IBM System/36

Concepts and Programmer's Guide

IBM System/36

Concepts and Programmer's Guide

Program Numbers 5727-SS1
5727-SS6

File Number
S36-30

Order Number
SC21-9019-5

Sixth Edition (June 1987)

This major revision makes obsolete SC21-9019-4.

This edition applies to Release 5, Modification Level 1, of IBM System/36 System Support Program Product (Program 5727-SS1 for 5360 and 5362 System Units and Program 5727-SS6 for the 5364 System Unit), and to all subsequent releases and modifications until otherwise indicated. Also, this publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

The numbers at the bottom right of illustrations are publishing control numbers and are not part of the technical content of this manual.

Publications are not stocked at the address below. Requests for IBM publications should be made to vour IBM representative or to your IBM-approved remarketer.

This publication could contain technical inaccuracies or typographical errors. A form for readers' comments is provided at the back of this publication to make comments about this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department 245, Rochester, Minnesota, U.S.A. 55901. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

When You Are:	You Can Find Information In:
Communicating with Another Computer or Remote Device	Using System/36 Communications Communications and Systems Management Guide Distributed Data Management Guide Advanced Peer-to-Peer Networking (APPN) Guide Remote Operation/Support Facility Guide MSRJE Guide 3270 Device Emulation Guide SSP-ICF Guide and Examples Interactive Communications Feature: Base Subsystems Reference Interactive Communications Feature: Finance Subsystem Reference Interactive Communications Feature: Upline Subsystems Reference Interactive Communications Feature: Programming for Subsystems and Intra Subsystem Reference Using the Asynchronous Communications Support (message manuals)
Determining the Cause of a Problem	(message manuals) System Problem Determination (online problem determination)
Planning for and Adding New Programming Support	Planning for New Devices and Programming Available at Release 5 (new program product manuals) Changing Your System Configuration
Using Your IBM PC with the System/36	5250 Emulation Program User's Guide Enhanced 5250 Emulation Program User's Guide PC Support/36 Technical Reference PC Support/36 User's Guide PC Support/36 Organizer
Managing Your System/36 Office	System/36 in the Office Planning Your System/36 Office Setting Up Your System/36 Office Administering Your System/36 Office Planning for System/36 Office in a Network Administering Personal Services/36 in the Office
Doing Office Tasks	Getting Started with Interactive Data Definition Utility Getting Started with Query/36 Getting Started with Personal Services/36 Getting Started with DisplayWrite/36 Business Graphics Utilities/36 User's Guide Practicing with DisplayWrite/36 Practicing with Personal Services/36 Practicing with Query/36 Practicing with Interactive Data Definition Utility Using DisplayWrite/36 Online Information: Query/36 Personal Services/36 DisplayWrite/36 Interactive Data Definition Utility

S9015015-9

When You Are:	You Can Find Information In:
Planning to Install Your Computer	What to Do before Your Computer Arrives Converting from System/34 to System/36 System/34 to System/36 Migration Aid Planning for New Devices and Programming Available at Release 5
Getting Your Computer Ready to Use	Setting Up Your Computer Installing Your New Features Performing the First System Configuration System Security Guide Updating to a New Release
Operating Your Computer	Learning About Your Computer Operating Your System Development Support Utility Guide Source Entry Utility Guide Data File Utility Guide Work Station Utility Guide Changing Your System Configuration Using and Programming the 1255 Magnetic Character Reader Using Your Display Station
Programming Your Computer	(language manuals) RPG II, BASIC, COBOL (message manuals) FORTRAN IV, Assembler Concepts and Programmer's Guide System Reference Getting Started with Interactive Data Definition Utility Development Support Utility Guide Source Entry Utility Guide Creating Displays: Screen Design Aid and System Support Program Data File Utility Guide Sort Guide Functions Reference Overlay Linkage Editor Guide System Measurement Facility Guide Character Generator Utility Guide Ideographic Sort Guide

S9015001-9

Contents

About This Manual

Who should use this manual . . .

This manual provides programmers with information needed to design and develop application programs for System/36. It also describes the system components and how (in general terms) the system works.

How this manual is arranged . . .

This manual contains the following chapters.

Chapter 1 does the following:

- Defines applications and programs.
- Introduces the system's productivity aids and programming tools.
- Provides a simple explanation of how the system works and what its components are.
- Provides brief explanations of the application components that the manual will help you design and develop.

Chapter 2 provides an overview of the application design process. It also describes each step of the process in moderate detail.

Chapters 3 through 19 provide concepts, design guidelines, and programming guidelines for the various application and system components.

Chapter 20 describes ideographic character concepts and considerations for the ideographic version of the SSP.

Chapter 21 provides a summary of the major considerations presented in the manual.

Appendix A describes access algorithms that can be used with direct files.

Glossary provides definitions of terms used in this manual.

So, if you are about to design and program your *first* application, read Chapter 1 and Chapter 2 before you begin, then refer to Chapters 3 through 19 as you need them.

If you are an *experienced* application programmer, you can get right into the details presented in Chapters 3 through 19.

Note: This manual follows the convention that **he** *means* **he** *or* **she**.

How this manual has changed . . .

The major changes are described below.

Information about restricting users from creating folders when using IDDU or DisplayWrite/36 was added to Chapter 10, "Folders."

Information about restricting the size of folders created when using DisplayWrite/36 was added to Chapter 10, "Folders."

Note: This manual may refer to products that are announced, but are not yet available. Such information is for planning purposes only and is subject to change before general availability.

What you should know . . .

Before you read this manual, you should be familiar with data processing concepts (such as disk storage, files, and members) and with the system and its display stations.

In this manual, personal computer means one or all of the following:

- IBM 5150 Personal Computer
- IBM 5160 PERSONAL COMPUTER XT[1]
- IBM 5170 PERSONAL COMPUTER AT®

If you want more information . . .

If you want more information on a topic while you are using this manual, refer to the *Guide to Publications*, GC21-9015. System/36 PC users should refer to the *Guide to Optional Information*, GX21-9817, for related publications.

[1] Trademark of IBM

Chapter 1. Introduction

Welcome to application programming on System/36! Whether you are ready to design and program your first application or you have designed and programmed applications before, this manual has useful information for you.

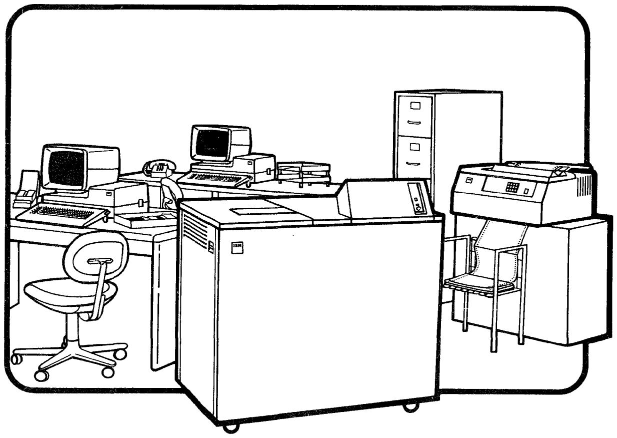
An **application** is a group of programs that apply to a particular business function, like accounts receivable or inventory control. Because there are countless types of applications and because each one can be unique, this manual cannot provide all the details for the application you might have in mind. What this manual can do, however, is:

- Describe how the system works (not all the details; but the important ones for programming the system).
- Describe the steps that application programmers follow when they design and program applications.
- Describe how to design and program the various parts of an application (printed reports, records, files, libraries, menus, displays, message members, programs, jobs, and procedures).

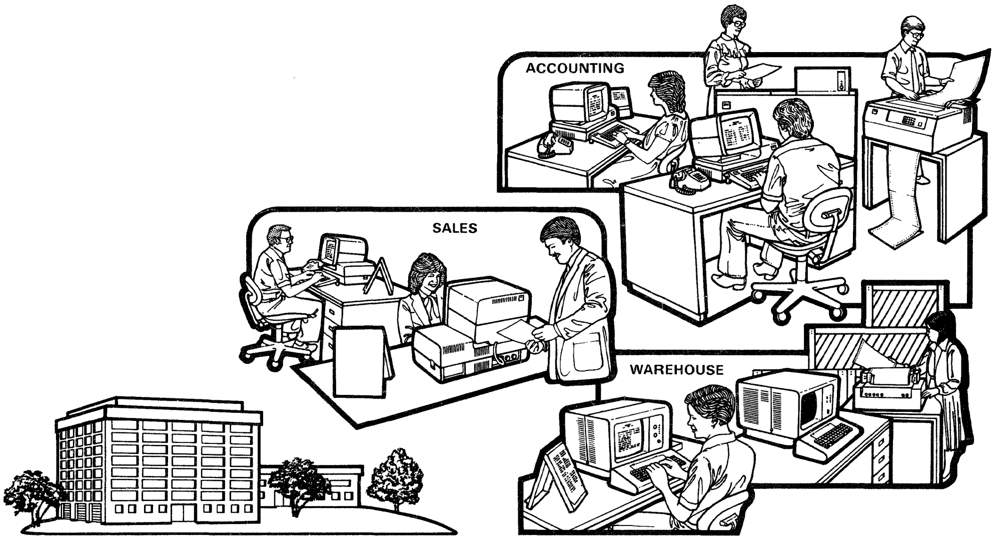
This introductory chapter describes programming on the system, productivity aids, the system in general, and applications in general.

Programming on Your System

Do you see anything missing from this picture?



Of course, there are no **people** in it; and that is where you come in. You bring the system to life by providing application programs for the users.



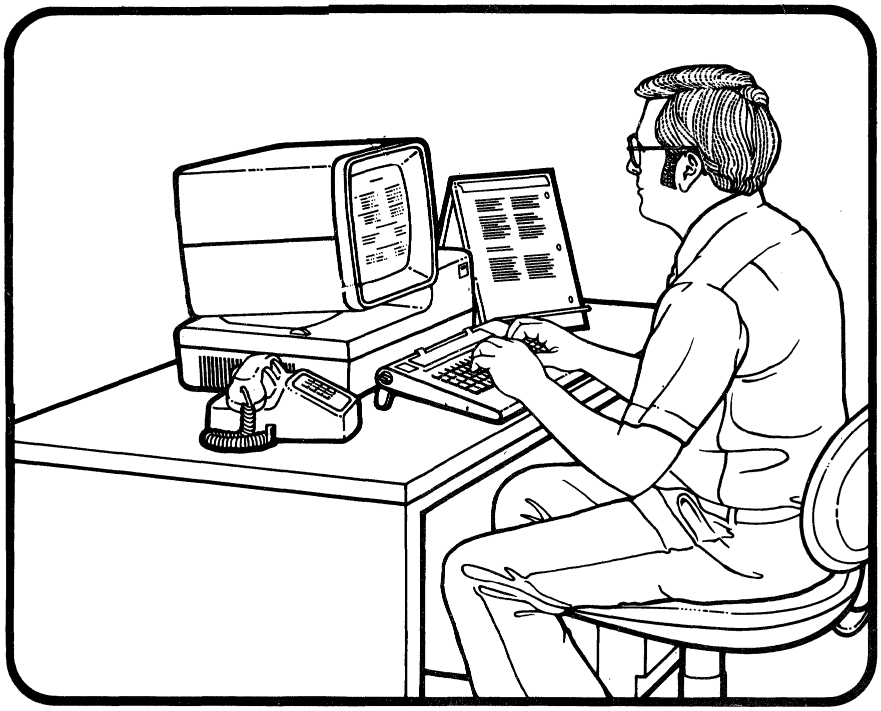
You translate users' requests into reliable, easy-to-use programs that make the system work. Your programs must help people do their work naturally, quickly, and accurately, or they will not be satisfied with the system. You have quite a job: to see that the system helps people do their jobs better.

Programming Tools

The system has several programming tools that you can use to do your work:

- System Support Program Product (SSP)
- Utilities Program Product
- Development Support Utility
- Business Graphics Utilities/36
- OFFICE/36 Program Products
- PC Support/36 Program Product
- Programming languages

Most programmers use the SSP, Utilities Program Product, and at least one programming language to do their work. It pays to know the capabilities of each tool, because choosing the right one for the job at hand can help you be more productive.



System Support Program

The System Support Program provides the procedures, operation control language statements, utility control statements, and control commands that you use to:

- Create and maintain disk files
- Sort disk files
- Create and maintain libraries
- Create and maintain folders
- Process information on diskettes
- Create and maintain display formats, menus, and message members
- Create and maintain procedures
- Run programs and procedures
- Create and maintain ideographic characters using the character generator utility (CGU).

You need not code your own procedures and programs to do these typical tasks.

The SSP also includes IDDU (the interactive data definition utility), which provides you the support to define interactively the characteristics of data and the contents of files stored on your system. IDDU is used to define disk files which are to be used with DW/36 (Display Write/36) and Query/36; it is used to define communications files, which are used with advanced program-to-program communications (APPC).

You use IDDU to define the contents of fields, records, and files. You also use IDDU to define such characteristics of the fields as how large they are, the type of data they contain, and how fields are arranged in a record format. Once you have defined the files *externally*, DW/36, Query/36, and APPC can use the file description instead of defining the file in a program.

Getting Started with IDDU has more information about IDDU.

Utilities Program Product

The Utilities Program Product offers a variety of useful functions for creating and maintaining the parts of applications that you work with day in and day out. It consists of:

- Source entry utility (SEU): for creating and maintaining procedures and source programs.

- Screen design aid (SDA): for creating and maintaining displays and menus.

- Data file utility (DFU): for creating and maintaining simple data entry programs, file update programs, file inquiry programs, and report printing programs. DFU is a way for you to interactively create and maintain **programs**. Using DFU may be faster than coding a program (in RPG II, COBOL, . . .) to do the same job.

- Work station utility (WSU): for creating data entry programs, file update programs, and file inquiry programs.

