system issues messages to request the user to supply additional information before the command is executed.

The system issues three types of messages to the conversational user.

- Prompting messages
- Response messages
- Diagnostic messages

Prompting Messages

Prompting messages normally ask the user to supply command operands or additional information under these conditions:

1. When a mandatory operand has been omitted.
2. When the user's conversational task has been terminated, the system always asks for the disposition of his uncataloged data sets.
3. When the user has omitted a DDEF command for a data set that he refers to.

Response Messages

Response messages generally tell the user of the actions the system has taken in executing a command.

Diagnostic Messages

Diagnostic messages inform the user of errors he made in entering a command and of source language errors; the messages ask the user to correct his errors.

SYSOUT

The SYSOUT data set consists of the system messages and messages that respond to command execution, developed during execution of a user's task. In conversational mode, the information in SYSOUT is printed at the user's terminal but not recorded by the system in any form except the printed listing at the terminal. SYSIN and SYSOUT are thus interspersed on the terminal listing, together with problem-program communication with the user.

Conversational Task Interruption

The user can interrupt execution of his conversational task by pressing the ATTENTION key at his terminal. If the system has not begun execution of a command (because it has not been entered completely), the ATTENTION signal cancels that command. However, if the command is being executed when the ATTENTION key is pressed, the effect of ATTENTION depends on the command. See Table 1.

Conversational Task Termination

The user can end his conversational task by

1. Entering a LOGOFF command at his terminal, or
2. Entering a BACK command to switch to nonconversational mode.

When the LOGOFF command is entered, the system issues a message asking the user how he wants to dispose of his uncataloged data sets. Uncataloged data sets exist on private volumes only. They may also be cataloged, erased, or ignored. If the user decides not to catalog or not to ignore these data sets, the disposition depends on the type of device: Data sets residing on direct-access private volumes are erased; no action is taken for data sets on magnetic-tape volumes. If the user chooses to take no action for a new private data set, a list of the volumes on which the data set resides is printed out for him. He may specify the same disposition for all his uncataloged data sets, or he may choose to have the data sets' names presented to him, one at a time, to give separate disposition instructions to the system.

The user can issue a BACK command to switch a conversational task to nonconversational mode; he can then initiate a new conversational task by logging on again at his terminal (see "Switching Modes").

The system will terminate a task when an uncorrectable error occurs. For conversational tasks, the user can use the DUMP and DISPLAY commands for problem investigation; these commands are used to obtain the contents of specified data areas so that changes to the

program can be made. For nonconversational tasks, control is transferred to a data set defined with a DDEF command that has a DDNAME operand of TSKABEND, if supplied; this data set can contain a sequence of program-control commands and statements to obtain a selective dump of the program before terminating the task with a LOGOFF command.

The user can specify a maximum execution time for his task by issuing a TIME command. If his task has not completed at the end of the specified time interval, a message is issued, and control is returned to the user so that he can enter commands from his terminal.

The user can issue an ABEND command in situations where other recovery procedures are not adequate and he must get out of an unresolvable situation. The ABEND command eliminates all system and user programs that exist in the user's virtual storage; the user's task is returned to the status that existed immediately after he logged-on. The ABEND command can be entered by typing ABEND on the terminal keyboard or induced automatically by pressing the terminal ATTENTION key five consecutive times.

Nonconversational Mode

In nonconversational mode, there is no direct communication between the system and the user. The manner in which the user defines what is to be done in the task varies with the type of nonconversational task, as described under "Nonconversational Task Initiation." Any system messages that develop during execution of the task are received by the user as printouts from the central computer installation.

Nonconversational Task Initiation

Nonconversational tasks can be set up in the system by the user or by the system operator.

The user may set up a nonconversational task by issuing any of these commands: PRINT, WT (Write Tape), PUNCH, or EXECUTE.

- For PRINT, WT, or PUNCH tasks, the associated command completely defines the nonconversational task for the system.

Table 1. Effect of ATTENTION Interruption

Situation When ATTENTION Interruption Is Issued	Effect of Interrupt
A new command is being typed in, but the RETURN key has not yet been pressed to end the line.	All characters entered for the command are canceled. The user may retype the command after the system prompts him.
The system has just issued a message that asks the user to type in some information.	The command in operation (i.e., the last one transmitted to the system) is canceled. The user, after the system prompts him, may then type in his next command or may repeat his previous command. *This use of ATTENTION is important:* it allows the user to break out of loops formed when the system prompts for an operand the user is unable to give.
A command is being executed.	In most cases, the command is completed before the ATTENTION interrupt is recognized, but there are several exceptions to this. See *Command System User's Guide.*
A message is being printed out on the terminal.	The remainder of the message is canceled, the system prompts, and the user may then enter his next command.
A language processor (such as the assembler or compiler) is in operation.	Language processing stops at the point of interruption, and the system requests the user's next command. He can then enter any commands. If he has not interrupted anything else, he can then issue a command to restart language processing from the point of interruption.
A user program is in operation.	The effect is the same as for a language processor. The user can interrupt and then restart at the point of interruption.
Note: If the user creates his own ATTENTION interruption handling program, the response to the interruption will be determined by that program.	

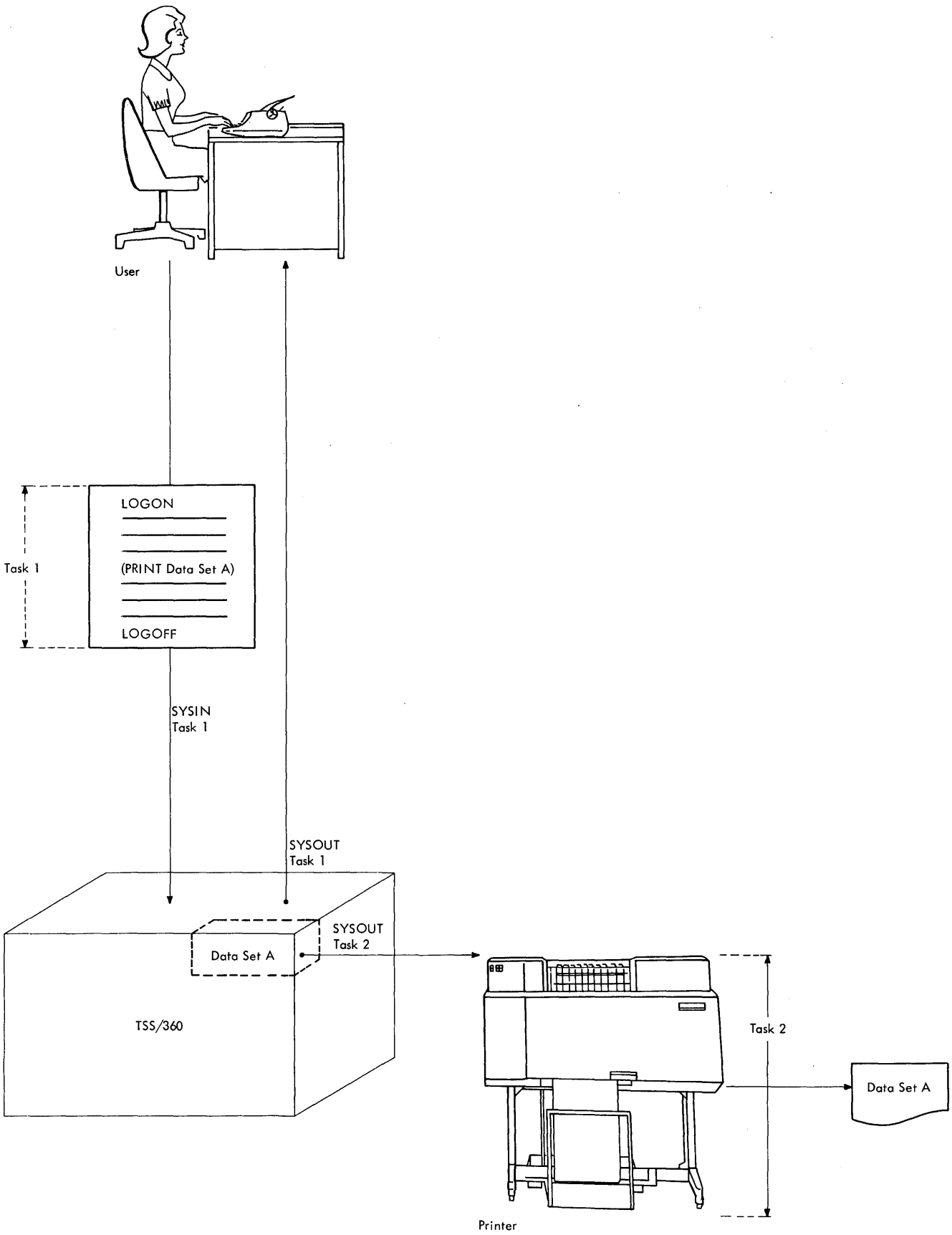


Figure 17. Nonconversational Task Initiated by PRINT Command

Example 1: Initiation of a nonconversational task by the PRINT command. The conversational user has initiated a task (TASK1) by logging on at his terminal (see Figure 17). This task has its own system input and output streams. From his terminal, the user enters a PRINT command, naming the data set to be printed; a separate task is established for this printing operation (TASK2). TASK2 does not have a system input stream, but it has an output stream. Two separate data sets are maintained for this task: one for the printed data set, one for system messages.

Each of the user's two tasks operates independently. The user could initiate another nonconversational task from within his conversational task by including, in his system input stream, another bulk output command (PRINT, PUNCH, or WT) or an EXECUTE command (see Example 2).

For nonconversational tasks initiated by the EXECUTE command, the user must have prestored and cataloged a SYSIN data set that defines what the system is to do. This data set must begin with a LOGON command and end with a LOGOFF command; it may contain any command except EXECUTE or BACK.

Example 2: Initiation of a nonconversational task by the EXECUTE command. The user has initiated a conversational task (TASK1) by logging on at his terminal (see Figure 18). The system input stream contains the user's input commands and data; the output stream contains system messages and other output from the task. As part of his input, the user includes an EXECUTE command, which designates a prestored sequence of commands that is to be executed as a nonconversational task. This sequence begins with a LOGON command and ends with a LOGOFF command; it is executed as TASK2, independently of TASK1. TASK2 has its separate system input and output streams. This system input stream is the prestored command sequence; the output stream, which contains system messages and task output, is printed on the installation's printer.

The user may continue a conversational task in nonconversational mode by issuing a BACK command (refer to "Switching Modes").

At the request of a user, the system operator can also initiate nonconversational tasks by issuing RC or RT commands (refer to "Use of TSS/360 by the System Operator" in Section 4).

Nonconversational Task Execution

Regardless of which method of initiation is used, the nonconversational task gets a batch-sequence number and is executed as soon as the required resources are available. The batch-sequence number is a four-digit decimal number assigned by the system to identify the user's nonconversational task. The user must refer to this number to obtain information about the previously initiated nonconversational task, or to cancel the task (refer to "Nonconversational Task Termination").

During execution of a nonconversational task, there is no communication between the user and the system. The system analyzes, in sequence, each command of the SYSIN data set and, if it is valid, executes it. If a command is invalid, the system terminates the task; otherwise, the system goes on to process the next command.

When execution of a nonconversational task begins, it cannot be interrupted by pressing the user's ATTENTION key, because his terminal is not associated with the task.

Nonconversational Task Termination

Execution of a nonconversational task initiated by EXECUTE, RC (to read in a nonconversational SYSIN data set), or BACK is terminated when:

- A LOGOFF is executed in the task,
- The user issues a CANCEL command for the task,
- The system requires information that is not available in the task's SYSIN data set,
- The system operator issues a CANCEL command for the task, or
- The system operator issues a SHUTDOWN command.

Execution of RT, RC (to read in data), PRINT, PUNCH, and WT tasks is terminated when the data transfer is completed. Also, these tasks are terminated when the user includes, in a conversational task, a CANCEL command that specifies the batch-sequence number of a nonconversational task. A user can cancel only his own nonconversational tasks, and he can issue a CANCEL command only in a conversational task.

A nonconversational task is terminated normally when the LOGOFF command of its SYSIN data set is executed.

If the nonconversational task is waiting to be executed when the CANCEL command is honored, the request for its execution is eliminated from the system. If the task is executing, execution is halted and the task is eliminated. If the task is already completed, the user is so notified.

The system terminates a nonconversational task whenever it finds a situation that requires resolution by the user. Typically, such a situation arises when the system must prompt for an omitted operand in a command or must issue a diagnostic message that demands a user response. A diagnostic message explaining the reason for abnormal termination is printed as part of SYSOUT for the task; all opened data sets are

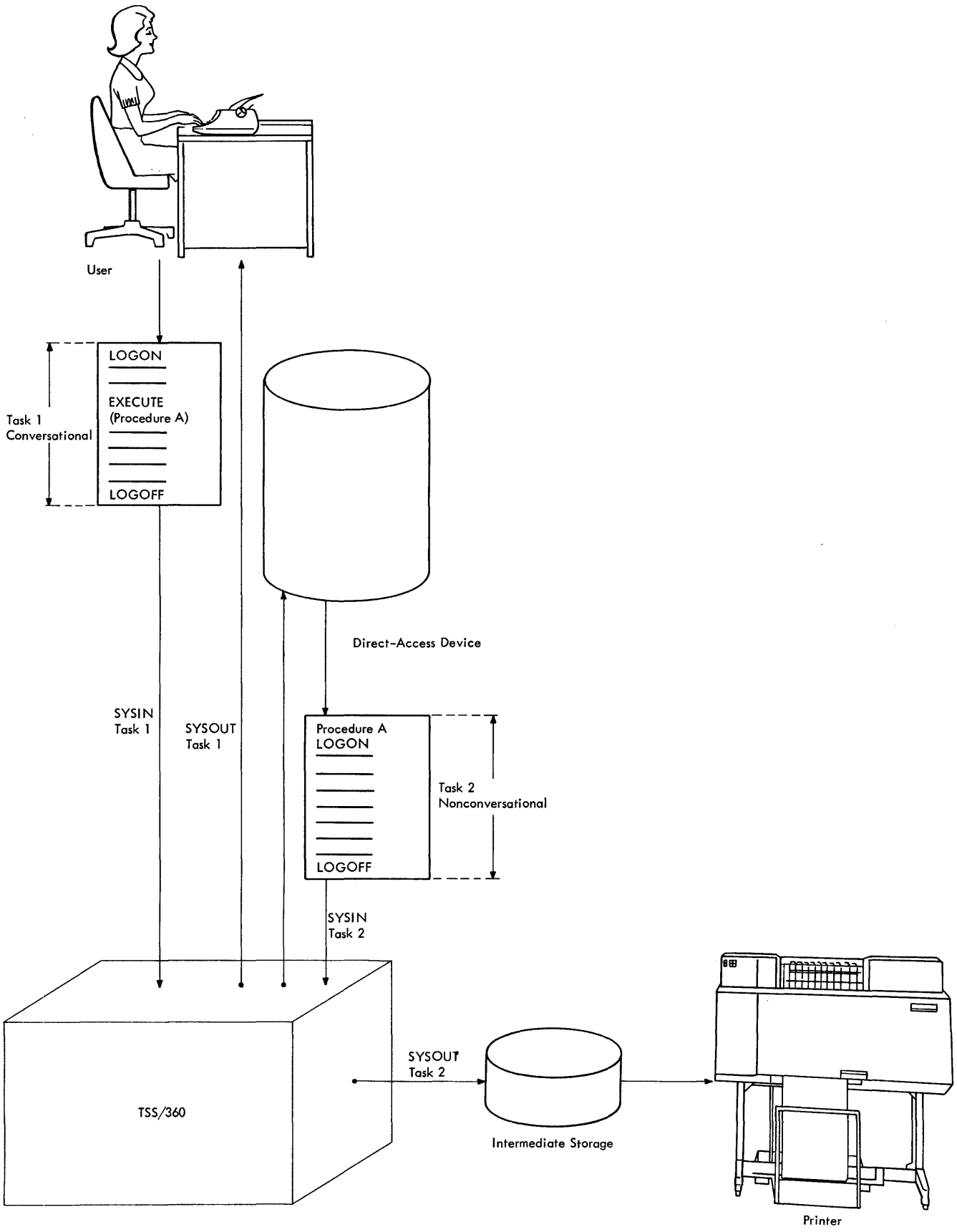


Figure 18. Nonconversational Task Initiated by EXECUTE Command

closed, if possible, and all system interlocks set for the task are removed.

The user designates, by commands in his SYSIN data set, the output expected from his nonconversational task. He must define the data sets that are to be generated, catalog data sets on private volumes if he wants to refer to them in a subsequent task, and indicate their outputs. The only other output from a nonconversational task is a printout of the SYSOUT data set. This includes any messages issued by the system, interspersed in a listing of the commands for the task. All tapes, punched cards, and listings produced by a nonconversational task are available only at the central computer installation.

The user can specify a maximum execution time for his task by issuing a TIME command. If his task has not completed at the end of the time interval, it will be abnormally terminated.

Switching Modes

The user can issue a BACK command to switch a conversational task to the nonconversational mode of operation; he cannot switch a nonconversational task to conversational.

The user can switch the mode of his task if all these conditions exist:

1. He entered a nonconversational SYSIN data set for nonconversational continuation of the task and defined the data set to the system. Unlike the nonconversational SYSIN data set described earlier in this section, this one may be uncataloged and it need not begin with a LOGON command.
2. The user enters a BACK command at his terminal, requesting nonconversational continuation of his task.
3. The system is able to accept another nonconversational task. If not, the user is informed; he may then try to initiate the switch later.
4. All required private devices have been acquired through a SECURE command.

The user does not initiate a separate task when he issues the BACK command (see Figure 19); he still has only one task in the system. This task, however, is nonconversational and has no connection with the user's terminal. After waiting several seconds after issuing the BACK command, the user may log-on at his terminal and create a new conversational task. The system input stream for the nonconversational portion of the task consists of a prestored command sequence designated by the BACK command. The output stream for the nonconversational portion consists of system messages and task output; it is printed on the installation's printer.

If the system accepts the user's request, it establishes the nonconversational task, assigns a batch-sequence number to that task, writes out the batch-sequence number at the user's terminal, and starts task execution.

Tailoring TSS/360

Each user can adapt or "tailor" the system to fit his needs without affecting others' use of the system. Facilities exist for

- renaming commands, operands, expressions, and values
- altering the default values for command operands
- defining command symbols
- creating new commands
- rewriting system messages
- filing changes for permanent use

User Profile

The system maintains a special data set called a user profile containing information pertinent to users. When a user first logs on, the prototype user profile in the system library (SYSLIB) is copied into his virtual storage, where it resides during the task.

The user profile in virtual storage can be altered by the DEFAULT, SET, and SYNONYM commands. If the user wants to preserve his changes beyond the current task, he can issue the PROFILE command to copy his altered user profile from virtual storage into his USERLIB. Since USERLIB is searched before SYSLIB, the system will copy the altered user profile from USERLIB instead of the prototype user profile from SYSLIB into virtual storage for that user's subsequent tasks.

The DEFAULT command allows the user to change system-supplied default values for command operands.

The SET command allows the user to define a symbol that may be referenced or modified by other commands. This symbol is called a command symbol.

The SYNONYM command allows the user to rename commands, operands, expressions and values; the new names created with this command are called synonyms for the original names. The ZLOGON command can be redefined with the SYNONYM command.

Command Creation

The user can create commands to supplement system-supplied commands. A user-written command may consist of a series of system-supplied commands

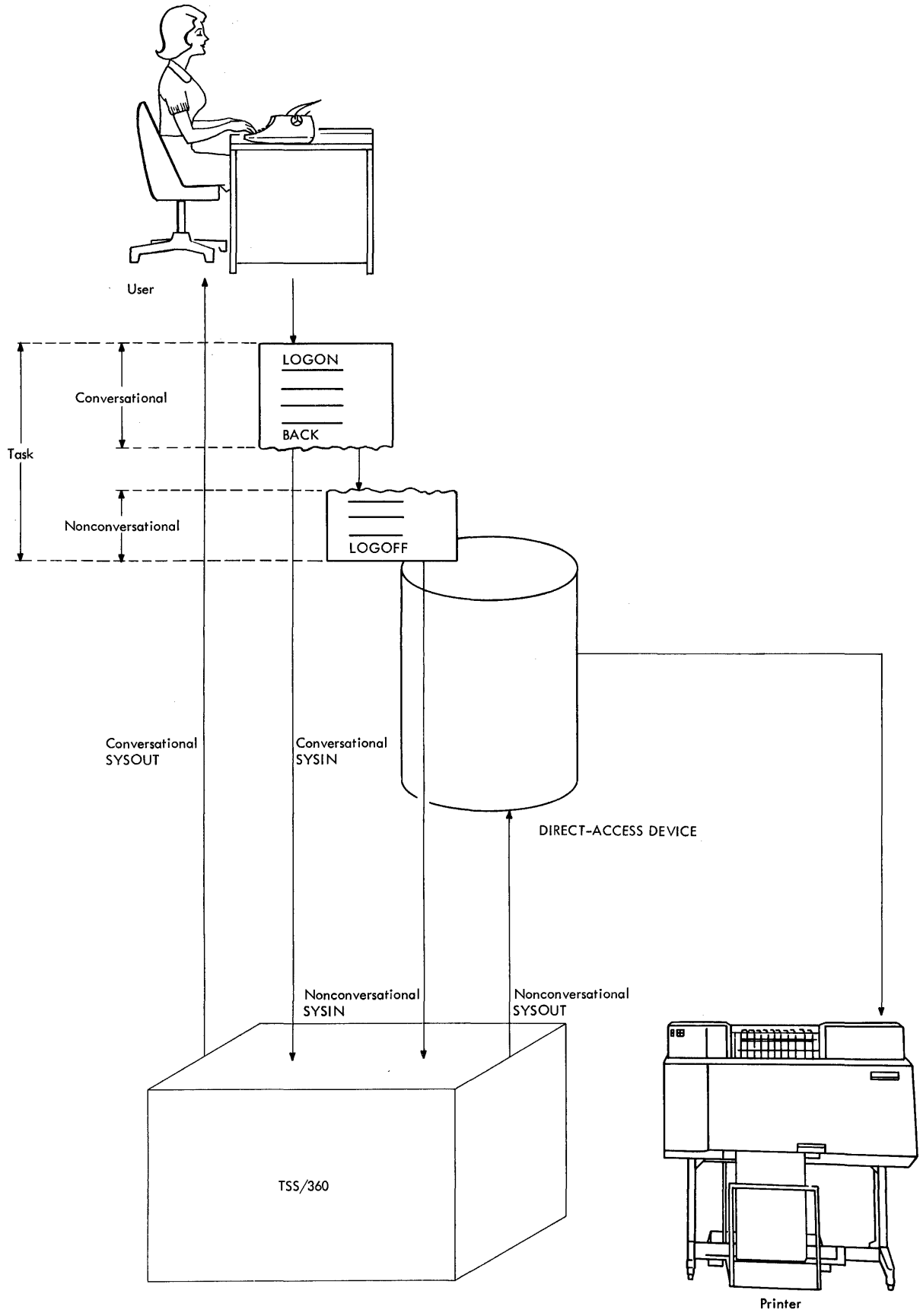


Figure 19. Converting a Conversational Task to Nonconversational Mode Using the BACK Command

(called a command procedure), or it may consist of an object program module that can be invoked as a command.

The PROCDEF command allows the user to define a command procedure. Issuing PROCDEF invokes the text editor; the user can then utilize the facilities of the text editor in entering the lines that will make up his command procedure. (Refer to "Text Editor" in Section 6.) If operands are required for the commands in the command procedure, they can be entered as operands to the user-written command and associated correctly with the command procedure.

This process has advantages over the similar process of storing a series of commands as a nonconversational SYSIN data set and issuing an EXECUTE command against them.

1. A command procedure can be executed conversationally or nonconversationally; the EXECUTE command can be issued only in conversational mode.

2. A command procedure is executed in the same task; EXECUTE initiates a separate nonconversational task.

3. To use the PROCDEF command, it is not necessary to explicitly define a data set in which to store the command procedure.

The BUILTIN command allows the user to define an object program module so that he can invoke it as a command. If the user wants to define operands for a command to be created through the facilities of the BUILTIN command, he must supply the coding within his object program module to handle the parameter values supplied when the module is called (i.e., when the user-written command is issued).

The ZLOGON command can be redefined with the PROCDEF or BUILTIN commands.

Message Handling

The user can write his own messages to replace those issued by the system. User-written messages are filed in a user's message file, which is a virtual index sequential member of his USERLIB. A user-written message has the same message identification code as the system message it replaces. The system searches the user's message file before searching the system message file; if it finds an entry in the user's message file that matches the message identification code it is seeking, it will print out the text of the message from the user's message file.

The user profile contains a filter option to indicate the level of messages the user wants to receive. Messages are classed by severity and length. There are five levels of severity: information, warning, normal error, serious error, and termination error. The system gives the user only warning and error messages unless the filter option is changed. There are four levels of message length: message identification code only, standard messages, extended messages, and reference messages. The system will give the user standard messages unless he changes his filter option.

An explanation of certain messages or of certain words in messages may be requested with the EXPLAIN command. This command can be used to clarify

- the message itself
- designated words in the message
- possible responses the user can make to the message
- the origin of the message

Messages for which the system can supply explanations are indicated by an underscore character printed in the position just before the message text. Specific words for which explanations can be supplied are indicated by an underscore character just before the word. Two underscore characters preceding the text of a message indicate that both the message and the first word of its text have explanations.

Explanatory messages may also have their own explanations, or contain words for which there are explanations. Further explanations can be requested as needed with the EXPLAIN command.

The origin option of the EXPLAIN command is primarily intended to help system programmers find the module that generated a message.

If a message that is being printed out at the terminal is interrupted because the user pushed the ATTENTION key, the user can later cause the message to be repeated in full by issuing the REPEAT command.

Section 6: Data Management

TSS/360 provides comprehensive facilities for systematic and convenient management of data sets. These facilities group naturally into two categories (see Figure 20):

- Data set management
- Problem program I/O

Data set management facilities provide the means for identifying data sets; for efficiently storing and retrieving them within the system; for sharing them with other users; for copying, modifying, and erasing them; and for defining their existence and use in the system.

Problem-program I/O facilities provide for the actual transfer of data to and from problem programs. Some of these facilities are available for storing input data in the system prior to program execution, others are available for data transmission during program execution, and another set is available for transferring program output to conventional output devices during or after program execution.

Data Set Management

Data Set Names

A data set name is a series of one or more simple names, called components, joined so that each represents a level of qualification. For example, the data set name AR.TWO.DESIGN consists of three components that are delimited by periods to indicate a hierarchy of categories. Starting from the left, each component of the name can be considered a category within which the next component is a unique subcategory. This structure for data set names facilitates cataloging data sets, which is a facility that permits a user to store and retrieve data sets on the basis of their names alone. Data sets can be referred to by either fully or partially qualified names.

A *fully qualified data set name* identifies an individual data set and includes all components of that data set's name. For example, data sets 1 and 5 in Figure 21 are uniquely identified by their fully qualified names, A.A and A.C. Similarly, data sets 2 through 4 can be individually identified by their fully qualified names: A.B.A, A.B.B, and A.B.C. Fully qualified names are used in the DDEF command to identify the input and output data sets of a problem program. They are also used in connection with a variety of commands, such as those that manipulate data sets as an entity (e.g., CDS and PRINT), and those that affect system functions relative to the data set (e.g., CATALOG, DELETE, ERASE).

A *partially qualified data set name* identifies a group of data sets, and omits one or more of the rightmost components of a data set name. The group of data sets referred to includes all that have qualifiers identical to those present in the partially qualified name. For example, in Figure 21, data sets 2 through 4 (A.B.A, A.B.B, and A.B.C) may all be referred to by the partially qualified name A.B. Partially qualified names are used in several commands when it is convenient to refer to the specified data sets as a group; e.g., in erasing the group, or in removing it from the catalog, or in specifying that it can be shared by other users.