Development Support Utility

The Development Support Utility (DSU), like SEU, is used to create and maintain procedures and source programs. Unlike SEU, DSU contains a full screen editor that allows you to edit a full screen of statements in a member rather than edit one statement at a time. DSU can coexist with SEU and requires no conversion of data. The Development Support Utility (DSU) is not a part of the Utilities Program Product.

DSU is easy to use whether you are a new or an experienced user.

Business Graphics Utilities/36

The Business Graphics Utilities/36 (BGU/36) is an interactive utility that allows you to design simple business and scientific graphs quickly. You can also design chart pages containing up to eight graphs on a page.

BGU/36 also supplies a procedure that allows you to design a graph using data in a report that was created by your application.

OFFICE/36 Program Products

The OFFICE/36 program products include the following: DW/36 (DisplayWrite/36), Personal Services/36, and Query/36.

DisplayWrite/36: DW/36 is a word processing program. You can use DW/36 to do the following:

- Create documents (letters, memos, reports, and so on)
- Create help text for application programs

The *Getting Started with DisplayWrite/36* manual has more information about DW/36.

Personal Services/36: The Personal Services/36 program product is a group of programs that provides automated ways to handle everyday office tasks. This includes electronic mail handling, a way of logging and doing searches for hard copy mail, calendar management, directory support, group processing, message handling, and administrative support. The *Getting Started with Personal Services/36* manual has more information about Personal Services/36.

Query/36 Program Product: The Query/36 program product allows you to request a variety of reports based on information in your files, even if you have little or no programming experience.

Query/36 prompts you, through a series of displays, to specify which information you want included in your report, whether you want to print or display the report or store the query data in a disk file, and how you want the report to look.

The data entry facility part of Query/36 allows you to put data into your files.

Once a query is created, you can save it in a library. You can then revise your saved queries, copy them, or delete them from the library.

The *Getting Started with Query/36* manual has more information about Query/36.

PC Support/36 Program Product

The PC Support/36 program product lets you use a personal computer as a work station attached to your System/36. Working from your personal computer, you can take advantage of several System/36 functions. The following are included in PC Support/36:

- *Virtual disk support* lets you use the storage on System/36 as if it were attached to your personal computer.

- *Virtual printer support* lets you use printers attached to System/36 as if they were attached to your personal computer.

- *Transfer facility support* lets you transfer System/36 source members, procedure members, and files to the personal computer; it also lets you transfer personal computer files to System/36.

The *PC Support/36 User's Guide* and the *PC Support/36 Technical Reference* manuals have more information about PC Support/36.

Programming Languages

Although the System Support Program Product, Development Support Utility, and Utilities Program Product offer a variety of programming functions, a programming language provides more flexibility and control.

The programming languages available on the system are:

- Assembler

- BASIC

- COBOL

- FORTRAN IV

- RPG II

When you use a programming language, you control how information appears on the users' displays, how the information appears on printed reports, and what processing the program does.

System Productivity Aids

In addition to the programming tools, several productivity aids are available to you.

You can do much of your work from a display station. The menus, displays, procedures, and commands you see and use have been designed to simplify your work. In addition, the following types of assistance called **help support** are available to make your job easier:

- Help menus lead you to the command or procedure to do the task you have in mind. You start with a menu called MAIN and type in the number of the task you want to do. If you choose not to use these menus, you can always enter command or procedure names directly.

- Help text for menus and displays lets you see explanations at the touch of the Help key.

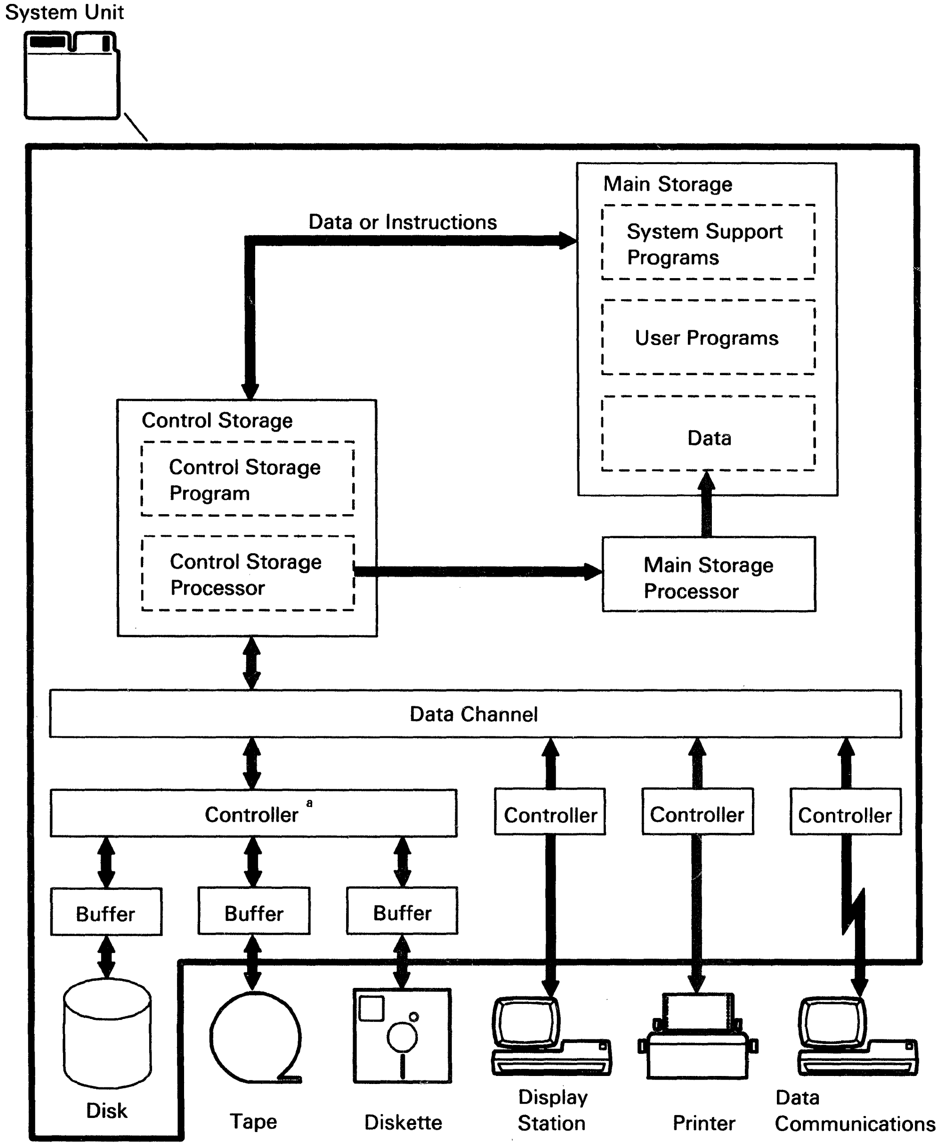
- Help text for commands and parameters lets you see explanations at the touch of the Help key. It is as if someone put part of the *System Reference* manual inside your system.

- Help text for status displays lets you see explanations, again at the touch of the Help key or by command key 8.

Using the help support is quite simple. If you want more information about it, press command key 12 (when you have signed on a display station), and you will see a series of displays that explain how to use help.

System Overview

To design and program applications on the system, you should know some (but, of course, not all) of the details about how the system works. Figure 1-1 shows the important parts of System/36 with the 5360 System Unit (the 5362 System Unit is similar). The sections that follow describe the important parts of the System/36.



[a] This controller is present only when the magnetic tape units are installed. If the tape units are not installed, the disk and diskette buffers are attached directly to the data channel.

S9019001-1

Figure 1-1. System/36 Overview

Figure 1-2 shows the important parts of a System/36 PC.

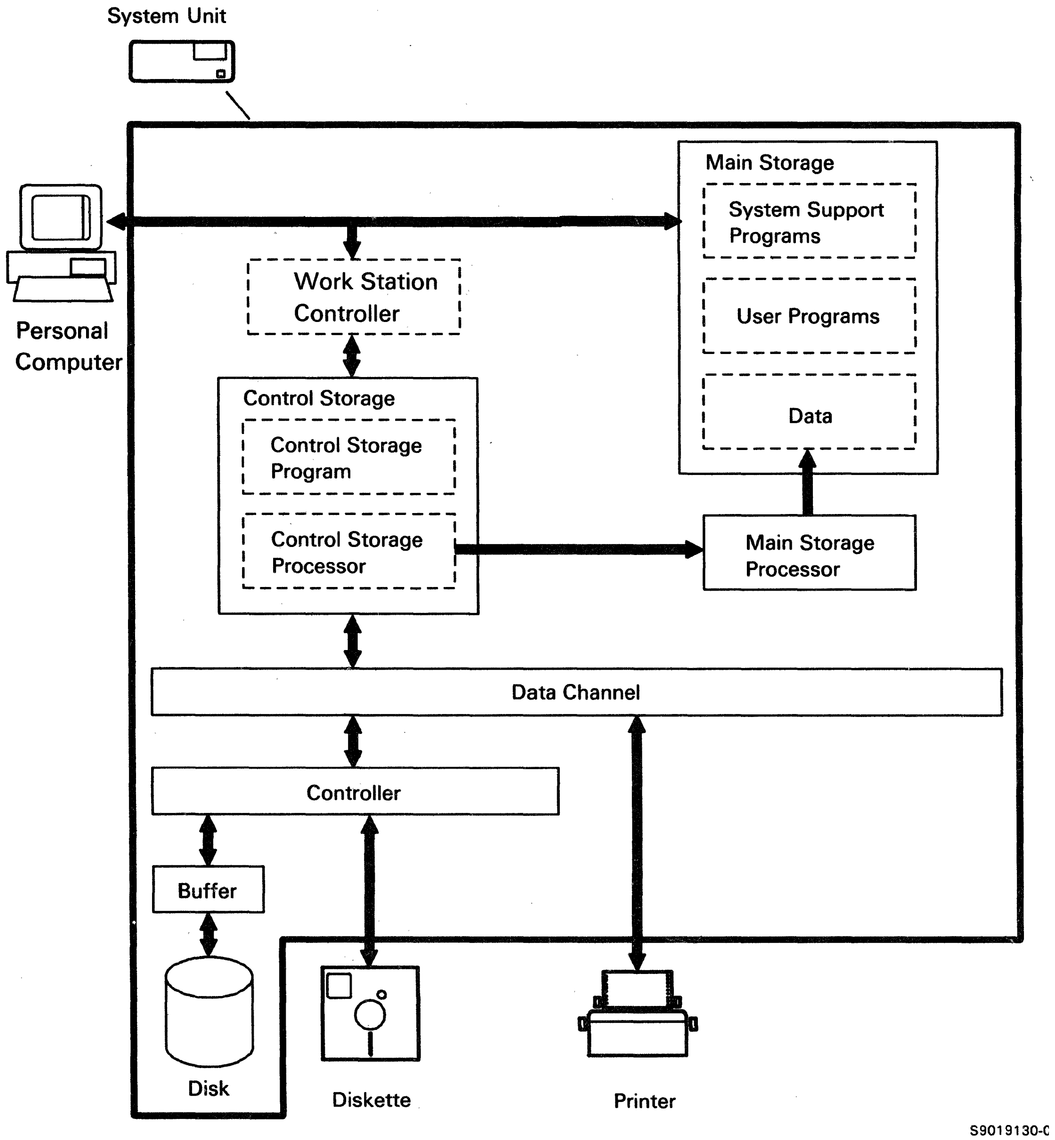
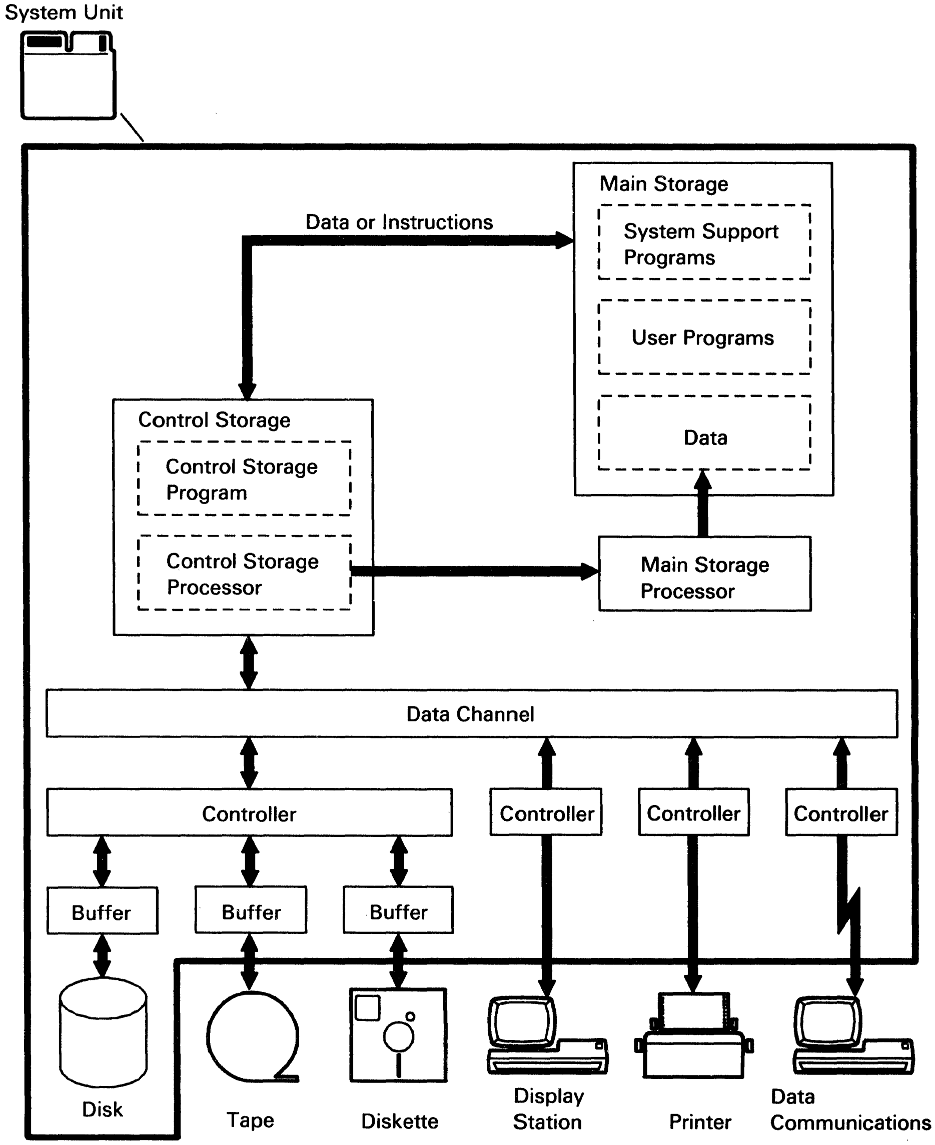


Figure 1-2. System/36 PC Overview

Main Storage

Main storage is the heart of the system—where the action is.