These basic rules are to be observed by the user in the design of data set names:

1. Each component, or simple name, can consist of from one to eight alphameric characters; the first must always be alphabetic.
2. A period must be used to separate components.
3. The maximum number of characters (including periods) in the data set name is 44. For data sets used exclusively within TSS/360, the user is limited to 35 characters, because the system automatically prefixes each name with his eight-character user identification

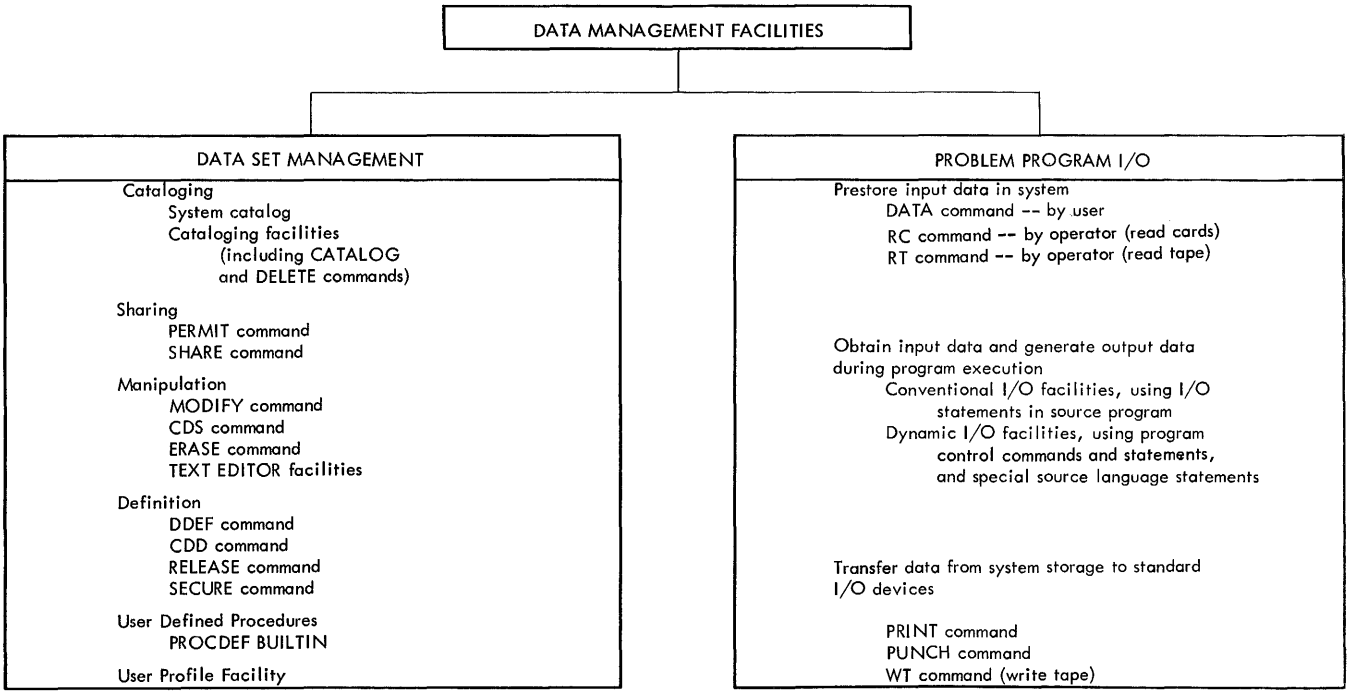


Figure 20. Time Sharing System/360 Data Management Facilities

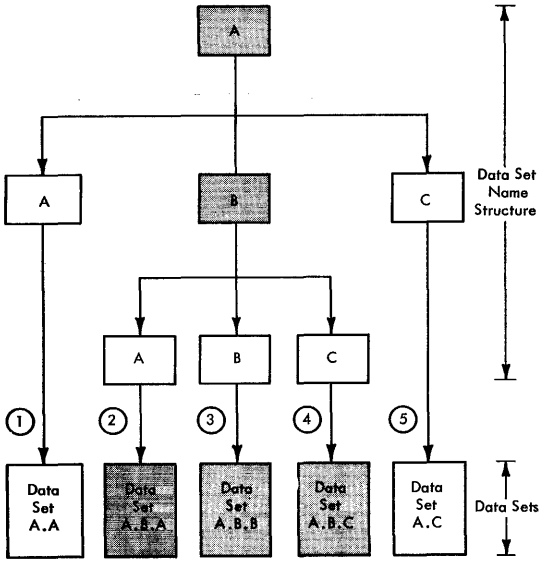


Figure 21. Fully and Partially Qualified Names

followed by a period. For data sets to be interchanged with the IBM System/360 Operating System, the user can employ 44-character data set names. These data sets, however, cannot be cataloged in TSS/360 without being renamed.

4. The maximum number of single-character qualification levels to a single-character basic name is 17, for data sets used in TSS/360. Normally, however, the user will employ only a few qualification levels.

5. The fully qualified data set names in each user's data set name-structure must be unique, and each must uniquely identify one data set.

System Catalog

The system catalog is a special data set that resides on one or more direct-access devices. It is used for filing data set descriptions that must be stored within the system so that, once a data set is created and cataloged, it can subsequently be located by using only its name.

Volume Concept

When a data set is stored in the system, it (1) resides on one or more direct-access or magnetic-tape volumes, and (2) the identification of these volumes is available in the system catalog. A volume can be a removable disk pack, a disk module, or a reel of tape.

At system startup time, the system operator designates each direct-access device and its associated volumes as either public or private. A *public volume* is a direct-access volume that must be mounted prior to the beginning of system operation, and must remain mounted during operation. A public volume can be

used by many users concurrently. A *private volume* can be mounted or dismounted at any time prior to or during operation. Private volumes are restricted to use in one task at a time. Magnetic-tape volumes are always classified as private volumes; direct-access volumes can be either public or private.

Catalog Structure

The system catalog is organized into this hierarchy of indexes (see Figure 22):

- The highest level index, called the *master index,* is a set of user identification codes, one for each user who has been joined to the system. The master index is maintained by the system and updated, based on the JOIN and QUIT commands given by the system administrators and manager.
- The collection of indexes that is subordinate to each user identification code in the master index is called a *user catalog.* The first of these indexes corresponds to the user's identification code. Each of the remaining indexes corresponds to a level of qualification in the data set name structure adopted by the user. Each user, therefore, determines the nature of his catalog by the way he designs his data set naming structure. The system maintains each user's catalog based on the RC, RT, CATALOG, DELETE, ERASE, SHARE, and PERMIT commands. (The effects of these commands on the system catalog are described under "Cataloging and Uncataloging Data Sets" and "Data Set Protection and Sharing.")

When a user's data set is cataloged, the required indexes are established in his user catalog, in accordance with the fully qualified name of the data set (see Figure 22). An index is established for each level of qualification. The master index points to the highest-level index of the user's catalog. This index, and each index thereafter, points to the location of the next lower index. The lowest-level index points to the specific volumes on which the data set is located.

On direct-access volumes, a volume table of contents (VTOC) keeps track of data set locations within each volume.

In the case of magnetic-tape volumes, the lowest-level index of the user's catalog also gives the order or sequence number of that data set on the volume, relative to the beginning of the volume.

Volume and Data Set Labels

All volumes used to store cataloged data sets must con ain standard volume and data set labels to permit the system to locate the data sets.

All public direct-access volumes automatically contain standard volume and data set labels which the

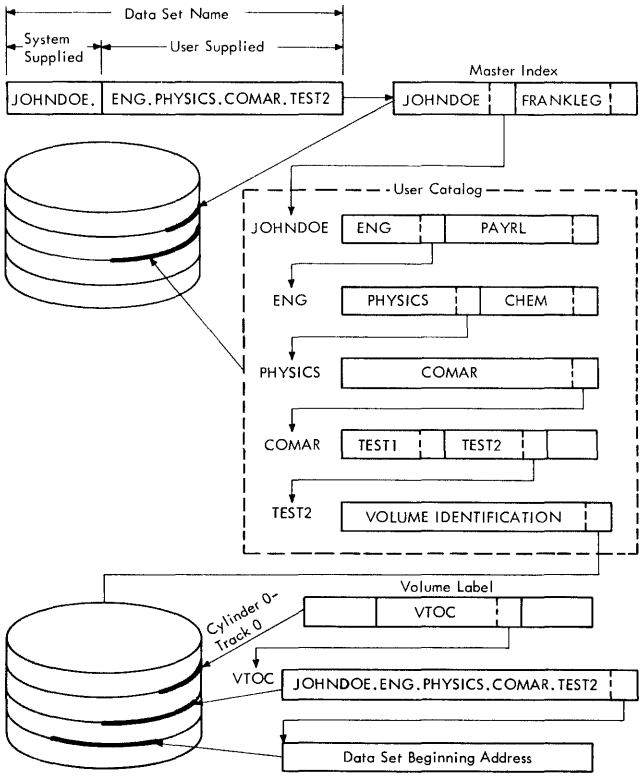


Figure 22. System Catalog Concept

system creates and maintains. Direct-access volumes can also contain user's data set labels, which are processed by user-written routines.

Magnetic-tape volumes may contain (1) standard volume and data set labels, or (2) standard volume and data set labels *plus* user's data set labels, or (3) they may be unlabeled. All labels, on magnetic-tape volumes with standard labels, are also created and maintained by the system; user's data set labels are processed by user-written routines.

Detailed explanations of standard labels and their use on direct-access and magnetic-tape volumes are given in *Assembler Programmer's Guide.*

Generation Data Groups

A generation data group is a collection of successive, historically related data sets, such as similar payroll data sets that are created every week. Cataloging such data sets with unique data set names would cause inconveniences that can be avoided. The system provides an option, with the cataloging facility, that assigns numbers to individual data sets in a chronological collection, thereby allowing the user to catalog the entire collection under a single name. The user can distinguish among successive data sets in the collection without assigning a new name to each data set. Because each new data set is normally created from

the one created on the previous run, the new data set is called a generation, and the number associated with it is called a generation number. The user can refer to a particular generation by specifying, with the common name of the group, either the relative generation number or the absolute generation name of the data set.

Relative Generation Numbers

At any time, the relative generation number of the most recently cataloged data set in any generation data group is 0. The relative generation numbers of previously cataloged data sets in the group are negative integers that indicate relationships to the latest cataloged generation. New data sets for the group are created by using positive integers as relative generation numbers.

Example: Payroll data sets consist of a generation data group named PAYROLL. The user can refer to the most recent generation as PAYROLL(0) and to immediately preceding generations as PAYROLL(-1), PAYROLL(-2), etc. A new generation would then be referred to as PAYROLL(+1). After this new generation is cataloged, it automatically becomes PAYROLL(0), and the old PAYROLL (0) is referred to as PAYROLL (-1). Thus, adding a generation changes the relative generation numbers of all data sets in the group.

Relative generation numbers depend upon the physical position of the generation name in the index. If a name is dropped from the index, the contents of the index are shifted.

Absolute Generation Names

To each data set in a generation data group, the system assigns an absolute generation name of the form GxxxxVyy, where xxxx, an unsigned four-digit decimal generation number, and yy, an unsigned two-digit decimal number, indicate the version of a particular generation. Appending the absolute generation name to the name of the generation data group provides a unique name for the data set. For example, if 0001 is the generation number initially specified for generation data group A.PAYROLL, the user can refer to the first generation as A.PAYROLL.G0001V00 and to the next generation as A.PAYROLL.G0002V00.

The system is requested, by the DDEF command, to create a data set as a new generation of a generation data group. The system develops the generation and version numbers as follows: The generation number is obtained by adding the positive relative generation number specified in the DDEF command to the previous generation number; the version number is set to 0. Thus, if the present generation is G1384V03 and the incrementing factor specified is the relative generation number (+2), the new generation is G1386V00. The system does not automatically create nonzero version numbers; if the user wants to replace an existing generation and change the version number, he must issue a CATALOG command specifying this change (refer below to "Cataloging and Uncataloging Data Sets"). The system automatically changes the catalog entry for the named generation. Thus, if a new data set that replaces the generation named G1386V00 is to be cataloged, it may be named G1386V01, which replaces G1386V00 in the generation data group index when the system changes the catalog entry. The data set that is to be replaced will be erased automatically if it resides on public storage, but not if it resides on private storage. In either case, the system sends a message to SYSOUT indicating the generation name, its location, and its disposition.

Each data set in a generation data group has a unique fully qualified name. This name consists of the group name, a period, and the low-order qualifier GxxxxVyy. This leaves a total of 25 characters available for higher-level qualification of the group name.

Cataloging and Uncataloging Data Sets

Data sets may be cataloged and uncataloged in several ways. *Example:* In the RC and RT operations (initiated by the operator), an input data set is always cataloged; an input data set entered by the DATA command is cataloged by the system if the data set resides on public storage. If the input data set resides on private storage, it is cataloged only if the user issues a CATALOG command for the data set after giving the DATA command.

All VSAM, VISAM, and VPAM data sets that reside on public storage are cataloged by the system when they are defined. Hence, there is no need to issue a CATALOG command for these data sets.

All output data sets produced by the TV (tape-to-VAM) and VV (VAM-to-VAM) commands are cataloged by the system. All output data sets, except those with a non-TSS/360 data set name, produced by the VT (VAM-to-tape) command are cataloged by the system.

The CATALOG command may be used to catalog a data set that resides on private storage, or to alter the entry of a previously cataloged data set (e.g., to change the catalog index structure for a renamed data set, or to change the version number of a generation data group member).

The CATALOG command is also used to:

- Structure the catalog for an entire generation data group. The user can indicate the number of genera-

tions to be retained, as well as the disposition of old generations when the specified number of retentions is exceeded.

- Catalog data set as a new generation of an existing generation data group if the data set resides on private storage.

NOTE: Catalog control of the generations of a generation data group can be exercised only by the owner of the generation data group (refer below to "Data Set Protection and Sharing").

The DELETE command allows the user to remove the catalog entry for a data set. It should be used only when (1) the data set resides on a private volume (i.e., a magnetic-tape volume or a private direct-access volume), and the user wishes to remove it from the catalog but not erase it; (2) the user's catalog contains the names of data sets the user is sharing (but does not own) and he wishes to remove them from his catalog, as he no longer wishes to share those data sets (refer below to "Data Set Protection and Sharing"). In all other cases, the ERASE command must be used. A data set with bulk I/O pending may not be deleted (DELETE command).

The ERASE command removes the catalog entry for a data set and also erases the data set if it resides on direct-access storage; i.e., releases its storage space for other use.

When the user logs-off in conversational mode, he is presented with the name of each uncataloged data set he used during his time on the terminal; he is asked to indicate whether he wants those data sets cataloged. If he requests cataloging, the system makes the appropriate entries in the catalog.

Assembler-language programmers can also use the CAT, DEL, ERASE macro instructions to perform the same functions as the CATALOG, DELETE, and ERASE commands.

Data Set Organizations

A data set's *organization* defines the overall relationships of the component records into which the data set is logically subdivided. The component records are called *logical records,* because each is a logical entity containing information for the problem program that is to process the data set.