S9019002-1

Main storage has:

- User programs and system programs—while they run
- User program data—as the user programs need it
- Instructions used by the system to process jobs
- Work areas, called buffers, used by the system to process jobs

Chapter 15, "Main Storage," has more information about main storage.

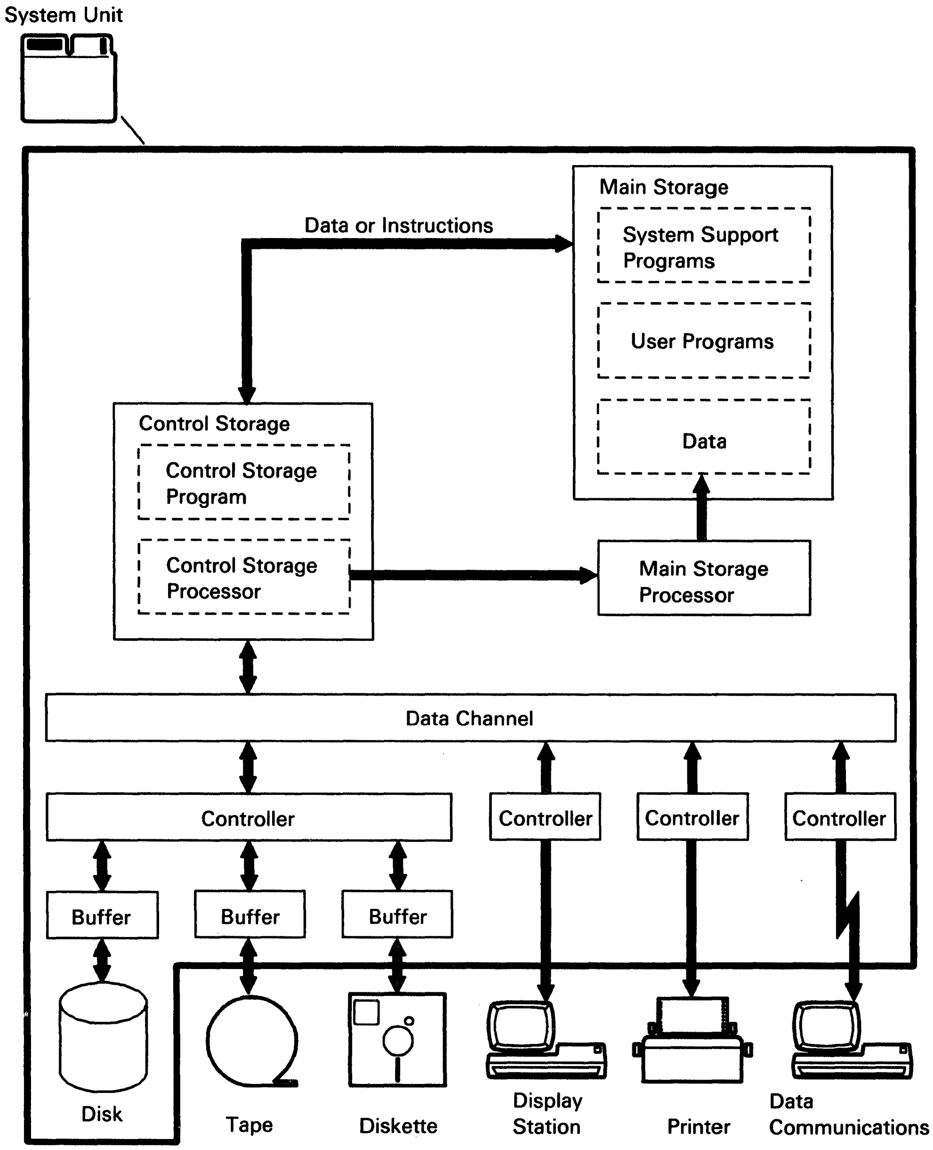
Main Storage Processor

The main storage processor processes application program instructions and system commands. The system is designed so that the main storage processor can devote full time to processing instructions; it does not have to do other things such as get data for programs or control input and output; the control storage processor takes care of those things.

Control Storage

Control storage, as its name implies, controls important system activities. Control storage contains system programs that do input/output and supervisory functions. Because the system has a separate area for these functions, more of main storage can be used by application programs.

For more information about control storage, see "Control Storage Considerations" in Chapter 17, "Jobs and Job Processing."



S9019003-1

Control Storage Processor

Just as main storage has its own processor, so does control storage: the control storage processor. This processor controls input and output and does supervisory functions, which allows the main storage processor to devote its time to processing application program instructions. The control storage processor:

- Allocates main storage for application programs
- Handles data input and output for programs
- Diagnoses and recovers from errors
- Manages system resources
- Controls the main storage processor
- Allows use of the scientific instruction set for BASIC and FORTRAN IV

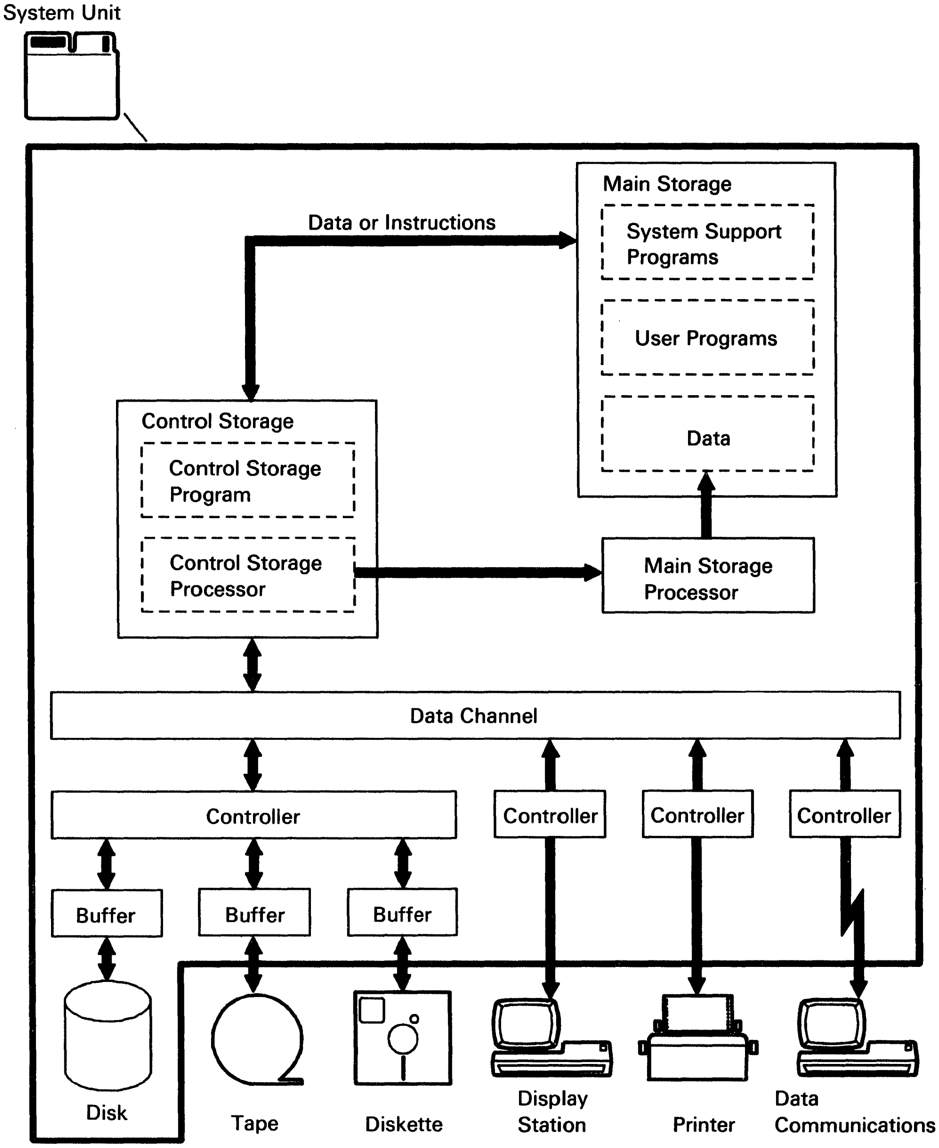
There are several types of control storage processors:

- Control storage processor (CSP) stage 1 (CSP 1.3) has 32K-words and is 1.3 times faster than System/34.
- Control storage processors (CSP) stage 2 and 2.1 (CSP 1.8) have 64K-words and are 1.8 times faster than System/34.
- Control storage processor (CSP) stage 3 (CSP 2.3) has 64K-words and is approximately 2.99 times faster than System/34. It is also 2.3 times faster than stage 1.

  Note: 1 K-word is 2K-bytes long.

| Data Channel

A data channel controls the flow of data between control storage and the input and output devices. The data channel is like a pipeline between control storage and the devices.



S9019004-1

Buffer Areas and Controllers

The data channel uses an input/output controller and work areas called buffers to store data that comes from or goes to disk, diskette, or tape. (Other devices don't have buffers.) The buffers allow the devices to be used at the same time without slowing the performance of the system.

The data channel is shared by all the system devices.

Other input/output controllers process data going to or from the other devices used by the system, such as display stations and printers. Because controllers handle input and output requests from the data channel, the control storage processor need not bother with the requests.

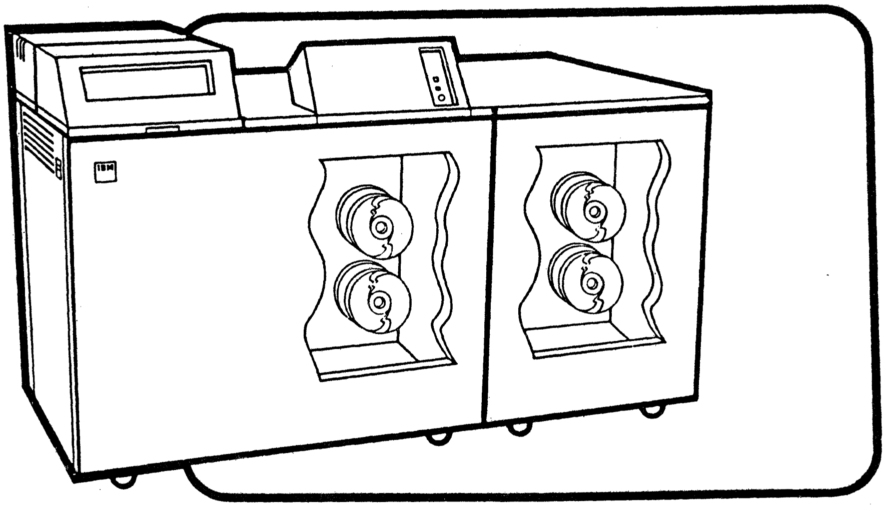
Disk Storage

Main storage and control storage are used primarily to **process** information; they are not intended to **store** all of your information. Nearly all of your programs, files, and libraries are stored on disk (you probably put your infrequently used information on diskette). The system programs, files, and library are also stored on disk.

Several sizes of disk storage are available. The smallest disk holds 30 megabytes of information; the largest disk holds 358 megabytes of information.

In comparison, the smallest main storage size is 128K bytes and the largest size is 7 megabytes, much less storage than is available on disk.

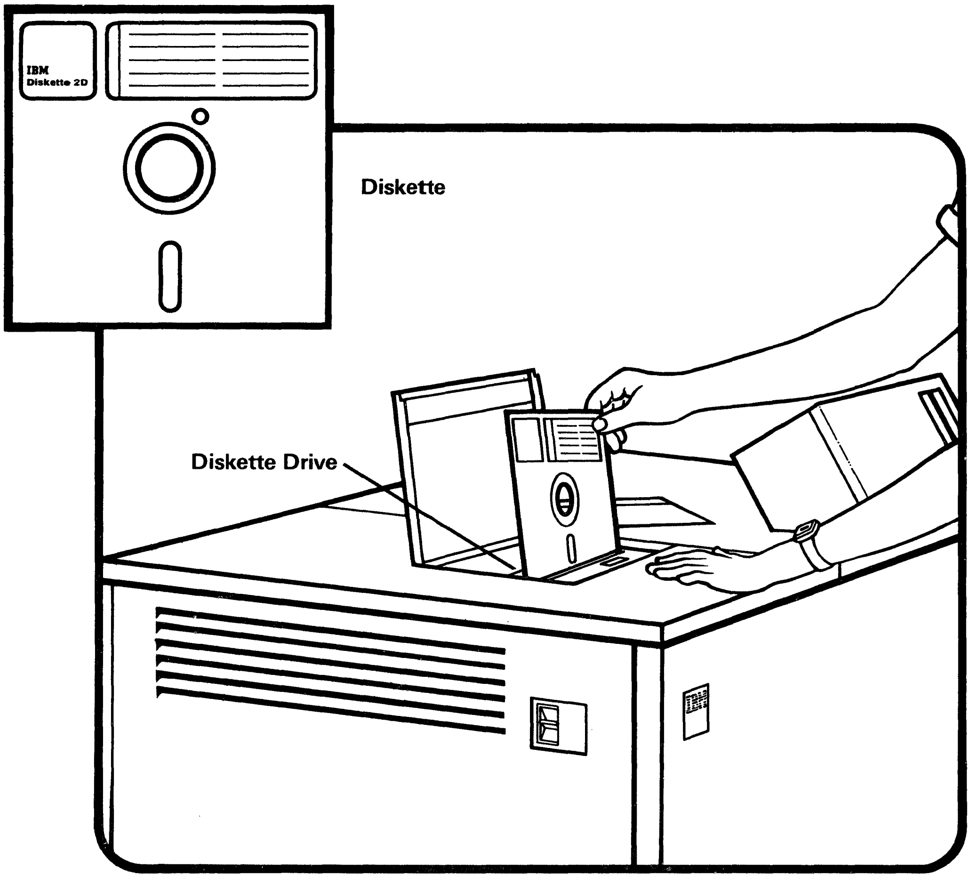
The system unit can have one to four **internal** disks. These disks are not removable. Two disks can be **externally** attached depending on the system unit.

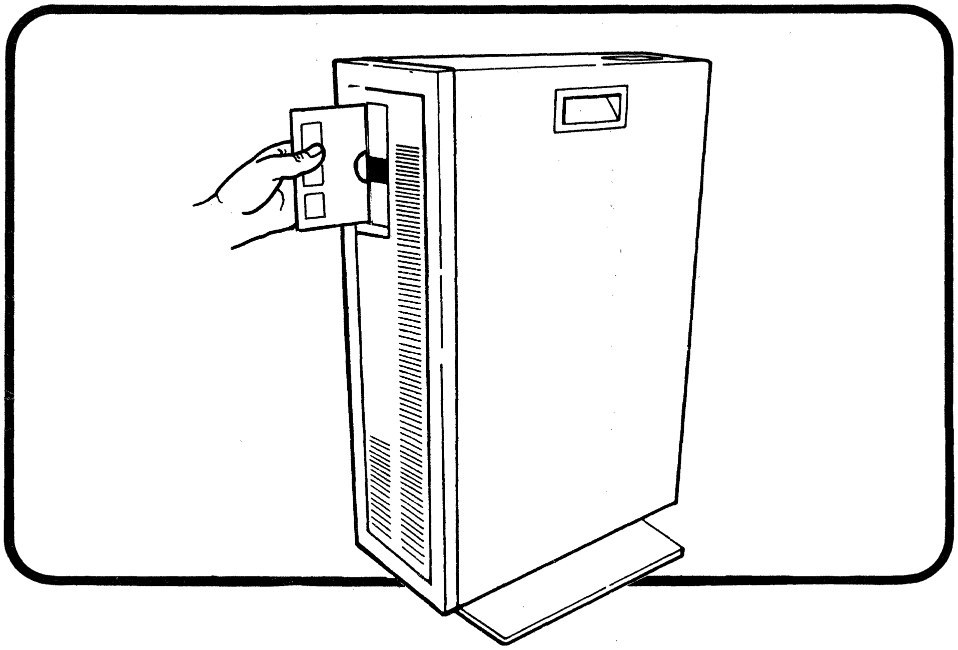


Chapter 4, "Disk Storage," contains more information about disk storage.

Diskette Storage

A diskette is a thin, round, flexible plate that looks very much like a 45 rpm record. To help prevent damage that might result from handling, a protective jacket permanently encloses a diskette.





You can store files and libraries on diskette rather than on disk. Diskette storage is called off-line storage because you can take the diskette out of the system and store it in a safe place. (In order for information to be used by a program, the information must be on **disk**.)

Information is stored on diskette for several good reasons:

- To create a backup copy of the information so that it is in two places at the same time: on disk and on diskette

- To free disk storage for oıher information

- To exchange information between systems

The system unit has a diskette drive, which is the mechanism that reads data from and writes data to diskette. The system has either a diskette drive that can hold one diskette, or a diskette magazine drive that can hold three individual diskettes as well as two diskette magazines.

A diskette magazine is a container that holds up to 10 diskettes. Diskette magazines are handy for saving and restoring files and libraries that won't fit on a single diskette.

Diskettes have several formats, types, and storage capacities.
Chapter 5, "Diskette Storage," has more information about them.

Tape Storage

You can also use magnetic tape on reel or cartridge, depending on your system unit, to store your files and libraries.

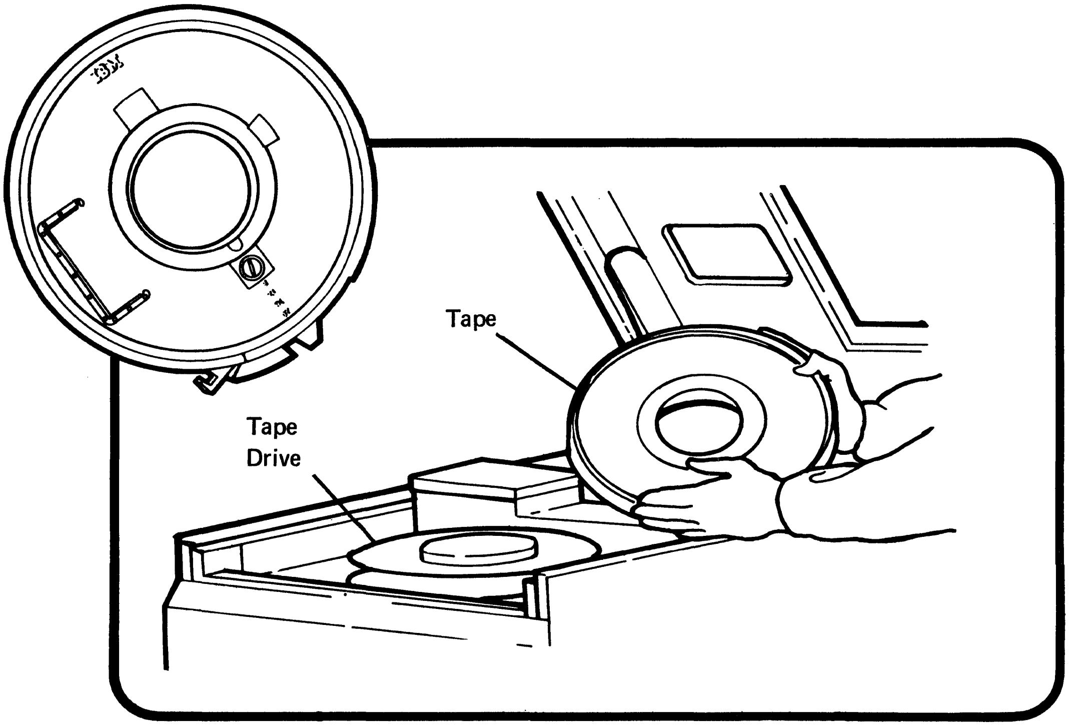


Figure 1-3. Tape on Reel

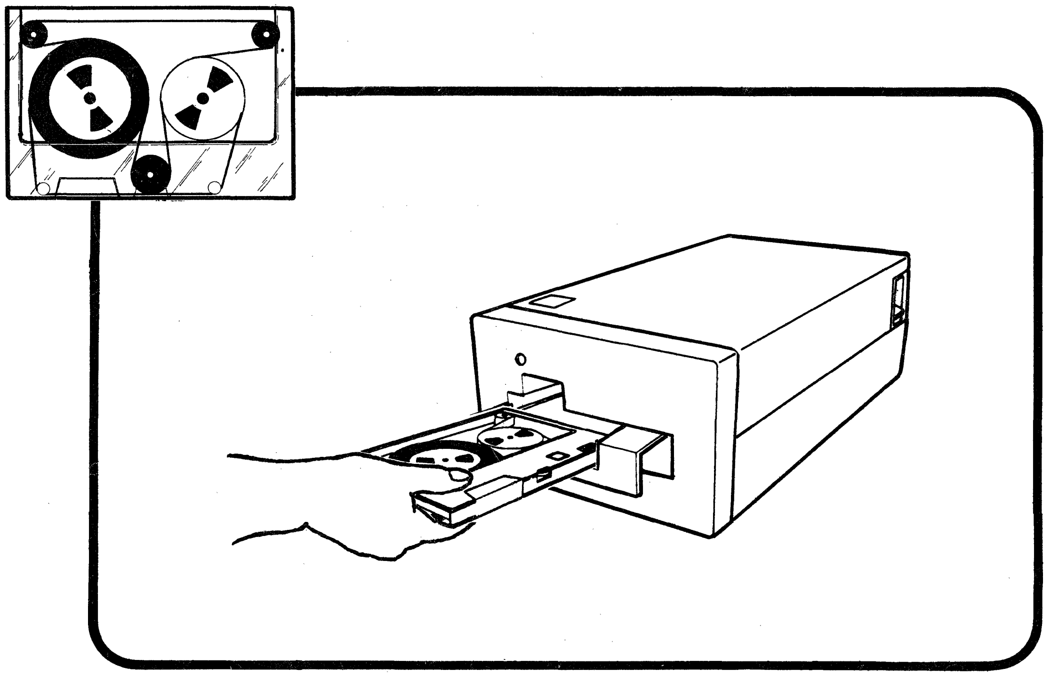


Figure 1-4. Tape on Cartridge

Tapes are handy for saving and restoring files and libraries because:

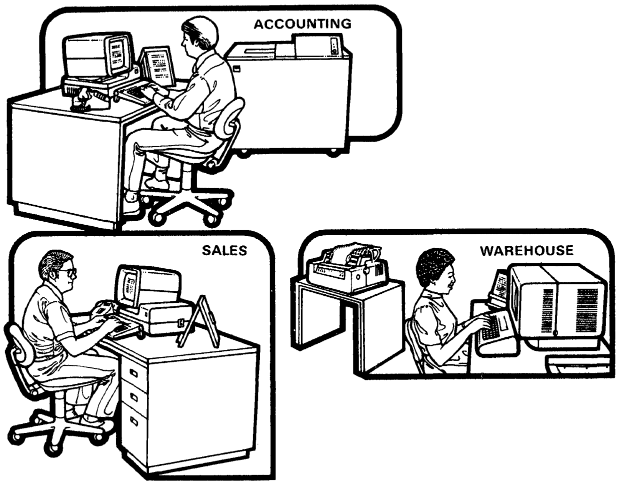
- Large amounts of information can be read from or written to tape faster than it can for diskette.

- More information can be stored on a tape than on a diskette.

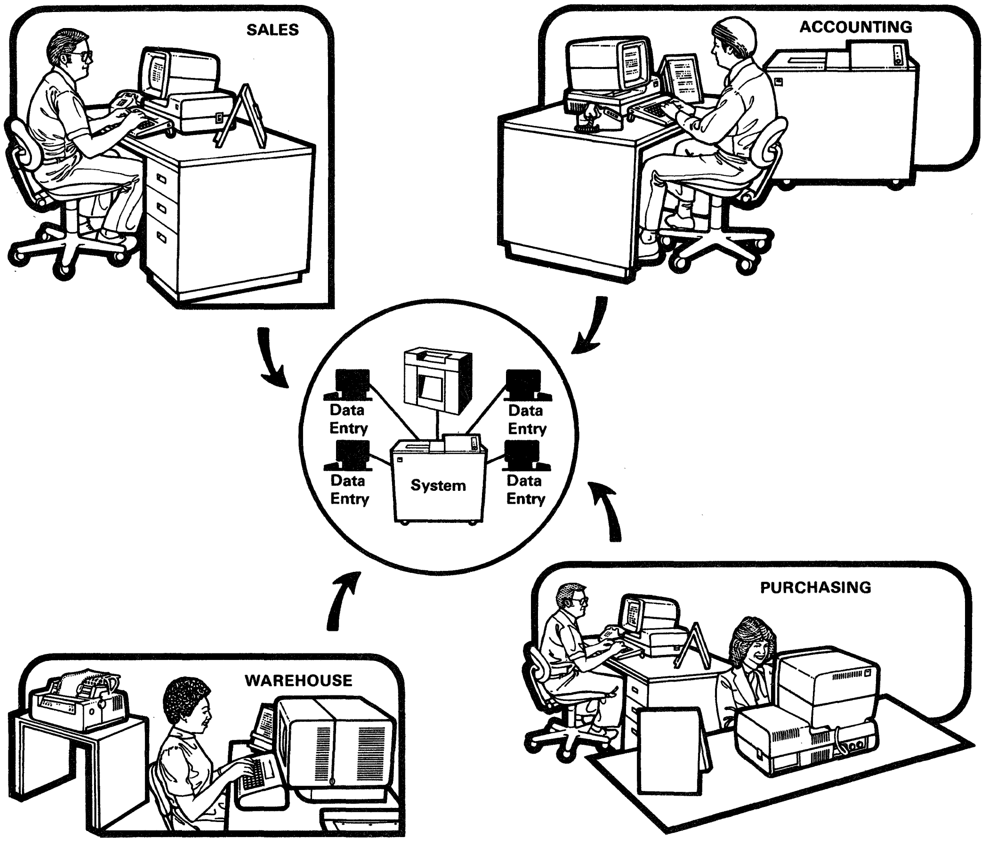
Tapes have several formats and storage capacities. Chapter 6, "Magnetic Tape Storage," has more information about them.

Display Stations

Display stations are extremely important devices on the system because they bring the computer to the users. Each person using a display station can feel that he has the complete system to himself.



Display stations allow the system to operate in an interactive environment. This environment puts the **power** of the computer on the desks of the users. Many users (but only those you want) have simultaneous access to current information, and data is entered by the people who use it. Users in this environment enter data before or while a program runs, request to see the latest information in the files, start jobs, and receive the output.