In TSS/360, two fundamentally different types of data set organizations can be processed: virtual storage data sets, and physical sequential data sets (organized for interchange with the IBM System/360 Operating System). Both types of data sets may not be stored on the same volume.

Virtual Storage Data Sets

These data sets are specifically organized for efficient processing within TSS/360. Data sets with a virtual organization reside only on direct-access volumes. Users create, read, and process these data sets on the basis of the logical records they contain. The system, however, organizes these data sets by pages (4096 bytes) and uses the page as the unit of transfer between the direct-access device and the user's virtual storage. The system also ensures that only the required pages of a data set are in virtual storage.

Virtual storage data sets may have any of these organizations:

1. Virtual sequential
2. Virtual index sequential
3. Virtual partitioned

1. In a *virtual sequential* data set, the order of the logical records is determined solely by the order in which the records were created. With this type of data set, the user provides the system with a stream of logical records. The system concatenates the records, organizes the data into pages, and stores the data set (page by page) on a direct-access device. After each record is stored, the system makes its retrieval address available to the user's program. Users employing assembler language can form another virtual sequential or virtual index sequential data set containing these retrieval addresses; in effect they create their own direct-access criteria. After the data set has been created, if the user wishes to make an orderly sweep through it, the records can be read back in the order in which they were created merely by requesting one logical record after the other. An assembler user can also read and update logical records nonsequentially by providing the required retrieval address of each record involved, using addresses supplied by the data sets in which he has defined his direct-access criteria.

2. A *virtual index sequential* data set's logical records are organized into an ascending collating sequence, based on a data key associated with each record. The data key may be a control field that is part of the record (such as a part number), or it may be an arbitrary identifier (such as a line number) that is the beginning of each logical record, and is added to each record to give it a unique key.

One special form of virtual index sequential data set is the line data set, with a maximum of 132 bytes per record, which has the line number of each record as the key. The line number is a seven-digit decimal number at the beginning of each record.

Another special form of the virtual index sequential data set is the region data set. Records in a region data set are indexed by two fields. Region names arranged alphabetically divide the region data set into

regions; line numbers index the lines within each region.

Line numbers in a region data set are seven-digit decimal numbers, like those in line data sets, but they follow the region names rather than being in the first position in each record. Region names vary in size to a maximum of 247 characters. Records in a region data set can be a maximum of 256 bytes long. Thus the region name can occupy 247 bytes of the longest possible records.

However, the region name need not be a field arbitrarily added to each record; it may be an integral part of the record, such as a name field.

The text editor, discussed later in this section, creates and modifies line and region data sets. It treats line data sets as region data sets with region lengths of zero. (See "Text Editor.")

In addition to the logical records, virtual index sequential data sets contain a page directory and locators that relate the keys and physical addresses of the records in the data set (see Figure 23).

The page directory is set up when the data set is created. It gives the value of the key for the first record in each data page. The directory can consist of one or more pages, depending upon the size of the data set.

In each page of the data set there is an ordered set of locators, one locator per record. Each locator specifies either the physical location of the record in the page, *or* the position of a corresponding locator in an overflow page. Overflow pages are provided automatically by the system when it is necessary to logically insert a record between two existing records after the data set has been created, *and* there is not room to place it on the page on which it belongs. The record is thus available in proper logical sequence even though it is not physically located where it appears to be located.

The user can optionally specify in the DDEF command's DCB operand (or, assembler users can make that specification in the DCB macro instruction) that a certain percentage of space be left in each page for expansion of the data set. This minimizes the number of overflow pages and shortens the storage and retrieval time for records inserted between existing records.

Because records in the virtual index sequential organization have logical and physical relationships, the user can request the system to perform any or all of these operations:

- Retrieve or create (in a manner similar to that for sequential organization) logical records whose keys are in ascending collating sequence.

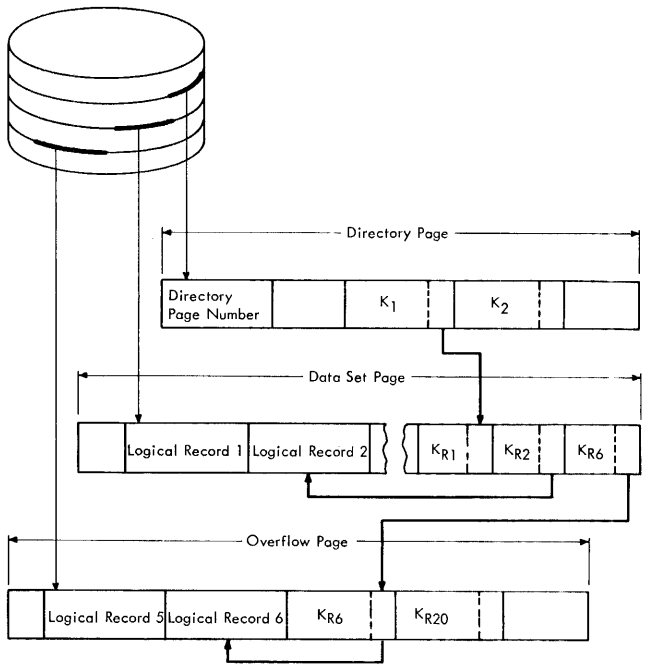


Figure 23. Typical Virtual Index Sequential Data Set

- Retrieve or create individual records whose keys are in any order. (Processing is, of course, slower here than if it were being done in the collating sequence, because a search is required to locate each record's position.)
- Add records, with new keys, to the data set. The system automatically locates the proper position in the data set for the new control and makes all necessary adjustments for subsequent retrieval in logical sequence.
- Delete existing records from the data set. The system automatically updates the page locators (and the page directory if necessary) and makes the space used by the deleted records available for other uses.
- Update existing records in the data set, either expanding or contracting their size.

3. A *virtual partitioned* data set is used to combine individually organized data groups into a single data set. Each group of data is called a member, and each member is identified by a unique name. The member name may consist of from one to eight alphameric characters; the first character must be alphabetic. The partitioned organization allows the user to refer to either the entire data set (via the partitioned data set's fully qualified name) or to any member of that data set (via a name consisting of the fully qualified name of the data set suffixed by the member name in parentheses).

Example: A partitioned data set named MATHLIB, whose members consist of mathematical subroutines such as SQRT, ARCTAN, and COS, could be referred to on any of these bases:

MATHLIB	Entire library of subroutines
MATHLIB(SQRT)	Square-root subroutine
MATHLIB(ARCT)	Arc-tangent subroutine
MATHLIB(COS)	Cosine subroutine

References to individual members are possible because when a partitioned data set is created, a directory is set up to keep track of each member. As members are added, deleted, or changed, the directory information (member location, size, attributes, etc.) is automatically updated.

The partitioned data set may be composed of virtual sequential or virtual index sequential members or a mixture of both. Individual members, however, cannot be of mixed organization.

The organization of a virtual partitioned data set is shown in Figure 24.

The first entry in a data set is the partitioned organization directory, which is used to locate the members of the data set. Each member begins on a new page; any space remaining on the previous page is unused (see Figure 24).

Users can assign additional names, called aliases, to each member, and subsequently locate a member on the basis of either the member name or any of its aliases. The partitioned data set organization is ideally suited for storage of libraries of programs or other groups of data that are frequently referred to together.

Physical Sequential Data Sets

Physical sequential data sets, organized for processing by the IBM System/360 Operating System, can also be processed by TSS/360 when interchange is required.

The logical records in these data sets are organized solely on the basis of their position relative to the beginning of the data set. When these records are processed in TSS/360, a block of one or more logical records is the unit of transfer to and from the device involved.

Record Formats

The data management routines of TSS/360 require format specification (and in some cases other information) about the logical records in each data set. Three types of record format can be specified by the user:

- *Format F* (fixed length) is specified for data sets whose logical records all contain exactly the same number of bytes.
- *Format V* (variable length) is specified for data sets containing logical records that vary in length,

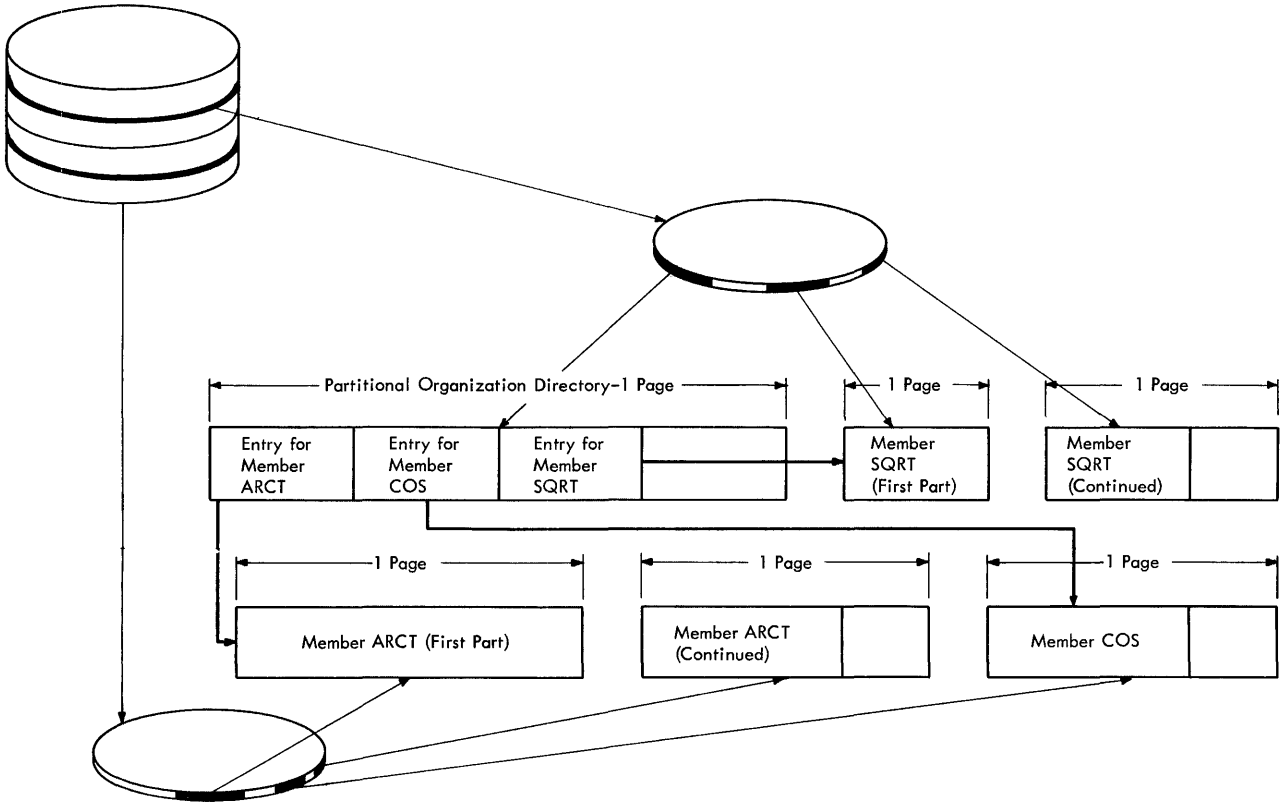


Figure 24. Virtual Partitioned Data Set

and each of which contains user-supplied information that gives the length of the record according to system-defined conventions.

- *Format U* (undefined format) is specified if the logical records in a data set are neither format F nor format V.

Details on the specification of record format and other record-oriented information (such as physical attributes) for each type of data set organization are described in *Assembler Programmer's Guide* and *FORTRAN Programmer's Guide.*

Basic Concepts in Device Management

Use of Public and Private Volumes

Data sets that are to be cataloged can be stored on either public or private volumes. Data sets used within a task need not be cataloged if their use is temporary (i.e., confined to one task) and they reside on private storage.

The system assumes that a user desires storage on a public volume unless he specifically asks for storage on a private volume.

Users will make the most effective use of TSS/360 by storing their data sets on public volumes when it is necessary to retain the data sets in the system. Public volumes are always mounted and available for allocation to a user's task.

If a user employs private volumes, he may need to wait for devices on which to mount those volumes. Each time a request is made for a device on which to mount a private volume, the system must determine whether or not it can honor the request, based on the current requirements throughout the system for that device. If the system cannot allocate a private device to a user's task, one of two actions occurs, depending upon the operational mode:

- For a conversational task, the system issues a message to the user during the execution of the DDEF command, if it cannot allocate the required space or if the required volumes cannot be mounted.
- For a nonconversational task, the system will place the task in abeyance until the operating situation permits the allocation of all required private devices (refer to "SECURE Command," in this section).

Use of Unit Record Devices

A key concept in efficient device management in TSS/360 relates to the proper use of conventional I/O unit record equipment (printers, card readers, and punches). These devices cannot be referred to during program execution, unless an installation elects, at system generation time, to incorporate its own special option that will make these devices available to problem programs. However, they can be effectively used where necessary prior to, and after, program execution.

Planning Data Flow

Three fundamental rules should be observed, wherever possible, in planning data flow through TSS/360.

1. A problem program's input data can be made available to it most efficiently, at execution time, as either terminal input or as a cataloged data set residing on a public volume.

   a. Input data can be entered from the terminal as dynamic input defined by program-control commands and statements, or by the facilities provided for problem-program communication with SYSIN.

   - In conversational mode, both facilities can be applied effectively for terminal input to programs with small input requirements or with input requirements that depend on intermediate results achieved in the program.
   - In nonconversational mode, both facilities can be similarly applied for limited dynamic input provided this input is properly arranged in the SYSIN date set; However, *conditional* dynamic input is not possible; that is, the input must be arranged as required in the SYSIN data set *before* the task processes it.

   b. Large volumes of input data can best be supplied as a cataloged data set residing on a public volume. Having the data sets cataloged simplifies their definition to the system; placing them on a public volume makes their availability certain at execution time.

   - If the data set involved is the output of a previous problem program, the data set should be created on a public volume.
   - If the data set to be processed is new, it should be prestored on a public volume. Small data sets can be prestored in this way, from the terminal, by using the text-editor or data-editing commands. (See "Text Editor" and "Data Editing" later in this section.) Large data sets can be entered into the system by nonconversational tasks, which the user asks the system operator to initiate, using the RC and RT commands. (During the execution of these commands, the data set is automatically stored on a public volume.)

2. Problem program output can be most efficiently handled during program execution as either terminal output or as a data set on public storage.

a. Terminal output data can be either dynamic output defined by program-control commands and statements, or by facilities provided for problem program communication with SYSOUT.