The interactive environment makes your job extremely important because you design the displays so that the workers can easily use them, design the files so that users can have simultaneous access to the information, and design the programs so that several people can use them simultaneously.

Several display stations are available on the system. Each display station can be attached locally or remotely to the system. Local display stations are near the system, within 1524 meters (5000 feet), and cabled into the system unit. Remote display stations can be farther away from the system and are connected to the system unit using a communications line and a controller.

Whether the display station is local or remote is not significant to the user; he sees the same information and uses the display station the same way. For you, however, there are programming considerations for remote displays. These considerations are described in Chapter 13, "Displays."

When your system was configured:

- One display station was designated as the system console, from which an operator controls the entire operation of the system.
- One or more of the display stations may have been designated as subconsoles, which means that they can control (for example, start and stop) one or more printers.
- One or more display stations may have been designated as command display stations, from which users (including programmers) do their work. Rather than control the entire system, each user controls his own work.
- One or more display stations may have been designated as data display stations, which are always under program control.

From the programmer's view, a display station can request (start) a job or can be acquired by a job that has already been started. A **requesting display station** starts an interactive program. The display station may be used later for data input and output. An **acquired display station** does not start an interactive program; it is acquired by the program after the program begins. The acquired data display station is used for input and output of data.

Work Station Data Management

Work station data management is a system function that allows programs (and, therefore, display station users) to use display stations as they would any input/output device. Work station data management allows your programs to treat a display station as any other file; the display station becomes another way to read data into the system and display output from the system.

The programs you code that use display stations are called interactive programs. Work station data management communicates with your interactive programs by using display formats, which define what information should be displayed or read from the display station. You design the displays (what the users see on their display stations), create the display formats, then design and code your programs to use the display formats. When your interactive program runs, work station data management brings your data and display formats together, showing your information to your users. Chapter 13, "Displays," provides more information about how you design and code displays. It also provides more details about work station data management.

Data Communications

Using data communications, System/36 can send and receive information from a wide range of devices and systems. System/36 can act as a host system to remote work stations, act as a secondary station to a remote host system, or communicate with another system as a peer. Programming support for System/36 communications consists of the following:

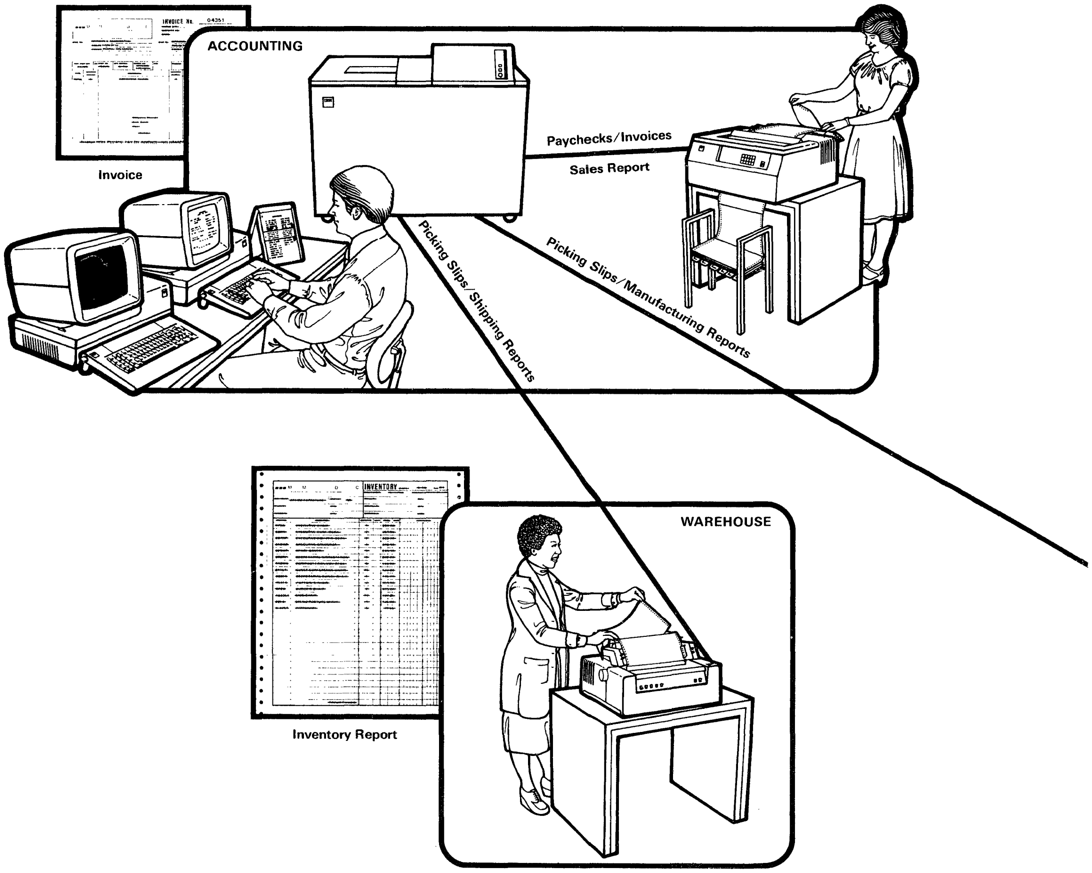
- Base SSP
  - Remote work station support (RWS)
  - Batch binary synchronous communications
  - 3270 remote attachment
- Communications feature
  - Base support for all communications features
  - Autocall support
  - X.25 support
  - Basic conversation support for advanced program-to-program communications (APPC)
  - Asynchronous communications feature
  - Intra
- SSP-ICF Finance
- APPC, BSCEL, CCP, and Peer Subsystems
- Multiple Session Remote Job Entry (MSRJE)
- CICS, IMS, SNUF Subsystems
- 3270 Device Emulation
- Distributed Data Management (DDM)
- Communications and Systems Management (C & SM)
  - Alert support
  - Distributed Host Command Facility (DHCF)
  - Distributed Systems Node Executive (DSNX)

- S/36-S/38 Display Station Pass-Through
- SNA/Advanced Peer-to-Peer Networking (SNA/APPN)
- IBM Token-Ring Network

Information about using communications can be found in the manuals for the individual features. These manuals are listed in the *Guide to Publication* or *Guide to Optional Information*

Printers

Like display stations, printers can be installed in the users' departments. Frequently, a display station and printer are side by side so that the printed information is readily available to the users.



Several printers are available on the system. They provide an assortment of speeds, character sets and other options. Like display stations, printers can be attached directly or remotely to the system depending on your system unit.

Printer Data Management

Just as work station data management is the system function that allows your programs to use display stations, printer data management is the system function that allows your programs to use printers.

When producing reports, your programs create a print file that contains print records. Printer data management ensures that the information is printed. Chapter 3, "Printed Output," contains more information about printer data management.

Spooling

Spooling is a system function that saves printer output on disk for later printing. Even though printers can print at very high speeds, the processing unit can work much faster than a printer. If the processing unit could not start another job until a printer finished the output for the current job, the system would run very inefficiently.

The system can save output from one or more programs on a queue on disk (the queue is called a spool file) while the printer is printing other output. The system then processes the next job while the printer works at its own speed, taking the first job's output from the queue and printing it, then printing the next job's output, and so on down the spool file.

The spooling function is requested when the system is configured. Thereafter, spooling occurs automatically. Chapter 3, "Printed Output," has more information about spooling.

Security

Security is the protection of data, system operation, and system devices from accidental or intentional damage or exposure. What follows is an overview of the two types of security: physical and data.

Physical Security

Physical security involves protecting devices against damage and protecting the entire system from being used by people who are not supposed to use it. Ways of ensuring physical security are:

- Put the system unit in a locked room.

- Use the keylock on the system control panel.

- Put a keylock feature on the display stations.

- Copy programs and data onto diskettes and put them in a safe place (for example, an off-site location).

- Print listings of programs and files and put them in a safe place.

As a programmer, you would most likely be involved with the last two items in the list.

Data Security

Ways of protecting data in the system are:

- Password security. A user must enter an identifier and a password to sign on a display station.

- Badge security (display stations must be configured with badge readers). A user must enter an identifier, a password, and move a badge through the badge reader.

- Menu security. The user is restricted to one menu (a default menu that appears when the user signs on) or to other menus that the user can display from the default menu.

- Resource security. Files, libraries, folders, subdirectories, folder members, and groups are used only by those who are supposed to use them.

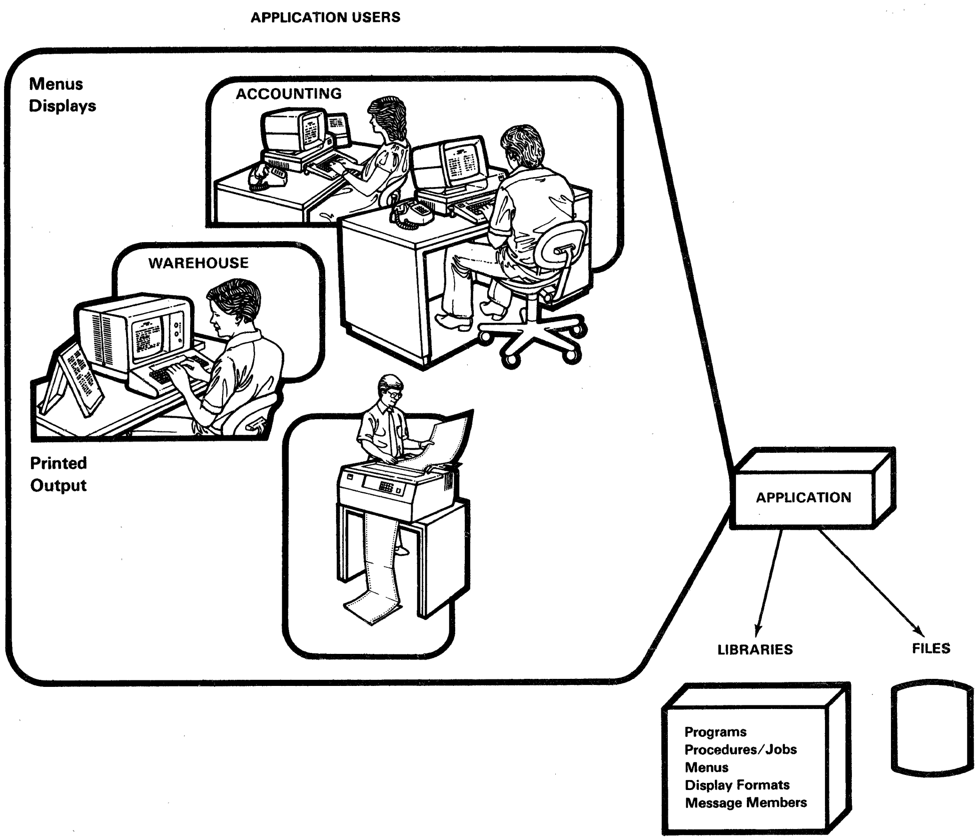
- Communications security. A remote location must identify itself with a remote location name and location password before it can run programs on your system.

As a programmer, you would most likely be involved with menu security and resource security.

Application Overview

As defined at the beginning of this chapter, an application is a group of programs that apply to a particular business function, like accounts receivable or inventory control.

Although there are many different types of applications, most of them are made up of the components shown in the following diagram. You design and create these components so that the application users get the results they want from the system.



The sections that follow present an overview of each component.

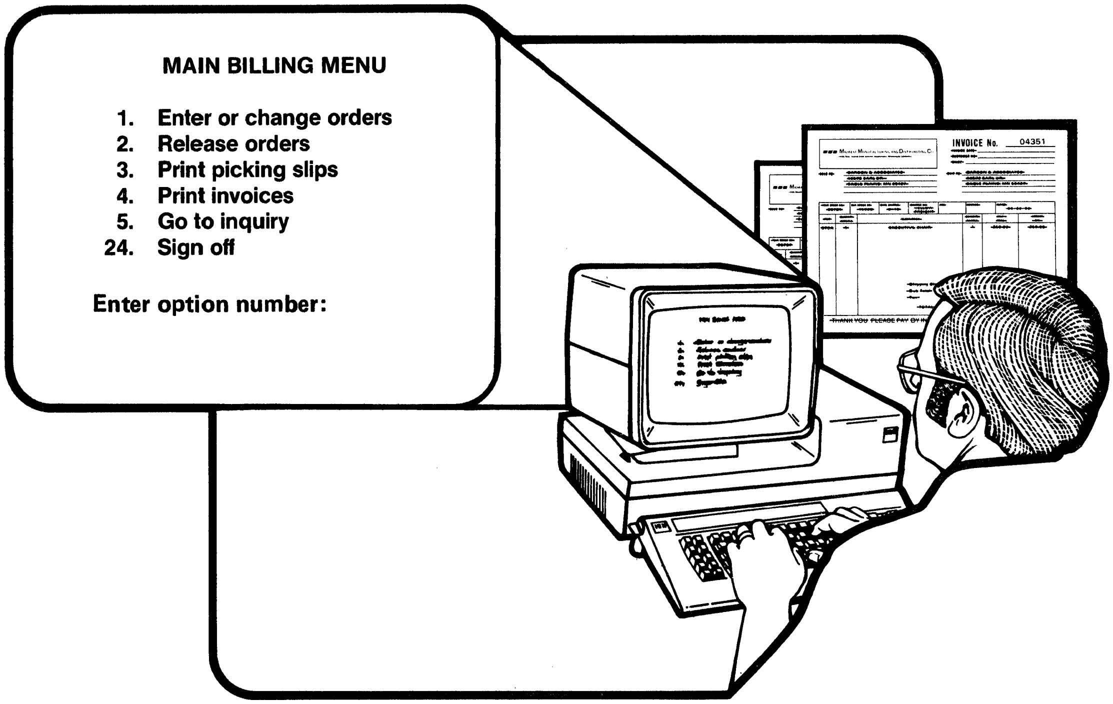
Printed Output

Printed output consists of the reports and listings that the users need. Examples of printed output are paychecks, inventory status reports, inventory reorder reports, picking lists, and customer master file lists.

Chapter 3, "Printed Output," has information about designing and programming printed output.

Menus

A menu is a displayed list of items from which an application user can make a selection. For example, here is a menu a user might see to begin a billing application:



MAIN BILLING MENU

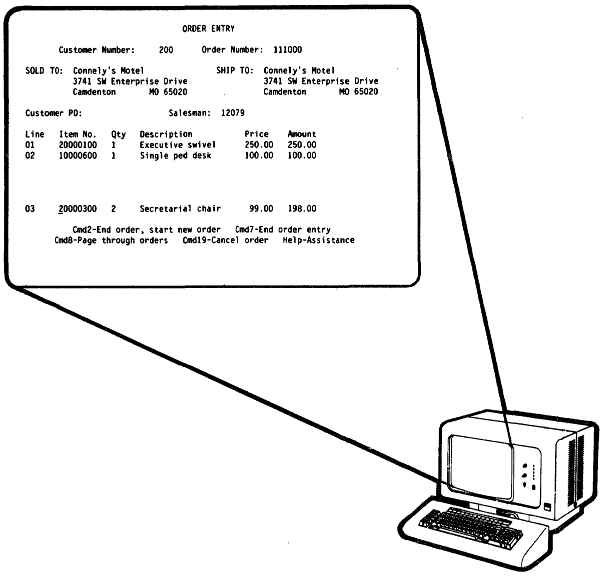
1. Enter or change orders
2. Release orders
3. Print picking slips
4. Print invoices
5. Go to inquiry
24. Sign off

Enter option number:

Menus can simplify work for the applications users if you design the menus with the users and their jobs in mind. Chapter 12, "Menus and Menu Design," has more information about designing and creating menus.

Application Displays

Application displays are *working* displays from which users enter, request, and update information. Menus usually lead to application displays. For example, the following is a display on which a user entered a customer's order:



```
                              ORDER ENTRY

       Customer Number:     200      Order Number:  111000

SOLD TO:  Connely's Motel                  SHIP TO:  Connely's Motel
          3741 SW Enterprise Drive                   3741 SW Enterprise Drive
          Camdenton        MO 65020                  Camdenton        MO 65020

Customer PO:                     Salesman:  12079

Line   Item No.   Qty   Description           Price    Amount
01     20000100   1     Executive swivel      250.00   250.00
02     10000600   1     Single ped desk       100.00   100.00



03     20000300   2     Secretarial chair      99.00   198.00

          Cmd2-End order, start new order   Cmd7-End order entry
       Cmd8-Page through orders   Cmd19-Cancel order   Help-Assistance
```

Like menus, application displays can simplify things if you design them with the users and their jobs in mind. Chapter 13, "Displays," has information about designing and creating displays.

Programs

You can code programs in Assembler, BASIC, COBOL, FORTRAN IV, RPG II and WSU; or you can create programs with DFU. Regardless of the language or utility you choose, programs do the work that the users request. Providing all the functions of an application may take dozens, even hundreds, of programs.

Applications usually have a mixture of interactive and batch programs. Interactive programs communicate with one or more display stations; batch programs do not have this interaction. Interactive programs are generally used for data entry, inquiry, and file update. Batch programs are generally used for printing reports and applying a batch of transactions to a master file.

Chapter 16, "Programs," has information about designing and coding programs.

Files

A file is a set of related records treated as a unit. Three main types of business files are transaction, master, and history.

Transaction files are rather temporary in the sense that they do not remain unchanged for very long. They contain data to be transferred to other files or used by other processes. A file holding the day's orders is a transaction file; once the data from it is processed, the orders are probably transferred to another file.

Master files are more permanent; the records within them contain frequently referenced information that reflects current status. A customer master file, for example, might contain the name, address, telephone number, and account number of each customer you do business with. An inventory master file might contain prices and descriptions of items for sale.

History files are almost never referenced, but may be retained for a period of time for legal or security reasons. A file containing all orders received over the past three years filed in chronological order would be a history file. The transaction file of daily orders would be added to the history file after the orders had been processed.

Deciding what files you need, what information they should contain, how they should be organized, and how they should be processed are some of the design decisions you make.

Chapter 7, "Designing Records," describes how to design records.

Chapter 8, "Files," has the information to help you make the best decisions.

Libraries

A library is a named area on disk that can contain programs and related information (not files). Libraries can contain the elements of an application. For example, all the menus, displays, programs, procedures, and message members for an accounts receivable application could be in one library.

In addition to grouping components of an application, you can use separate libraries for:

- A particular user
- A particular display station

The system can contain the following types of libraries:

- The system library (named #LIBRARY). This library contains most of the IBM-supplied programming support.
- Program product libraries. These libraries contain the IBM-supplied programming support for the Utilities Program Product, Development Support Utility, the OFFICE/36 Program Products, the PC Support/36 Program Product, and the programming languages.
- Your user libraries. These are the libraries that you and others create. These libraries contain your programs, menus, displays, procedures, and message members.

Four types of members can be in a library:

- Source members contain (1) the program statements that are used by a language compiler or the BASIC interpreter and (2) statements used by the system to create display formats, menus, message members, and help text.
- Procedure members contain the statements necessary to run a program or a group of programs.
- Load members contain information in a form that the system can use directly. Examples of load members are compiled programs, display formats, menus, and message members.
- Subroutine members are members that must be link-edited (using a system program called the overlay linkage editor) before the system can run them. Link-edit means to combine a program with one or more subroutines to create a single load member. BASIC and DFU programs can be stored as subroutine members as well.

Chapter 9, "Libraries," has more information about creating, maintaining, and using libraries.

Folders

A folder is a named area on disk that can contain members created by IDDU, DW/36, Personal Services/36, and PC Support/36.

You can group folders in various ways, and each folder can contain multiple members of the same type. Folders are often grouped in the following ways:

- By user (each user automatically is assigned his own folder)
- By projects, using resource security to allow only certain users to access them

Some of the types of folders you might use on your system are as follows:

- Document folders created by DW/36 or PC Support/36
- Data dictionaries created by IDDU
- Mail folders created by Personal Services/36
- Mail log folders created by Personal Services/36

Some of the member types kept in folders are as follows:

- Text in document folders
- Letters in mail folders
- Field, format, and file definitions in data dictionary folders

Chapter 10, "Folders" has more information about creating, maintaining, and using folders.

Procedures, Jobs, and Job Steps

A procedure is a collection of statements that cause one or more programs to run. Procedures eliminate the need to enter frequently used statements each time they are required. The procedure statements are in a library member called a procedure member.

The purpose of a procedure is to run a job. Jobs can have one or more job steps; a job step is a unit of work done by one program. A job step begins with the LOAD OCL statement and ends with the RUN OCL statement.

To run a procedure, you can enter a procedure command, which is simply the name of the procedure member in the library. Procedure commands are usually entered with information that tells the procedure what to do. The information is in the form of parameters.

Your application users generally don't know (and don't need to know) the procedure's name or parameters because you will provide menus for them. The user selects an option from a menu, and the system runs the appropriate procedure: the one you told it to run when you created the menu. The procedure, in turn, loads and runs the program to do the task the application user specified.

Many procedures are supplied as part of the SSP and the Utilities Program Product. For example, the SSP procedures allow you to create data files, create libraries, and copy data files; the Utilities Program Product procedures and the Development Support Utility procedures (DSU), allow you to create and change library members.

In addition to using the SSP and Utilities Program Product procedures and DSU procedures, you can design and create your own procedures.

Chapter 17, "Jobs and Job Processing," in this manual has information about jobs and job steps.

Chapter 18, "Procedures," in this manual and a chapter in the *System Reference* manual have information to help you design and create procedures.

Messages and Message Members

The system uses messages to communicate with you. Messages can inform you, request information from you, or notify you of errors that have occurred. Your programs also use messages to communicate with the application users.

A message member is a place in a library in which you can store messages. You need not use message members, but their advantages are fairly obvious; your messages are in **one** place, and can be referenced by a number called a message identification code (MIC) from **several** places: from programs, procedures, and display formats.

Message members ensure that your application users will not see the same message worded in several ways, and they save you time because you can code a message number rather than the message text.

Chapter 14, "Messages and Message Members," has information about messages and message members.

Chapter Review

This chapter provided (1) an introduction to programming the system, (2) an overview of the system's components, and (3) an overview of application components.

If you want a description of the application design process, you can read Chapter 2, "Application Design Steps." Otherwise, you are ready to use Chapters 3 through 19 to design and develop applications.

Chapter 2. Application Design Steps

Chapter 1 provided an overview of the system and application components. This chapter introduces the steps that programmers generally follow to design their applications. It is a good chapter to read if you are about to design your first application. Although it does not describe a particular design method, structured or otherwise, the chapter should help you:

- Realize the importance of application design

- Define, organize, and document your application design work

As defined in Chapter 1, an application is a group of programs (and associated displays, files, procedures, libraries, folders, data dictionaries, and menus) that allow users to do a particular business function, such as accounts receivable or order entry.

Application design is the integration of the application components, personnel, and system devices to accomplish a business function. Application design begins with a request for system support; it ends with installed, operational programs that meet the users' needs. When you design an application, you coordinate many interrelated activities rather than just one activity.

Application Design Steps

The application design steps are shown in Figure 2-1. As you can see, you **and the application users** participate throughout the project.

This chapter does not provide a lot of details about any one step. Its purpose is to help you structure your application design work by describing what an application designer does and the general order in which he does his work. The steps, of course, can overlap. For example, you could design a display while you design the program that uses the display.

Step	Application Users	Application Designer
Step1. Definition	Explain current methods. Provide complete examples of input and output for current methods. Explain sources and destinations of information. Describe desired methods for existing and additional application functions.	Identifies what users do. Defines input and output. Identifies what the application should do. Defines application functions. Makes preliminary schedule. Reviews work with users. Documents the definition.
Step 2. General Design	Help define the application functions. Review the design.	Specifies the input, processing, and output for the application in general.
Step 3. Detailed Design	Review and agree upon displays and reports.	Designs printed output, files, and displays.
Step 4. Program Design	Help define the sequence and frequency of functions. Identify deadlines and constraints.	Designs programs and then codes them.
Step 5. Testing	Help define test cases, test the application, and check the results.	Codes test cases. Ensures that each program and all application functions work.
Step 6. Conversion and Installation	Help convert information.	Trains application users. Ensures all existing information is converted to a form usable by the new application. Loads the application components onto the system.
Step 7. Operation	Use the application.	Turns the application and documentation over to the users.

Figure 2-1. Application Design Steps

Objectives of Application Design

Why should you take time to follow the application design steps? Because if you don't, here are some things that can go wrong:

- The application might not do what the users want.
- It might make the users drastically change the way (sequence in which) they do their work.
- It might be difficult to test, debug, and install.
- It might be difficult to maintain.
- As the company grows (bigger inventory, more customers), it might not be able to handle the growth.
- Items might be overlooked.
- Data might not be available when it is needed.

As you can see, poor planning can be costly. So resist the urge to say, "Let's start coding. We can work out problems as they occur."

Planning an application is similar to planning your home. Before workers pour cement and begin building, you consult an architect who:

1. Identifies what you want.
2. Sketches plans based on what you want.
3. Prepares a general schedule of what happens and when.
4. Draws a blueprint for your review.
5. Writes initial specifications for your review.
6. Provides a final blueprint, a detailed specification, and a final schedule to the contractor and workers.

The planning time is difficult to endure because you are eager to get started, but it pays off. There should be few mistakes and delays. You should have a home that is completed on time, costs what you expect, and suits your needs.

As an application designer, your objectives are much the same as an architect's. You prepare and document a plan for developing an application that will be:

- What the users want
- Completed on time with the expected effort
- Easy to use
- Easy to maintain

The application design steps described in this chapter can help you meet these objectives.

Step 1. Definition

The definition step is one of the most important steps in application design. The careful work you do during definition will pay off in the remaining steps. For this step you:

- Identify what the users do
- Define the information they process
- Identify what the application should do
- Define the major application functions
- Make a preliminary schedule
- Review your work with the users
- Document your work

Identify What the Users Do

Talking to the application users is the best way to identify and understand what the users do. To help you organize your notes and observations, you might make a work flow diagram. On it, you can indicate:

- What jobs are done
- Who does them
- In what order they do them
- What main steps are involved in each job
- Who provides or collects input
- What happens to the input
- Who receives output
- What management decisions are based on the output
- When are the busiest times during the day, month, and year
- When are the slack times
- What deadlines must be met daily, weekly, monthly, and yearly
- How people would change their jobs to do them better

Figure 2-2 shows an example.