- In conversational mode, both facilities can be effectively used to display program output at the terminal, for programs with small output requirements, or to notify the user of selected intermediate results that he can then analyze to determine what he wants to do next (such as dynamically input additional data, change sequence of program, run another object module).
- In nonconversational mode, both facilities can be similarly applied to transfer dynamic output to the SYSOUT or PCSOUT data set.

b. If an output data set resides on private storage, the required volume will be mounted when it is needed.

NOTE: If the user elects not to use dynamic output, as described above, he can still selectively inspect his output line data sets (e.g., display them at his terminal) by using the LINE? command.

3. Nonconversational tasks should be initiated using the PRINT, PUNCH, and WT commands to transfer output data sets from their resident public volumes to the system's output media.

a. PRINT and PUNCH can be used to transfer data sets to the printer and card punch.

b. WT can be used to record a data set on magnetic tape in a format suitable for later printing. If WT is used, the resulting data set can be cataloged (using a catalog option in the WT command) to facilitate later printing, either by a PRINT command or as an off-line operation.

NOTE: The PRINT command transfers data sets with virtual sequential, virtual index sequential, or physical sequential organizations. The PUNCH and WT commands transfer only data sets with virtual sequential and virtual index sequential organizations.

Data Set Protection and Sharing

A user cannot gain access to any data sets other than his own unless he has system authorization to do so, or he has been given authorization by another user who owns the data sets involved. In TSS/360, cataloged data sets may be shared or unshared.

A data set is unshared unless its owner (original author) issues PERMIT commands allowing it to be shared. If it is cataloged, it is under the owner's user identification and is accessible to that user only.

A *shared* data set is cataloged and belongs to one user, but may be shared with other users on any of these bases:

1. *Read-only access:* The sharer may read the data set, but may not change it in any way.
2. *Read-and-write access:* The sharer can both read and write to the data set, but he may not erase it.
3. *Unlimited access:* The sharer, in effect, can treat the data set as his own; he may even erase it.

The data set owner issues a PERMIT command to designate the other users who may share his data sets and to indicate the type of access those users may have. The PERMIT command also allows the data set owner to change any access authorization that he previously gave. A separate PERMIT command is required *for each level of access to a data set,* but any number of sharers may be authorized for the same level of access with a single PERMIT command. Each time a PERMIT command is issued, the owner's catalog is updated; information on who may share which data sets, and to what level of access they may share, is stored in his catalog.

To gain access to a data set for which he has been previously authorized, the sharer must issue a SHARE command. To see how this command is used, assume that the sharer's user identification is JMC200 and that he has been permitted to share one data set. The data set is owned by user RKP100, and is cataloged by him under the fully qualified name ENG.PHYSICS.COMAR.TEST2. Assume also that the sharer wants to name the data set ENG.CHEM.NOTAR.TEST1. He would then issue the SHARE command shown at the top of Figure 25. In response to that command, the system would search the owner's catalog to see if the prospective sharer is authorized. If he is not, the command is ignored; if he is authorized, the system places in the sharer's catalog a pointer to the *owner's (complete) name for the data set.* This is a sharing descriptor that bears the name by which the sharer refers to the data set. Whenever the *sharer* subsequently refers to the data set by *his* name, the system locates the data set by the search procedure shown on Figure 25.

NOTE: The name assigned to a data set by its owner is not affected in any way by other users who assign their own names to that data set. Sharers may use the same name as the owner because user identifications are unique in the system.

A sharer's catalog entry is not removed if the owner erases or uncatalogs a shared data set. Each sharer must use the DELETE command to update his own catalog (i.e., to get rid of sharing descriptor entries).

The PERMIT command can also be used by an owner to withdraw or restrict previous authorization to use the owner's data sets.

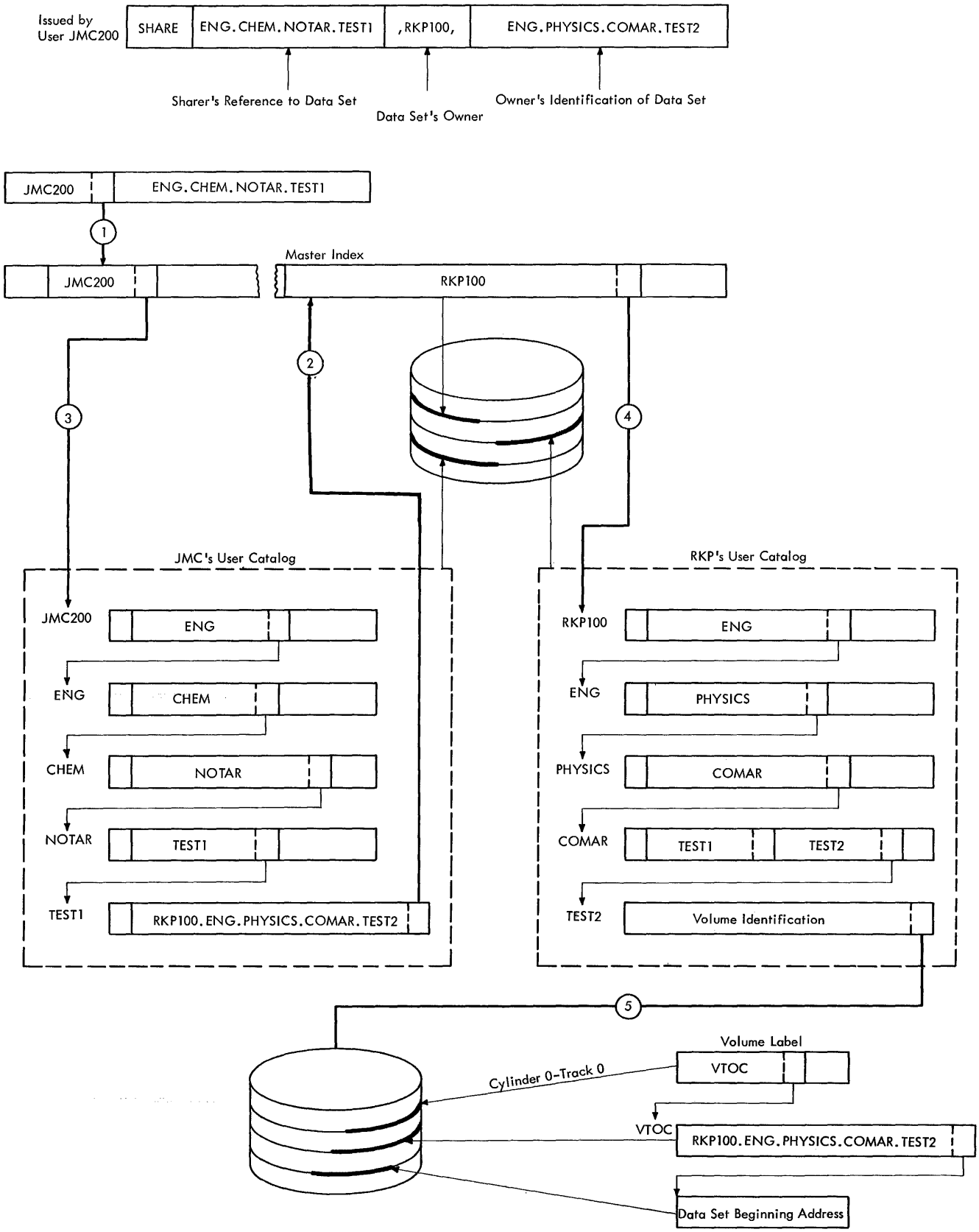


Figure 25. Sharing of Cataloged Data Sets

If the owner allows another user to share all his data sets, the sharer can refer to them as a group in the SHARE command by specifying his name for the collection and then specify ALL. In this case, the system places a pointer to the owner's user identification in the sharer's catalog, thereby making all of the owner's catalog available to the sharer. Similarly, groups of data sets with names having common higher-order components can be specified by using partially qualified names for the owner's catalog.

To be concurrently accessible by more than one task, a data set must have one of these organizations:

- Virtual sequential
- Virtual index sequential
- Virtual partitioned

To prevent several users from updating the same record at the same time, interlocks are put on the data set while it is being used. The two types of interlocks are: read and write.

A read interlock is imposed to prevent other users from writing into a data set or page of a data set. Multiple read interlocks may be established for a data set, permitting several users to read it simultaneously, or the interlocks may be set on a page basis, giving several users simultaneous access to the records within a page. A read interlock cannot be used if a write interlock has already been used for a data set or a page.

A write interlock prevents any user, other than the user who set the interlock, from reading or writing into a data set or page. Only one write interlock can be set at a time; when a write interlock is set, neither read nor write interlocks can be applied until the write interlock is reset.

Data Set Definition

Before a problem program or a command can process a data set, the system requires complete information about the data set, including the manner in which it is to be processed. The user can make this information available from a variety of sources; for example:

DEFINITION OF PROBLEM PROGRAM I/O DATA SETS	DEFINITION OF DATA SETS PROCESSED BY COMMANDS
1. I/O source statements	1. DDEF commands
2. DDEF commands or macro instructions	2. System catalog
3. System catalog	3. Command itself
4. Problem program	
5. Data set label	

The following paragraphs describe how to use these sources to identify data sets for the system.

Data Control Block

The information required to identify a data set to be processed by a problem program is contained in the data set's data control block (DCB), a group of contiguous fields in the user's program. The DCB contains these types of information:

- The DDNAME operand of the DDEF command to be associated with the data set
- Type of data set organization
- Record-format information (format type, record length, etc.)
- Device-dependent options
- Exit addresses:

  SYNAD: synchronous error exit address, for automatically transferring control to a user-supplied routine if an uncorrectable I/O error occurs.

  EODAD: end of data set address, for automatically transferring control to an end-of-data routine when end of an input data set is detected during processing.

  EXLST: address of an exit list in which the user, in the case of sequential data sets intended for interchange with the IBM System/360 Operating System, can define the address of routines for creating and verifying the user data set labels that can be employed on magnetic-tape and direct-access volumes; or the address of a routine to be used at OPEN time for modifying the data control block.
- Working storage used by the access method routines.

The user requests the system to begin construction of a data control block at assembly or compilation time.

There are various ways in which the fields in a data control block may be filled in. For example, some may be filled in at assembly/compilation time. Others may be filled in during program execution from user- or system-supplied information. In any event, the fields are filled in according to a fixed priority scheme based on the source supplying the information. The sources of information and their priorities are:

FORTRAN USERS	ASSEMBLER USERS
1. DDEF command (and system catalog)	1. User's program
2. Data set label	2. DCB macro instruction
3. FORTRAN compiler	3. DDEF command (and system catalog)
	4. Data set label

Not every source is valid for every field. These two general rules apply: (1) When a field has been filled by a higher priority source, it cannot be replaced by information from a lower priority source. (2) A field that has not been specified by a higher priority

source, may be filled in by a lower priority source if that source is valid for that field.

1. An assembler user can include one or more routines in his program, to add to or modify the contents of a data control block. Generally, these routines can be called at any time during execution. The restrictions on the use of a problem program to modify a data control block are described under the DCB macro instruction in *Assembler User Macro Instructions.*

2. The DCB macro instruction can be used to fill in any, or all, fields at assembly; however, once a field is specified in this way, it can be changed only by the user's problem program. In the case of FORTRAN users, data control block fields provided by the compiler can be overridden by any other source.

3. At OPEN time, information from the DCB operand of a DDEF command can be, and frequently is, used to complete the data control block. Any field that is empty at OPEN time, and for which the DDEF is a valid source, can be filled by DDEF information. If the data set to be processed is an input data set that was previously cataloged, its DDEF command will indicate this and the system will retrieve certain data control block information (e.g., the data set's location) from the system catalog.

4. Also, at OPEN time a field of the data control block for an existing data set can be filled with information from the data set's label. This field must not have been specified by any other source; the data set label must be a valid source for that field.

This procedure for data control block definition and modification can be greatly simplified for most applications. The flexibility is provided for special cases where data control block changes must be made between assembly time and the time a data set is actually processed. In these situations, the facility to modify allows the user to change only the required fields; he does not have to restate the entire data control block each time a program is run. To facilitate data control block modification, the user should include in the data control block only those fields needed for program execution — others should be left empty for possible subsequent fill-in. Once the data set is closed, the DCB is restored to its pre-OPEN state. When the data set is opened again, the system starts the fill-in procedure based on the data control block information provided by the DCB macro instruction and any problem program modification to the DCB since the last CLOSE.

Identification of FORTRAN Data Sets

Data sets to be processed in FORTRAN-written programs are identified for the compiler by a data set reference number in a FORTRAN READ or WRITE statement. *Data set reference numbers* are of the form xx, where x is any number 0-9. When the system is generated, an upper limit for the data set reference numbers is established.

During compilation, the FORTRAN compiler automatically generates the appropriate data control block for each data set reference number appearing in the source program. The information in this control block, plus information supplied from the sources (e.g., the DDEF command, system catalog, data set label), is used by the system at program execution time to identify the data set and to provide the resources (routines, devices, etc.) necessary to process it.

The basic method used to identify FORTRAN data sets is illustrated in Figure 26.

The system first relates the data set reference number of the READ or WRITE statement to the DDNAME operand of the corresponding DDEF command. For FORTRAN data sets, the DDNAME operand of the DDEF command is always of the form FTXXFyyy, where XX is the data set reference number, and yyy is a FORTRAN sequence number used to differentiate data sets (e.g., on the same device type with the same data reference number). Having found the corresponding DDEF command, the system then obtains the name of the data set from the DSNAME operand of that DDEF command. Other information in the data control block and DDEF command is then used to determine such things as:

- data set residence (i.e., where an input data set is located, or where an output data set is to be placed),
- organization of the data set, and
- routines necessary to process the data set.

NOTE: If a FORTRAN I/O statement, such as PUNCH or PRINT, does not contain a data set reference number, or if the data set reference number used in a READ or WRITE statement refers to a logical unit for which no DDEF command has been issued, the system assumes that the SYSIN or SYSOUT data set is involved.

Identification of Assembler Data Sets

A data set, to be processed by a problem program written in assembler language, must be identified by a DCB macro instruction in the source program. The assembler uses the DCB macro instruction to set up the data control block at assembly time, and, if the user has supplied operands in the DCB macro instruction, to enter those operands into the data control block. The number of operands that can be specified in the DCB macro instruction depends upon the organization of the associated data set.