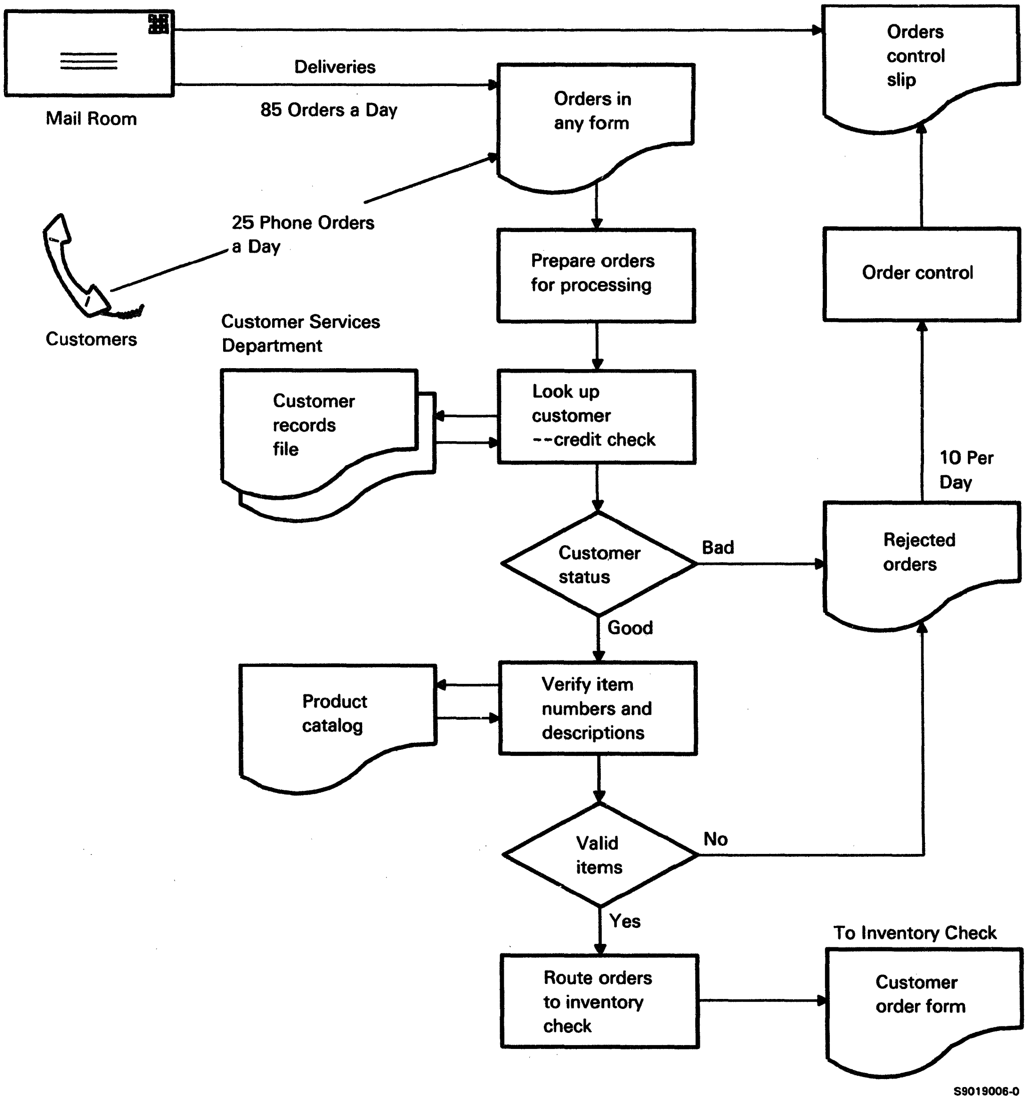


Figure 2-2. Work Flow Diagram

The work flow diagram is like an architect's initial blueprint. It is your interpretation and documentation of the current methods of doing work. It is a good document on which to center your discussions with the application users.

Notice that the work flow diagram can also show information flow. You can indicate what the input is, who (person or department) provides it, what the output is, and who receives it.

In addition to analyzing and documenting the work flow, you should collect documents, forms, written procedures, training manuals, and other items that workers use to do their jobs.

The work flow diagram and the material you collect will later help you:

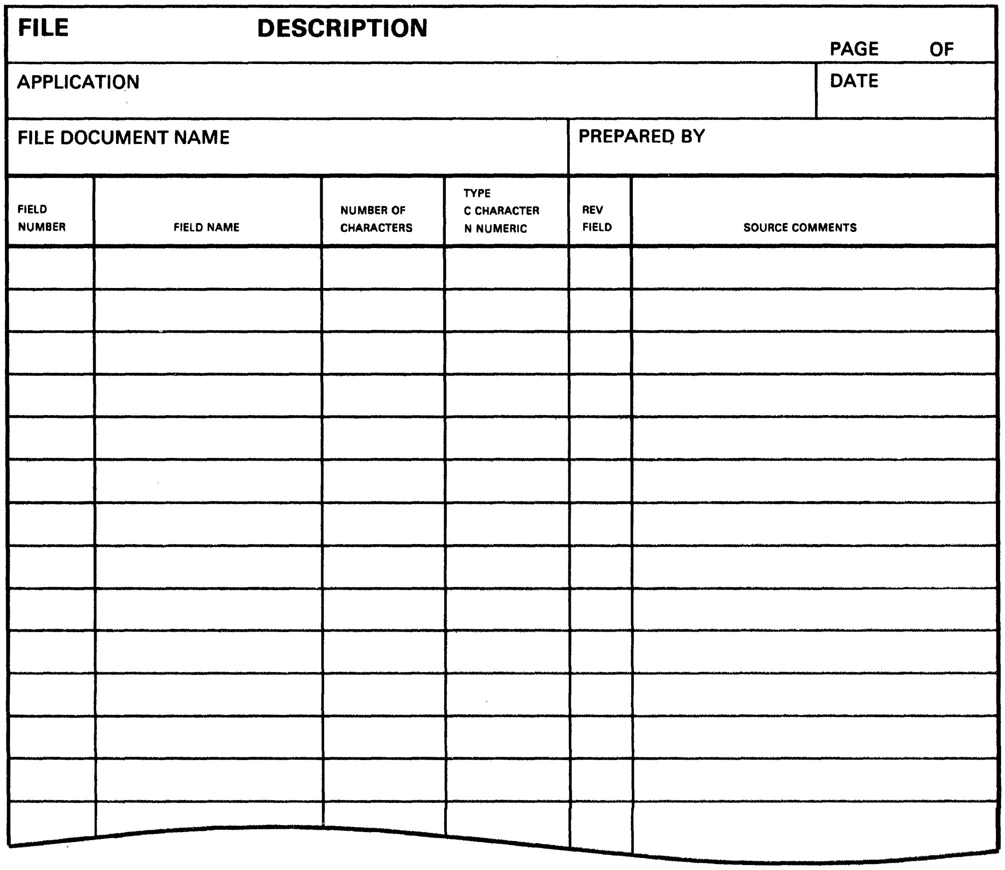
- Determine your application's input, processing, and output
- Develop documentation to help people use the application

Define the Information Processed

To design the application's files, displays, and printed output, you need to know the characteristics of the data that is processed by the present methods. For example, as you answer the following questions, you can list the important pieces of information (and the characteristics of each) that are used as input to the application or produced by the application. (Figure 2-3 shows an example.) These items will be the fields in your files, on your displays, and on your reports.

- What information is processed or produced?
  - A customer order?
  - A monthly sales report?
  - A list of items that need reordering?
- What items of information are processed or produced?
  - Customer name?
  - Customer number?
  - Item number?
  - Item description?
- What attributes does each item have?
  - All numbers?
  - All letters?
  - How many?
- What ways do people find, sort, or report information?
  - By customer number?
  - By item number?
  - By sales region?
  - By department?
- How much information is processed?
  - One hundred orders a day?
  - One thousand orders a day?

- How much information is stored?
  - One hundred items?
  - One thousand items?
- What form is the information in?
  - A telephone call?
  - A mailed order?
  - A price list in a filing cabinet?
- What information can be seen or handled only by certain people?
- Who maintains information?



FILE DESCRIPTION

PAGE OF

APPLICATION

DATE

FILE DOCUMENT NAME

PREPARED BY

FIELD NUMBER	FIELD NAME	NUMBER OF CHARACTERS	TYPE C CHARACTER N NUMERIC	REV FIELD	SOURCE COMMENTS

S9019007

Figure 2-3. Information Characteristics

Identify What the Application Should Do

Talking to the application users is the best way to identify and understand what the application should do. This seems obvious, but many programmers **assume** what users want and discover too late that the application does not do what the users expected it to do.

Keep in mind that the new application need not do the functions in the same way as the current methods do them. For example, you might replace some printed reports with displayed information.

Your discussions can be very informal, but it is a good idea to take notes for future reference.

Define the Application Functions

After you understand what the users want, what they do, and what information they process, you should be able to list the main functions that your application provides. Items you would list on the application menus are probably the main application functions. For example:

- Display customer information
- Maintain (update, add, or remove) customer information
- Print customer information
- Print mailing labels
- Save and restore customer information

When you list the functions, you might see some things that only certain people should be allowed to do. If so, you should consider designing the application to take advantage of the security features the system has. The *System Security Guide* has information about security.

Make a Preliminary Schedule

At this time, your schedule should specify:

- Dates on which you plan to begin and end each of the application design steps. You can schedule activities to avoid problems (such as converting to the new application during an extremely busy time of the year).

- Dates when users have time to discuss what they do and what the application should do.

- Dates when you demonstrate and review application functions with the users.

- Dates when you and management review the application design progress.

Review Your Work with the Users

You should discuss the definition with those who requested your application and those who will use it. This review can ensure that you correctly analyzed what they want.

Document Your Work

As you can see, the definition step is a fact-finding step. You organize your findings, present them to others, get agreement on application functions, and accurately estimate and schedule the application work.

Before you begin the next step, general design, be sure you have documented your work. Here are some suggestions for things to document:

- Work flow, which can also show information flow.

- What information is processed and the characteristics of each piece of information.

- Major application functions.

- A schedule.

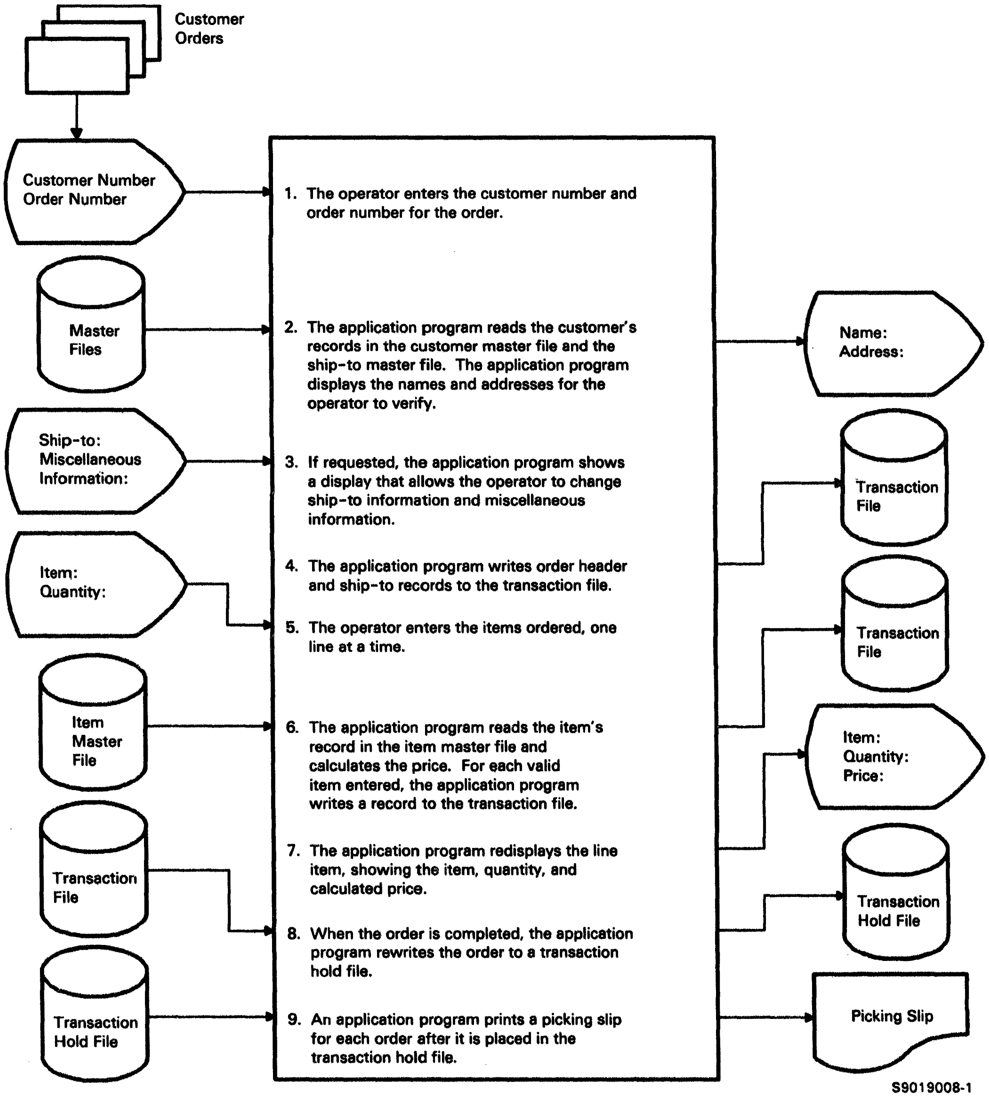
- Important assumptions you have made: those that are key to you doing what you promised. For example, you may have assumed that you and another programmer will work full-time on the project.