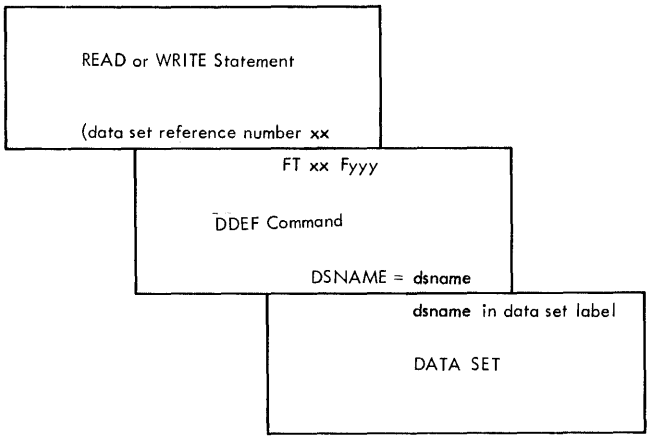


Figure 26. Data Set Identification, FORTRAN-written Programs

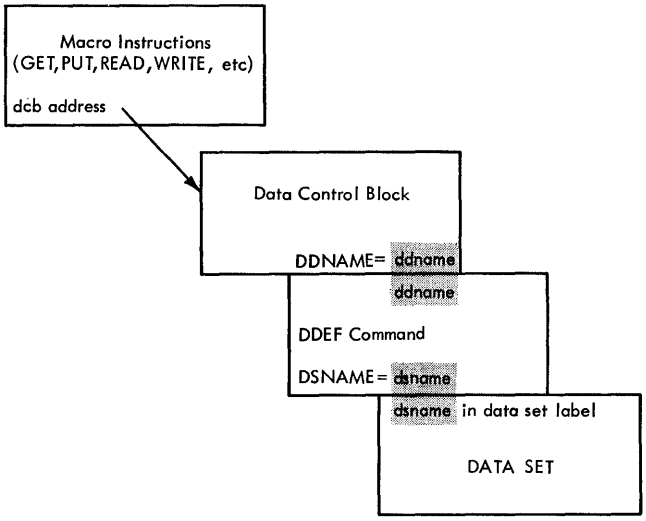


Figure 27. Data Set Identification, Assembler Language Program

As in the case of a FORTRAN data set, the information in the data control block, plus the information supplied from other sources, is used by the system at program execution time to identify a data set and to provide the resources (routines, devices, etc.) necessary to process the data set.

The symbolic chain that relates the macro instructions used for data retrieval and storage (GET, PUT, READ, WRITE, etc.) to their associated data sets is shown in Figure 27. It also illustrates how information in the data control block and DDEF command identify assembler data sets to the system.

DDEF Command

The DDEF command is used to identify a data set during execution of a task and to define its requirements for system resources. It may also be used to define a job library, to define a special data set for the DUMP program-control command, to complete the data control block of a program at execution time, and to concatenate input data sets (i.e., relate them so that several different data sets can be read in as if they were one).

A DDEF command issued during a task is valid throughout the task, unless the user enters a RELEASE command for that data set. The RELEASE command is the opposite of the DDEF command: the DDEF command sets up task control information for the data set; the RELEASE command removes that information.

The DDEF commands used in a task need not be issued directly during the task. One or more, or all, of the DDEF commands needed can be made available by using the CDD command (discussed later in this section).

If the DDEF command defines a virtual-storage data set that resides on public storage, the system catalogs the data set when it is defined.

In a conversational task, the system analyzes the data set's requirements at the time the DDEF command is issued. It will then attempt to allocate the required resources (and, for private volumes, issue any mounting messages that are required) at that time. If the required space cannot be allocated, or the specified volumes cannot be mounted, the system will inform the user of this immediately, thereby allowing him to proceed with other work. (However, if the user follows the basic rules stated under "Planning Data Flow," the resources he requires will always be available.)

Nonconversational tasks remain queued until the system is able to fill the requirements for private devices. This list of requirements is made available to the system by means of a SECURE command, which the user must include in the task's nonconversational SYSIN data set as the first command after LOGON. Then as each DDEF is read and processed, the required devices are allocated from those that have been reserved for the nonconversational task. Any attempt to allocate more devices than are available will cause the task to be terminated.

Definition of Problem Program Data Sets

The operands of the DDEF command allow the user to specify:

- Name of the definition (DDNAME)
- Organization of the data set

- Name of the data set
- Information for the data set's data control block
- Unit and volume residence requirements
- Storage requirements
- Label information
- Status of the data set (new, old, or being modified)
- Concatenation of data sets
- Job library

(Refer to *Command System User's Guide,* for detailed information on writing DDEF commands.)

Assembler-language programmers can also use the DDEF macro instruction to define a data set. The DDEF macro instruction allows the user to specify the same data set attributes and requirements as the DDEF command.

Program Library List Control

A program in TSS/360 can consist of one, or a series of two or more object modules that are linked and are executable. An object module is the output (exclusive of the listing) of a language processor or the linkage editor. All programs in TSS/360 are stored in object module form in program libraries that are partitioned data sets. A program consisting of only one object module is stored entirely within one library; however, a program that consists of several object modules may reside in different libraries, depending on how the user has stored them. During linkage editing and during execution, the system can automatically retrieve all required object modules, if the user has defined the libraries that hold those object modules.

There are four categories of program libraries:

- System library (SYSLIB)
- User library (USERLIB)
- User-defined job libraries
- Other user-defined libraries

System library, accessible to all users, includes TSS/360 programs and the installation's standard subroutines and functions.

User library is the private library assigned to each user when he is joined to the system. This library is automatically built for him and made a part of his catalog by the system. This library is available each time he logs-on. If the user does not employ job libraries in a task, all the object modules resulting from his use of the language processors are placed in his user library. In addition, if no special library is assigned for the output of the linkage editor, the linkage editor object modules are placed in his user library.

Job libraries are defined for use within one task when the user wants to restrict his user library to checked-out, standard object modules that he executes frequently or that he uses frequently in the buildup of other object modules. Or, he may want to use a special object module that will replace one he normally would use. In any of these cases, he can use the program library list and job library facilities of the system.

The program library list, a defined hierarchy of those libraries, is set up at log-on time, and consists of the user library and SYSLIB. Job libraries designated for a task are removed from the hierarchy at log-off time.

The library at the top of the list always automatically receives all object modules resulting from language processing. If no job libraries are defined, the library at the top of the list is always the user library. However, the user can specify that a job library be added to the program library list to receive the output of the language processors. He does this by issuing a DDEF command that defines the job library and contains the JOBLIB operand. When this command is executed, the name of the job library is added to the top of the program library list. That library then receives all subsequent outputs of the language processors until another job library is defined (and *it* is placed at the top of the list), or until a RELEASE command is issued for the job library.

In addition to using the program library list to store object modules, the system uses this list to control its order of search when looking for object modules that must be loaded at execution time. The library at the top of the list is searched first, then the next-to-the-top library, etc.; then the user library and SYSLIB are searched.

The program library list can also be used, during linkage editing, to define the following for the system:

- The library that is to receive the link-edited object module.
- The sequence in which libraries are to be searched by automatic call if the system must find other object modules that will complete the link-edited object module.

For example, if no other library is specified, the output of the linkage editor is stored in the library currently at the top of the program library list. If another library is specified at the time the linkage editor run is defined, that library receives the link-edited object module. The library can be the user library, any of the current job libraries, or a special library defined by a DDEF command that has no JOBLIB operand.

During linkage editing, the library or libraries containing the object modules to be included in the link-

edited object module are either defined by INCLUDE statements in the linkage editor source program, or they are defined by the program library list. The object modules whose libraries are identified by INCLUDE statements are placed in the output module at the time the INCLUDE statement is processed. Those object modules required in the output module, but whose libraries are not defined by INCLUDE statements, are found at the end of the linkage editor source program. They are retrieved by automatic call, using the program library list as the basis of the search. (Refer to *Linkage Editor* for an explanation of this form of library control.)

Symbolic Libraries

The symbolic library is a special form of TSS/360 library that can be used apart from the conventional TSS/360 libraries (partitioned data sets); or, that can be used as a special indexing facility to supplement the capabilities of partitioned data sets.

A symbolic library consists of a symbolic component and an index component. The symbolic component is a line data set and it contains the records, or groups of records, that constitute the usable portion of the library. The index component has a virtual sequential organization; as its name implies, it contains pointers from names and aliases to all portions of the symbolic component that are to be used in library references. The two components must be distinct data sets.

The system assembler macro library, provided as a part of TSS/360, is a good example of a symbolic library. Its symbolic component contains the symbolic assembler statements that constitute the macro definitions for all system macro instructions (READ, WRITE, GET, PUT, etc.). Its index component contains a list of macro instruction names, each of which has a pointer to the first statement of the macro definition corresponding to that name.

User-written macro libraries and special symbolic libraries used for indexing data sets are examples of symbolic libraries. A user-written macro library, which can be used in addition to the system macro library during assembly, takes precedence over the system macro library. Special symbolic libraries can be used to define other access criteria for line data sets.

Concatenating Data Sets

Virtual storage data sets cannot be concatenated; i.e., automatically retrieved by the system and processed successively as a single data set. However, two or more physical sequential data sets can be concatenated by using the CONC operand in DDEF commands, with the same DDNAME for each command. The concatenation is in the order of the DDEF commands in the task. The CONC operand must be specified for all but the first data set in the concatenation; it must not be specified for the first data set definition. Refer to "RELEASE Command," below, for the method used to remove data sets from a concatenation.

These considerations apply to concatenation of sequential data sets:

1. Concatenation applies only to data sets opened for INPUT.
2. A maximum of 255 data sets can be concatenated.
3. Only one data control block is associated with all the data sets in any concatenation. The user enters DDEF commands in the order of retrieval. If data sets with unlike characteristics (e.g., block length, record format, etc.) are to be concatenated, the data control block must indicate this.
4. When the data sets to be concatenated have unlike characteristics, the user must reissue any input requests that are unfilled when the system determines that a data set that is being processed does not contain more blocks. The data set is closed, and the next data set to be processed is opened. The user should supply an entry in the exit list for his data control block exit routine, so that control may be passed to it during the opening process. From this routine, the user must be able to determine the status of all outstanding input requests; any incomplete requests must be reissued after the opening process is completed (they cannot be reissued in the user's data control block exit routine). When using the basic sequential access method, each CHECK macro instruction should be immediately followed by a test of a program switch whose value indicates whether to reissue the associated READ macro instruction and all other READ macro instructions that were issued after the READ that was just checked. The program switch should be set ON in the data control block exit routine, and set OFF whenever the test determines an ON condition.
5. When the data sets to be concatenated have identical characteristics, the system performs the concatenating function without performing the closing and opening processes described in 4, above. Therefore, the data control block exit is not taken and the user is not aware of when the concatenation process takes place.
6. The user's end-of-data set routine is not entered until the last data set has been retrieved.
7. Volume switching is automatically performed, as for a single data set on multiple volumes.
8. When two data sets on the same volume are to be concatenated consecutively, the disposition option of of the OPEN macro instruction should be LEAVE.
9. User label exit routines are executed for each data set, as requested.

NOTE: A DDEF command identical to a previously issued DDEF command, except for its DDNAME, can be used to change the DDNAME assigned to the previously issued DDEF command.

CDD Command

The CDD command is used to retrieve one or more DDEF commands from a cataloged line data set; the user must supply the name of the data set. If this is all he specifies, the system assumes that wants to use all the DDEF commands in the line data set. If he wishes to use only selected DDEF commands, he identifies each by its DDNAME.

If all DDEF commands are indicated, the system processes them in the order of their appearance in the line data set. If the user specifies selected DDEF commands, the system selects and executes the DDEF commands in the order they are named in the CDD command.

Frequently used DDEF commands should be prestored in the system and retrieved by the CDD command wherever possible. CDD can be used in either conversational or nonconversational tasks.

Assembler-language programmers can also use the CDD macro instruction to retrieve one or more DDEF commands from a prestored line data set. This macro instruction provides the same options and facilities as the CDD command.

RELEASE Command

The RELEASE command removes data sets from a concatenation, removes a job library from the program library list, and releases I/O devices associated with data sets stored on private volumes. The RELEASE command is applicable to public cataloged data sets, and to private cataloged or uncataloged sets.

The user must specify the DDNAME of the DDEF command associated with the data set involved. If the DDEF command represents a data set that is part of a concatenation, he can specify that the entire concatenation be released, or he can name a data set in the concatenation and release only that data set.

If the DDEF command represents a job library, that library's name is removed from the program library list, and that list is updated accordingly.

The RELEASE command always removes the definition of the data set from the system. The effects of the command depend upon the nature of the data set.

1. If a cataloged data set on public storage — no subsequent effect.

2. If a cataloged or uncataloged data set on private storage — I/O devices are freed for other uses only if all data sets on the volume have also had RELEASE commands issued for them.

NOTE: The RELEASE command does not uncatalog any data set.

Assembler-language programmers may also use the REL macro instruction to perform the same functions as the RELEASE command.

SECURE Command

The SECURE command reserves all devices that will be required for private volumes during the execution of a nonconversational task; SECURE is never used in a conversational task. It must appear immediately after the LOGON command, and only one SECURE is allowed for each task. The devices specified for the private volumes will be reserved so that the task can be executed without waiting for I/O devices; waiting may be necessary to reserve the devices at SECURE time rather than during execution time.

Data Set Manipulation

Copying Data Sets

The CDS (Copy Data Set) command makes a copy of an existing data set or member of a partitioned data set. Also, it can be used to renumber the lines of a line data set.

To issue a CDS command, the user must have at least read-access to the source data and he must ensure that the requirements for data set residence and data definition are met as specified in *Command System User's Guide.*

The organization of the copied data is not altered, even though the device type may be changed. However, a virtual sequential or virtual index sequential data set may be copied as a member of a virtual partitioned data set, and a virtual sequential data set or member may be copied as a virtual index sequential data set or member and vice versa.

A user who has unlimited access to a source data set that resides on a direct-access volume (he owns it or has been permitted to share it), can *optionally* specify in the CDS command that the source data be erased:

- If the source data is a data set, its space is made available for subsequent use; if the data set is also cataloged, it is removed from the catalog.
- If the source data is a member of a partitioned data set, the partitioned data set's directory is updated to make that member's space available for subsequent use.

NOTE: The data set copy that is generated must be defined by a DDEF command before the CDS command is issued. When the data set is defined, it is cataloged by the system, if it resides on public storage.

Assembler-language programmers can also use the CDS macro instruction to copy an existing data set. They must follow the same rules for data set organization and residence, when using this macro instruction, as when using the CDS command.

In addition to the facilities available with the CDS command, the user can use three commands for copying virtual storage data sets.

The VT (VAM-to-tape) command copies a data set with virtual storage organization from direct-access storage onto a 9-track tape as a physical sequential data set. The TV (tape-to-VAM) command restores a data set to VAM-formatted direct-access storage from a 9-track tape produced by the VT command. The VV (VAM-to-VAM) command copies a virtual storage data set from one direct-access storage device to another; this command can be used to copy an entire VPAM data set.

These commands can be used to conserve VAM public storage or to interchange data on tape reels between systems.

Erasing Data Sets

The ERASE command releases the direct-access storage that is allocated to specified data; if the specified data is a cataloged data set, the command removes the data set's entry from the catalog. The data must be a data set that is either cataloged or defined in a previous DDEF command. The types of data that can be specified in the ERASE command, and the effects of that command, are summarized in Table 2.

Table 2. Effects of ERASE Command

Data Type	Effect of ERASE Command
Cataloged data set	Data set's direct-access storage is released for other use; data set's name is removed from catalog
Uncataloged data set	Data set's direct-access storage is released for other use
Member of partitioned data set	Member's name is removed from directory of partitioned data set
Group of cataloged data sets or generation data group	All data sets indexed under partially qualified name, or all generations in generation data group, are located. In conversational mode, name of each is presented to user; he is asked to enter E (to erase) or R (to retain) each; if E, storage is released and associated catalog entry is removed; if R, no change in storage or catalog — in nonconversational mode, all data sets or generations specified are erased.

Data is also erased:

1. At log-off, for conversational tasks, the user can specify that the data set on a private volume is to be cataloged or erased.
2. After PUNCH, PRINT, WT, or CDS, if the ERASE option has been specified.

If the user attempts to erase a shared VAM data set that is currently being shared by other users, the system issues a diagnostic message and ignores the command.

Text Editor

The text editor is a facility for creating or altering line or region data sets. With it, the user can

- correct, insert, or delete lines
- transfer lines from one data set to another
- segment the lines of a data set
- display lines in SYSOUT
- nullify previous editing

The text editor is invoked by the EDIT command or the PROCDEF command. Text-editing is halted with the END command (which should not be confused with the FORTRAN or assembler language END statement).

A data set can be created by issuing a text-editor command to input data. These commands may be used:

- REGION — creates a region of a region data set; the system prompts with line numbers.
- EXCERPT — enters a region or a range of lines from another data set.

Like all text-editor commands, REGION and EXCERPT can also be used to edit an existing data set. All text-editor commands can be used to edit a data set while creating it — that is, to edit the lines already entered.

Nullifying Changes

The text editor maintains a transaction table to record changes that are made to a data set through its facilities. Additions and deletions are recorded separately. When a text-editor command alters data, the changes are recorded in the table. This table facilitates nullification of changes that recorded in it.

The STET command causes all changes that are recorded in the transaction table at the time the command is issued to be reversed. Additions that are recorded in the table are deleted from the data set and deletions that are recorded are added to it. These changes that are made by the STET command are also recorded in the table; in effect, the addition and deletion columns of the table are switched.

In normal operation, any text-editor command that alters data causes all previous entries in the transaction table to be removed. Therefore the STET command, during normal operation, can reverse only the

changes that were made by the last command that altered data.

The DISABLE command, however, can be issued to prevent entries in the table from being removed. When a DISABLE command is in effect, the text editor is considered to be disabled; text-editor commands are executed, but their effect can be reversed with the STET command. (If a single line of data has been changed by more than one text-editor command, however, only the last change to that line can be reversed.)

The effect of the DISABLE command can be canceled by the ENABLE command. This command does not remove entries from the transaction table; it causes the next text-editor command that alters data to have the effect of removing entries. Therefore a STET command issued immediately after an ENABLE command would still result in reversing all the changes made since the DISABLE command.

The text-editor commands that do not alter data (and therefore do not cause the transaction table to be cleared of previous entries) are: DISABLE, ENABLE, LIST, and LOCATE.

Text-Editor Commands

Listed below are the text-editor commands with brief explanations of the function of each.

- The CONTEXT command replaces a string of characters within a line, a range of lines, or a region data set with another character string. Every occurrence of the first character string is replaced by the second character string.
- The CORRECT command changes or inserts characters in one or more lines of a region. The corrections are made as indicated by the lines that follow this command from SYSIN.
- The DISABLE command — discussed above — causes modifications made to a data set to be recorded in a transaction table so they can be nullified.
- The EDIT command invokes the text editor. (It is also invoked by the PROCDEF command.)
- The ENABLE command — discussed above — cancels the effect of a previous DISABLE command.
- The END command terminates processing by the text editor. It denotes the completion of editing initialized by the EDIT command, or completes the contents of a PROCDEF command procedure.
- The EXCERPT command inserts a region or range of lines from another data set into the data set being edited. Following a REVISE command, it replaces a range of lines in the current data set; following an INSERT command, it adds to lines that were entered from SYSIN.
- The EXCISE command deletes a line or a range of lines from a region.
- The INSERT command inserts into a region the lines that follow this command in SYSIN.
- The LIST command displays a line or a range of lines in SYSOUT. It does not alter data, and therefore does not cause entries to be removed from the transaction table.
- The LOCATE command searches a region of a region data set for a specified character string. It does not alter data, and therefore does not cause entries to be removed from the transaction table.
- The NUMBER command renumbers a line or a range of lines within a region or within contiguous regions of a region data set.
- The REGION command prefixes a region name to a line number or to a series of line numbers, designating the line or lines as a region. This command can also be used to position the data set so that text-editor commands can be entered.
- The REVISE command specifies a point or range in a data set and inserts into the data set at that point or range the line or lines that follow the command in SYSIN.
- The STET command — discussed above — reverses the effect of the changes to a data set that are recorded in the transaction table at the time that the command is issued. When the text editor is enabled, it cancels the effect of the last text-editor command that altered data; when the text editor is disabled, the STET command cancels the effect of all text-editing that was done since the DISABLE command was issued.
- The UPDATE command adds or inserts into a data set the lines that follow the command in SYSIN.

Break Characters

During text-editing, the underscore character can be entered as the first character in a line to cause the system to consider what follows as a command, not as data. The vertical bar can be entered as the first character in a line to cause the system to consider what follows as a control statement for a language processor. Used this way, the underscore and the vertical bar are known as *break characters.* (The user may also define other characters as break characters.)

When a break character is repeated once or more at the beginning of a line, the effect of the first break character in that line is canceled. The system processes the line as if no break characters had been entered, removing the first break character and treating the remainder of the line as data. This permits the user to enter lines beginning with break characters into command procedures or data sets.

Data Editing

The data-editing commands duplicate some functions of the text-editor commands. With one of the data-editing commands, however, the user can create virtual sequential data sets; wih another of the data-editing commands, he can create a virtual index sequential data set with the key field embedded in each record. The text editor cannot be used to create a virtual sequential data set or to create a virtual index sequential data set other than a line or region data set.

The DATA Command

With the DATA command, the user can create a line data set or a virtual sequential data set. If the data set is to reside on public storage, it need not be previously defined; if it is to be on private storage, the user must issue a DDEF command for it before issuing the DATA command. If the data set is on public storage, the system automatically defines and catalogs it.

The user can modify, delete, or insert lines of a line data set being created with the DATA command by specifying the line numbers of the lines he wants to modify, delete, or insert.

The MODIFY Command

With the MODIFY command, the user can insert, delete, replace, or review lines of a virtual index sequential data set or a virtual index sequential member of a virtual partitioned data set. He can also create a virtual index sequential data set or member.

To use the MODIFY command on an existing data set, the data set must be defined within the current task or be cataloged. A data set to be created with the MODIFY command need not be previously defined or cataloged.

If the user requests review when he issues the command for an existing data set, the system will display each line that is modified as the line was *before* it was modified.

Strings of hexadecimal digits may be entered with the MODIFY command. The system will convert the digits as entered from the terminal into machine representations of hexadecimal digits. This permits the user to enter data he could not represent otherwise.

The MODIFY command can be used to create a virtual index sequential data set or member by entering lines as if they were a series of insertions. A virtual index sequential data set or member that is not a line data set can be created by including keys in the records and giving the key position and length. These key values can then be used just like the line numbers of a line data set — to insert, replace, delete, or review lines of the data set.

The MODIFY command offers less flexibility in editing than the text-editor commands. It does, however, permit a key anywhere in the records of a virtual index sequential data set or member, while the text editor works only with line and region data sets, which have the key in the first position.

The LINE? Command

Lines in any line data set the user owns or shares can be displayed in SYSOUT with the LINE? command. A single command can be used to display one line, a range of lines, or several ranges of lines.

The LINE? command will display lines of a listing data set — that is, a data set consisting of the listings specified as optional output of a language processor or the linkage editor. A listing data set is stored with its line number in the last position of each line rather than the first position. However, when a listing data set is displayed, its line numbers appear to be in the first position.

Inquiry Commands

Inquiry commands request specific information from the system. In conversational tasks, this information is displayed at the user's terminal; in nonconversational tasks, the information is sent to the task's SYSOUT data set.

The DSS? Command

The DSS? command is used to request presentation of the status of one or more cataloged data sets, with this information:

1. Sharing status; ownership and sharability
2. Access status — read-only, read-write, or unlimited
3. Resident device type and volume number
4. Dates of creation and expiration
5. Organization and (for virtual-storage data sets) date last used
6. Record format and length

The POD? Command

The POD? command is used to request a list of the member names of individual members of cataloged virtual partitioned data sets (and, optionally, the alias names and other member-oriented information). Users can conveniently examine the contents of a user library, cataloged job libraries, and other cataloged virtual partitioned data sets.

Operator-Assisted Input

The user can also enter data sets by means of nonconversational tasks that are initiated by the system

operator. A nonconversational SYSIN data set can be submitted on punched cards; after being entered as a data set, it will be executed as a nonconversational task. The user can submit a virtual sequential or a virtual index sequential data set on punched cards or magnetic tape to be entered and stored on a public volume. The tasks that enter data sets from cards or tape are initiated by the RC and RT commands, which are issued by the operator.

Bulk-Output Commands

With bulk-output commands, the user can transfer data sets from virtual storage to output devices other than his terminal. These devices at the central computer installation can output data sets faster than the terminal and – for two of the bulk-output commands – in forms not available at the terminal.

The bulk-output commands are PRINT, PUNCH, and WT (Write Tape). Each initiates an independent nonconversational task to which the system assigns a batch sequence number (BSN) for reference.

The PUNCH and WT commands are valid only for virtual sequential or virtual index sequential data sets; the PRINT command can be used on physical sequential data sets on magnetic tape as well as virtual sequential and virtual index sequential data sets. None of these commands can be used on a member of a virtual partitioned data set. However, the user can copy a member of a virtual partitioned data set with the CDS command and then issue a bulk-output command for the copy.

The PRINT command causes a data set to be output on the installation's high-speed printer. The PUNCH command causes a data set to be punched into cards by the high-speed card punch. The WT command causes a data set to be written on magnetic tape for later output on the printer or for off-line processing.

If the data set specified by any of these commands is not cataloged, it will be cataloged until output is completed and then erased. If it is already cataloged when the bulk-output command is issued, the user may specify that it be erased after being output.

A task initiated by one of these commands can be canceled with the CANCEL command, using the batch sequence number assigned by the system, provided the task has not already been completed when the CANCEL command is issued. If the task is canceled while partially completed, a message will explain why it was terminated.

The data set should not be used while bulk output is pending. If it is, the results will be unpredictable.

Input/Output During Program Execution

TSS/360 includes complete program I/O facilities for the conversational and nonconversational modes of operation. In both modes, conventional I/O and dynamic I/O facilities are provided. Depending upon the application, these dynamic facilities can be used alone, or in conjunction with those for conventional I/O.

With conventional I/O, the source program must contain all of the instructions required for I/O operations, and data to be processed must be made available and defined for the system prior to program execution. In effect, data processing must be preplanned in detail.

With dynamic I/O, the source program need not contain instructions for conventional I/O. All I/O can be achieved via SYSIN/SYSOUT, using either dynamic I/O source statements or program-control commands. Conditional dynamic I/O is possible. Data to be processed can be decided upon, based on results of processing; no predefinition for the system is required.

The program-control commands and the I/O facilities of the programming languages available with TSS/360 are explained and illustrated in *FORTRAN Programmer's Guide* and *Assembler Programmer's Guide.*

Appendix A: TSS/360 Glossary

Italicized terms are defined in this glossary.

access method — Any of the *data management* techniques that are available to the *user* for transferring data between *virtual storage* and an I/O device.

address constant (ADCON) — A value, or an expression representing a value, used in the calculation of *real* or *virtual storage addresses;* a word of *program text* that changes as a result of *relocating* the *program* in storage.

address translator — A feature of System/360, Model 67, for dynamically translating instruction and data addresses, which a *program* generates during its execution, by use of a table-lookup procedure.

alias — 1. An alternate name that may be used to refer to a member of a partitioned *data set;* 2. an alternate *entry point* by which the *program* (that is, a stored member of a partitioned *data set*) may be called.

allocate — To grant a *resource* to, or reserve it for, a *task.*

asynchronous — Not synchronized with respect to any other *event* (which is usually in a timed sequence); an *event* whose occurrence is not synchronized with *program* execution and, hence, may occur at any time.

attribute — A characteristic that is incorporated in a definition; for example, attributes of data include *record* length, *record* format, *data set* name, associated device type, and *volume* identification, use, and creation date.

auxiliary storage — Storage used for recording *pages* of a *task* whose *main storage* space is required for reallocation to another *task;* a form of secondary *storage* formatted into *pages.*

availability — A measure of a system's capability to perform its intended function.

background — on a *time-sharing* system, *tasks* that are executed *nonconversationally;* i.e., the *batch-processing* workload.

block — (n.) A generic term referring to a sequential collection of related items (e.g., storage block, *record* block); (v.) to group *records* to conserve storage space or increase the efficiency of access or processing.

buffer — (n.) (1) A storage device used to compensate for a difference in the rate of data flow, or time of occurrence of *events,* when transmitting data from one device to another; (2) portion of *main storage* into which data is read, or from which it is written; (v.) the act of filling and emptying buffers.

call — The transfer of control from a *routine* to a *subroutine.*

catalog — (n.) An indexed file of all *data set* names and location *indexes;* (v.) to include the name and location *index* for a *data set* in the collection of such *indexes.*

cataloged data set — A *data set* that can be located by its representation in an *index* or hierarchy of *indexes.*

command procedure — a sequence of commands defined by the user with the PROCDEF command to be invoked and executed as a single command.

control section (CSECT) — The smallest relocatable unit in a *program;* that portion of *text* specified by the programmer to be an entity, all elements of which are to be *allocated* contiguous *virtual storage* locations.

conversational — The interaction or dialogue between a *user* and a *time-sharing system,* via a terminal device, whereby the *user* is able to enter his program into the system and, thereafter, to control, interrogate, modify, and observe processing.

data control block (DCB) — A *system control block* through which the information required by access *routines* is communicated.

data definition name (DDNAME) — A *name* appearing in a *data control block* that corresponds to the *name* of the data definition statement for that *data set.*

data management — A general description of the collective functions of that part of a programming system that provide access to *data sets,* enforce data storage conventions, and regulate the use of I/O devices.

data organization — A reference to the arrangement of a *data set* in conformance with the *data management* conventions.

data set — A named collection of logically related data items, arranged in a prescribed manner, and described by control information to which the programming system has access.

data set control block (DSCB) — A *system control block* located within the volume table of contents (VTOC) that describes and points to a *data set.*

data set label — A collection of information that describes the *attributes* of a *data set*; a general term for *data set control blocks* and tape *data set labels.*

DDEF command — A command system statement that describes the attributes, residence requirements, and status of a *data set.*

device independence — The ability to request I/O operations, the nature of which is determined by the characteristics of the *data set,* rather than by the type of device upon which the data is stored.

direct-access — A type of storage medium that allows information to be accessed by positioning the medium or accessing mechanism directly at the location of the information required.

dump — (1) To copy the contents of all or part of storage onto an output device so it can be examined; (2) data resulting from 1; (3) a *routine* that will accomplish 1.

dynamic program loading — The process of *loading* a *program* module into *virtual storage,* based on an explicit or implicit reference to that *program* module by an executing *program,* rather than *loading* it without regard for whether it will be referred to by an executing *program.*

dynamic program relocation — Moving or relocating a *program* to another part of storage without modifying it, before it has completed execution, and yet permitting its subsequent execution.

dynamic storage allocation — Providing storage space to a *procedure* in response to the instantaneous or actual demand for storage space by that *procedure,* rather than its anticipated or predicted demand.

entry point — Any location in a *program* to which control can be passed by another *program.*

EODAD (end of data address routine) — A *routine* that receives control from the system when an attempt is made to read a *record* beyond the last *record* in a *data set.*

event — An occurrence of significance to a *task*; typically, the completion of some *asynchronous* activity.

event control block — A control block used to represent the status of an *event.*

extent — The locations or range of locations, occupied by or reserved for a particular *data set,* on I/O devices or *volumes.*

external reference — A reference to a symbol defined in another *program* module.

external storage — The part of *secondary storage* that is available to *users* for data storage (i.e., the part of *secondary storage* not used for *auxiliary storage*).

external symbol — A symbol whose value is used by a number of *program* modules, not just the *program* module in which it is defined; a symbol contained in a *program module dictionary.*

fetch — Transmit information from *main storage* to a central processing unit.

foreground — A generic reference to *tasks* being executed by a *time-sharing system,* while there is dialogue or *conversational* interaction between the *user* and the system.

generation data group — A collection of successive, historically related *data sets.*

index (data management) — (1) A table in the *catalog* structure used to locate *data sets*; (2) a table used to locate the *records* of an index sequential *data set.*

installation — A general term for a computing center, and the individuals who manage it, operate it, apply it to problems, service it, and use the results it produces.

job library — A set of *user*-identified *object program modules.*

language processor — A general term for any assembler, compiler, or other *routine* that accepts statements in one programming language and produces equivalent statements in another.

library — A collection (for example, *data sets* or *volumes*) associated with an application, the location of which is identified in a directory.

line data set — A virtual index sequential *data set* that is organized by line number with keyboard/printer-length *records.*

linkage — The means by which communication and interaction are effected between two *programs.*

linkage editor — A *program* that transforms program modules into a simple *object program module*; also, it resolves symbolic cross-references among them, and, optionally, combines separate *control sections* into a single *control section*, replaces, deletes and adds *control sections.*

load — The process of placing the beginning of a *program* into *virtual storage* and making necessary adjustments to the *program* so that it may have control transferred to it.

locate mode — A mode of *data management* operation in which *buffer* locations are pointed to, rather than moving *records* to or from *user*-supplied work areas (see "move mode").

location-free procedure — *Procedure* that is independent of its location in storage; *procedure* does not contain any constants that must be changed as a result of relocation in storage.

logical record — A *record* that is defined in terms of the information it contains, rather than its physical *attributes.*

macro instruction — A collective description of a macro instruction statement, the corresponding macro definition, the resulting assembler language statements, and the machine language instructions and data produced from the assembler language statements; loosely, any one of these representations of a machine language instruction sequence.

main storage — Storage that is directly addressed by the registers of a central processing unit; hence, storage from which *program* instructions may be fetched and executed.

mapping device — See "address translator."

module (programming) — The input to, or output from, a single execution of an assembler, compiler, or *linkage editor*; a source module or *program* module; hence, a *program* unit that is discrete with respect to compiling, combining with other program units, and *loading.*

move mode — A *data management* service in which a data *record* is moved between a *buffer* and a *user*-supplied work area.

multiprocessing — The simultaneous use of two or more processing units in the same computing system.

multiprogramming — A technique by which a computing system can be used to execute two or more unrelated *programs,* parts of which reside in *main storage.*

name — A string of characters that identifies a statement, *data set,* or *program.*

nonconversational — Processing of a TSS/360 task, during which there is no dialogue between the *user* and the *time-sharing system.*

object program module — The output of a single execution of an assembler, compiler, or *linkage editor,* which constitutes input to the dynamic loader or the *linkage editor*; an object module consists of one or more control sections in relocatable (but not executable) form, and an associated *program module dictionary.*

off-line — A generic reference to a device that is not directly controlled by the computing system with which it is associated.

on-line — A generic reference to a device that is directly controlled by a computing system with which it is associated and connected electrically.

on-line storage — Storage devices, especially the storage media which they contain, under the direct control of a computing system.

one-level storage — A technique that treats all *on-line storage* as though it were *main storage.*

owner — The creator of a *data set,* who may issue PERMIT commands so that other *users* may gain access to the *data set.*

page — A set of 4096 consecutive bytes; applied to *main storage,* the first byte of which is located on a page boundary.

page boundary — A TSS/360 storage address that is a multiple of 4096; in System/360, an address whose 12 low-order bits are 0.

paging — The process of transmitting *pages* between *main storage* and *auxiliary storage* to assist in allocating *main storage* among concurrently executing *programs.*

PCSOUT — A *data set* for collecting *dumps;* it is printed automatically at log-off.

post — To record, in a *system control block,* the occurrence of an *event* for later interrogation by a *procedure;* the procedure's action depends upon the status of the *event.*

privileged — (1) Privileged *user*: one who is authorized to execute a class of system control commands from a *terminal*; (2) privileged module: a system module that is allowed to communicate directly with the TSS/360 *supervisor.*

privileged program state — Within TSS/360, an operating state implemented via hardware and programming.

problem program — Any *routine* that performs processing of the types for which a computing system is intended; including *routines* that solve problems, monitor and control industrial processes, sort and merge *records*, perform computation, process transactions against stored *records*, etc.; generally interpreted to be any nonsystem program.

procedure — The step-by-step process for the solution of a problem; the machine instructions for arriving at the solution.

processing programs — Any *programs* capable of operating in the *problem program* mode, including IBM-distributed *language processors*, application *programs*, and *user*-written *programs*.

program — (n.) A collection of instructions and data produced for the solution of a well-defined problem; (v.) to create these collections.

program module dictionary (PMD) — The collection of control and descriptive information, concerning a *program module*, required by *programs* that must process that *module*.

protection key — A task-associated indicator that appears in the program status word (PSW) whenever the *task* is active; the indicator must match the *storage keys* of all *storage protection blocks* that the *task* is to use.

qualified name — A *data set* name composed of multiple names separated by periods (for example, TREE.FRUIT.APPLE).

read-only procedure — A collection of instructions and data that is never modified during its execution.

ready condition — The condition of a *task* that is in contention with other *tasks* for use of the central processing unit, all other requirements for its activation having been satisfied.

real address — The actual address used by the System/360, Model 67, for fetching or storing into processor storage, as compared with the address that serves as input to the *address translator*.

real storage — The *main storage* of System/360, Model 67, rather than its conceptual extension.

real time — The actual time during which a physical process transpires, especially if that process is monitored or controlled by a computing system.

record — A general term for any unit of data that is distinct from all others within a *data set*.

recursive — Repetitive on a cyclic basis; a *program* that calls or transfers control to one of its own *entry points*.

reenterable — The *attribute* of a *module* that allows the same copy of the *module* to be used concurrently by two or more *tasks*.

relocation — The movement of a *program* from one place in storage to another; any required modifications are made and incorporated in the move.

resources — The hardware and *programs* that are required by a *task*.

response time — The average time that a *terminal user* must wait to receive a response from the *time-sharing* system.

return code — A code that is, by system convention, placed in a designated register (the *return code register*) at the completion of a *program*. The value of the code, which is established by the *user*, may be used to influence the execution of succeeding *programs* or, in the case of an abnormal end-of-*task*, it may be printed for analysis.

return code register — A register, identified by system convention, in which a return code is placed by a *program* at its completion.

reusable — The *attribute* of a *routine* that permits the same copy of the *routine* to be used by two or more *tasks* (see "reenterable" and "serially reusable").

routine — A sequence of machine instructions that directs the execution of a well-defined function.

scheduling algorithm — A series of rules and decisions that must be made for the determination of how time in a central processing unit and other system resources is to be allocated among *tasks*.

secondary storage — The *on-line* storage of a computing system that is not directly addressed by the registers of a central processing unit.

segment — An area of contiguous *virtual storage* equal to 256 *pages*, and on a 256-*page* boundary.

sequential access storage — A storage device from which the data *records* must be accessed in a serial or sequential manner.

serially reusable — The *attribute* of a *routine* that, when in main storage, permits the same copy of the *routine* to be used by another *task*, after the current use has been completed.

service program — A system *program* that assists in execution of *problem programs*, without directly controlling the system or producing results.

session — The period of time during which the *user* engages in a dialogue with the *time-sharing* system.

shared data set — A *data set* to which the *owner* has granted access to other *users*.

shared routine — A *routine* that can be concurrently executed by several *users*.

sharer — A *user* who issues a SHARE command to gain access to a *data set* for which the *owner* has issued a PERMIT command.

source module — In the symbolic language of an assembler or a compiler, a series of statements that constitutes the entire input for a single execution of the assembler or compiler; all *source modules* are in *line data set* format.

storage key — Indicators associated with *storage protection blocks*, which require that *tasks* (if they are to use the blocks) must have matching *protection keys*.

storage protection block — A block of 2048 contiguous bytes of storage, with a starting address that is a multiple of 2048, and an associated storage *protection key*.

subroutine — A *routine* that is called by another routine to perform a specific function.

supervisor — The *program modules*, supplied by IBM, that control and monitor the usage of the *time-sharing* system.

supervisory programs — The *programs* that schedule, *allocate*, and control system *resources*, rather than process data.

SYNAD (synchronous error exit routine) — A *routine* that receives control from the system, if an error occurs when data is being processed.

synchronous — Occurring with a regular or predictable time relationship.

SYSIN — The *data set* that contains the input to the system.

SYSOUT — The *data set* containing the system output.

system control block — A storage area, containing information in a predetermined format, that is used by several TSS/360 routines.

system input unit — An *on-line* device used by the system as the source of commands.

system library (SYSLIB) — A partitioned *data set* whose members are the TSS/360 *programs*.

system macro instruction — A predefined *macro instruction* whose expansion provides some system service or *linkage* to a system service *routine*; e.g., GET, PUT, CALL, SAVE.

system output unit — An *on-line* device that is the destination of system messages and task output.

system residence volume — The *volume* that contains all *data sets* required for the initiation and operation of TSS/360.

task — All work performed by TSS/360 under the direction of a stream of commands from a *system input unit* that is associated with a *user*, between the LOGON and LOGOFF commands. Some *background* tasks are initiated for a *user* by the system, and are terminated at the completion of the operation.

text — The instructions and data of an *object program module*.

throughput — A measure of the volume and rate at which work can be performed by a computing system.

time sharing — A method of using a computing system that allows a number of *users* to execute programs concurrently and to interact with them during execution.

time slice — The time segment allocated to a single user, during a complete execution-time cycle for all *users* of a *time-sharing* system.

turnaround time — The elapsed time between submission of a *task* to a computing center and the return of results.

user — Anyone who uses the services of a computing system.

virtual address — An address generated by a *program* that references *virtual storage* and must, therefore, be translated into a *main storage* address as it is used.

virtual storage — Storage accessible to the TSS/360 *user*.

volume — A portion of a unit of a storage medium that is accessible to a single read/write mechanism.

volume table of contents (VTOC) — A table, in a *direct-access volume*, that describes the data on the *volume*.

wait condition — The condition of a *task* that is dependent on *events* before it can enter the *ready condition*.

Index

READER'S COMMENT FORM

IBM System/360 Time Sharing System
Concepts and Facilities

Form C28-2003-3

- Your comments, accompanied by answers to the following questions, help us produce better publications for your use. If your answer to a question is "No" or requires qualification, please explain in the space provided below. Comments and suggestions become the property of IBM.

	Yes	No
• Does this publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

- What is your occupation? _____
- How do you use this publication?

As an introduction to the subject?	☐	As an instructor in a class?	☐
For advanced knowledge of the subject?	☐	As a student in a class?	☐
For information about operating procedures?	☐	As a reference manual?	☐

Other _____

- Please give specific page and line references with your comments when appropriate.

COMMENTS:

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . .

This publication is one of a series which serves as reference for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 34
YORKTOWN HTS., NY

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM Corporation
PO Box 344
2651 Strang Boulevard
Yorktown Heights, N.Y. 10598

ATTN: Time Sharing System/360
Programming Publications Dept. 561

Fold Fold

Printed in U.S.A. C28-2003-3

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Printed in U.S.A. C28-2003-3

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]