Step 2. General Design

In the general design step, you shift your attention from what the present methods are to how your application can do the work on the system. In the general design step, you determine:

- What input the application requires
- What output the application produces
- What processing the application does

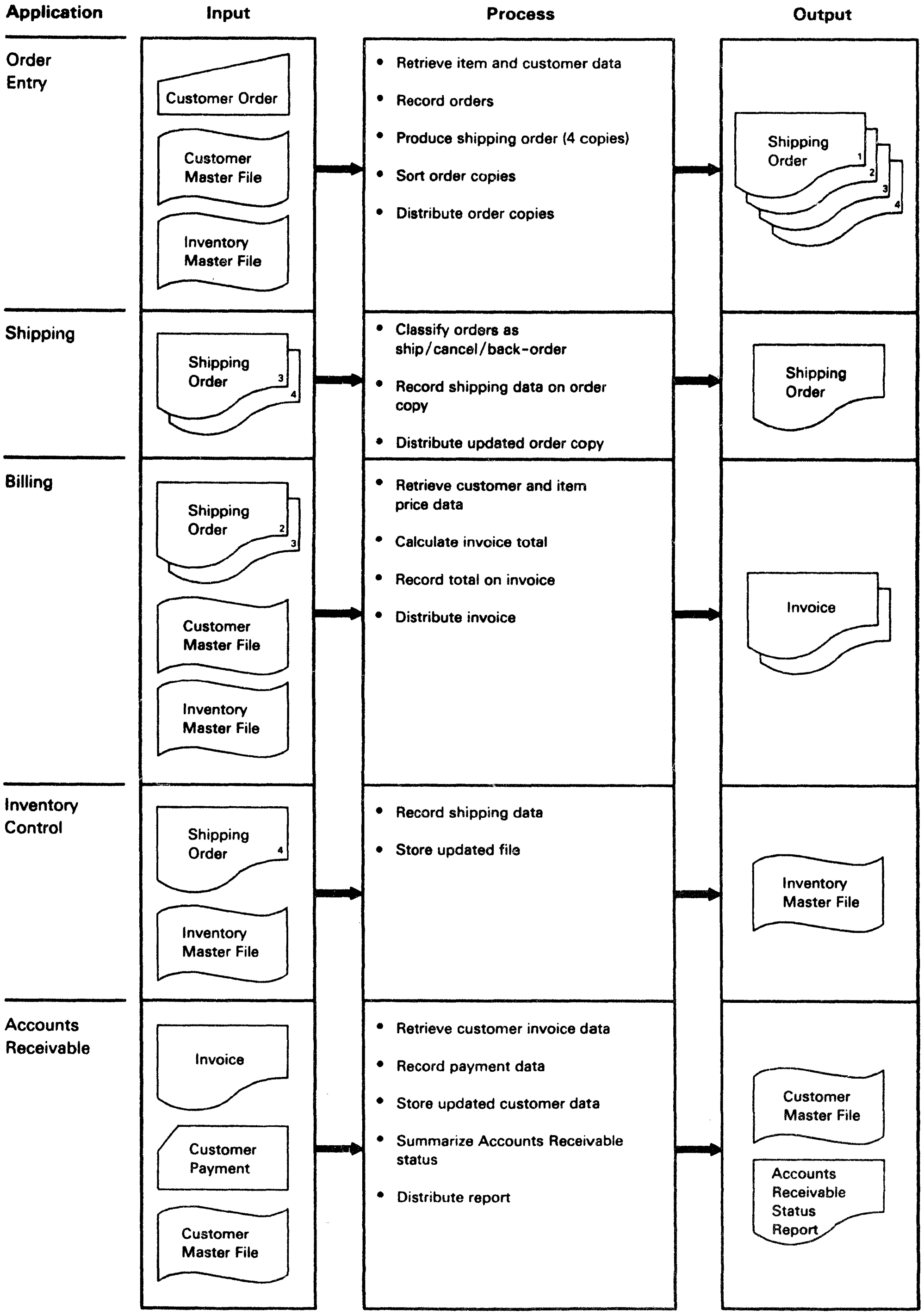
For example, the following is a general design diagram for an application that does order entry:



1. The operator enters the customer number and order number for the order.
2. The application program reads the customer's records in the customer master file and the ship-to master file. The application program displays the names and addresses for the operator to verify.
3. If requested, the application program shows a display that allows the operator to change ship-to information and miscellaneous information.
4. The application program writes order header and ship-to records to the transaction file.
5. The operator enters the items ordered, one line at a time.
6. The application program reads the item's record in the item master file and calculates the price. For each valid item entered, the application program writes a record to the transaction file.
7. The application program redisplays the line item, showing the item, quantity, and calculated price.
8. When the order is completed, the application program rewrites the order to a transaction hold file.
9. An application program prints a picking slip for each order after it is placed in the transaction hold file.

Notice that the information is general. You have not yet designed individual displays, files, or reports, and you have not decided how many programs you need. You are trying to get a big picture of what your application needs, does, and produces.

The output from an application is often the input to another application. In the general design step, you identify the links between applications, as the following example shows:



Application	Input	Process	Output
Order Entry	Customer Order Customer Master File Inventory Master File	• Retrieve item and customer data • Record orders • Produce shipping order (4 copies) • Sort order copies • Distribute order copies	Shipping Order 1 2 3 4
Shipping	Shipping Order 3 4	• Classify orders as ship/cancel/back-order • Record shipping data on order copy • Distribute updated order copy	Shipping Order
Billing	Shipping Order 2 3 Customer Master File Inventory Master File	• Retrieve customer and item price data • Calculate invoice total • Record total on invoice • Distribute invoice	Invoice
Inventory Control	Shipping Order 4 Inventory Master File	• Record shipping data • Store updated file	Inventory Master File
Accounts Receivable	Invoice Customer Payment Customer Master File	• Retrieve customer invoice data • Record payment data • Store updated customer data • Summarize Accounts Receivable status • Distribute report	Customer Master File Accounts Receivable Status Report

Step 3. Detailed Design

In the detailed design step, you (1) design each report, file, and display you identified in the general design step and (2) design application controls.

Designing Printed Output

Chapter 3, "Printed Output," has guidelines for designing your printed output.

Designing Files and Records

Chapter 7, "Designing Records," and Chapter 8, "Files," have guidelines for designing files and records.

Designing Displays

Chapter 13, "Displays," has guidelines for designing displays.

Designing Application Controls

Application controls are the plans you put in place to ensure that input data is correct, that your programs process data correctly, and that the results are correct.

Examples of Input Controls

You might verify input by having someone check all input documents before entering them into the system. For example, someone could verify that all required information has been filled out, that valid ranges of values have been specified, and that valid customer numbers have been used.

Also, you can record what input was entered into the system so that loss of an entire input document cannoι go undetected.

Examples of Processing Controls

A program could count the number of input documents it processes. You could compare this total with the count that was made when the documents were entered into the system.

A program can check whether or not records have been sorted into the correct sequence.

An audit trail can be used to record what work was done on the system and the order in which it was done. The audit trail can be generated manually by having users fill out a log of what they did and when they did it. You can also program the application so it generates an audit trail.

Examples of Output Controls

Output controls report the results of the processing done by the application. These controls can be especially effective when you combine them with input controls. For example, a program can compare input totals with output totals or compare the number of records processed with the number of documents entered to verify that output is correct.

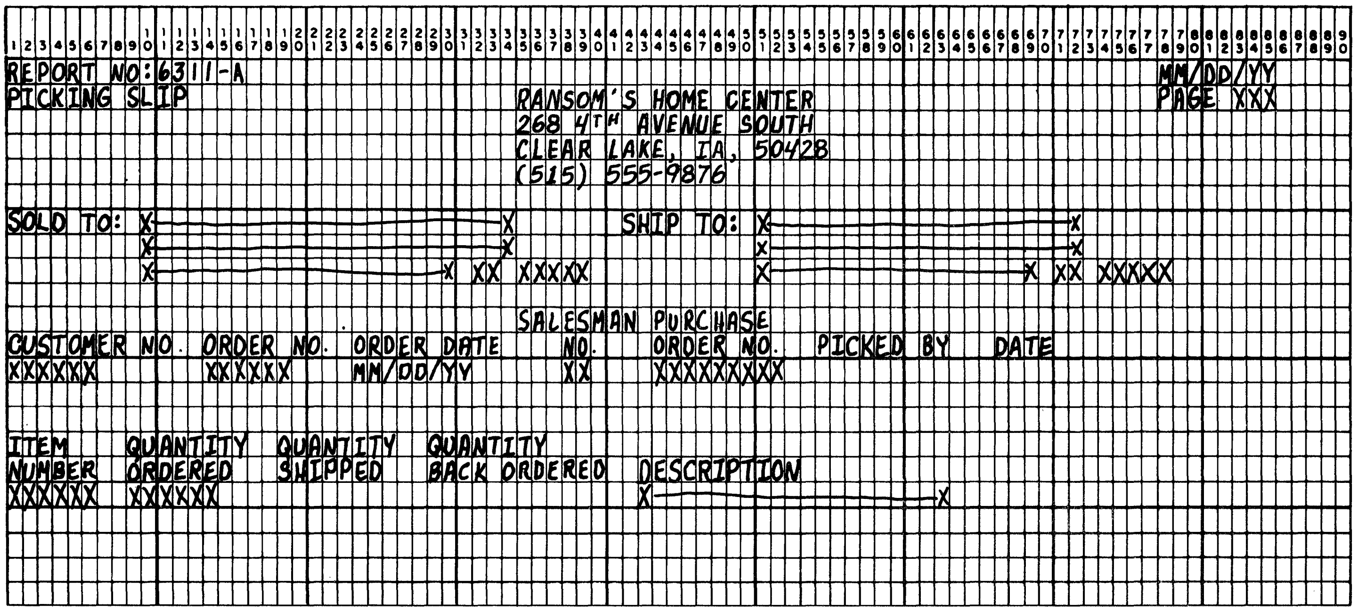
Documenting the Detailed Design

Your documentation for the detailed design step can include:

- The forms on which you design your printed output. On each form, you can note which program prints the output, who should receive it, and whether it requires special forms. Figure 2-4 shows a form used to design a picking slip.

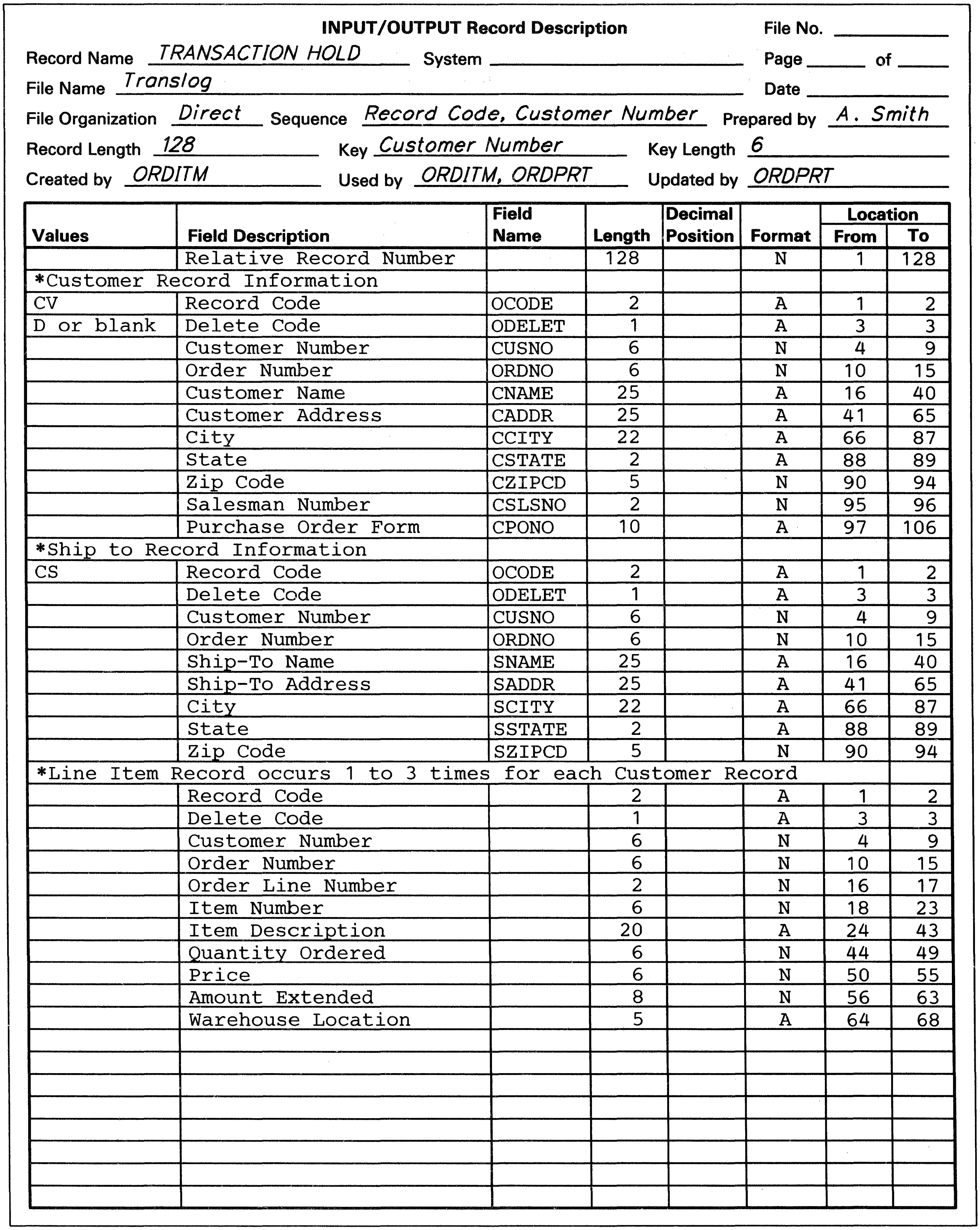
- The forms on which you design your files and records. On each form, you can note the programs that use the file and how each program uses the file. Figure 2-5 shows a form used to design a transaction hold file.

- The forms on which you design your displays and menus. On each form, you can note the display name or menu name and what programs use them. Figure 2-6 shows a form used to design a menu used to maintain a customer file. You can also save the output that prints when you create the displays.



```
REPORT NO:6311-A                                                             MM/DD/YY
PICKING SLIP                      RANSOM'S HOME CENTER                       PAGE XXX
                                  268 4TH AVENUE SOUTH
                                  CLEAR LAKE, IA, 50428
                                  (515) 555-9876

SOLD TO: X-----------------------X       SHIP TO: X--------------------X
         X-----------------------X                X--------------------X
         X-------------------X XX XXXXX           X-----------------X XX XXXXX

                                  SALESMAN PURCHASE
CUSTOMER NO. ORDER NO. ORDER DATE    NO.   ORDER NO.  PICKED BY  DATE
XXXXXX       XXXXXX    MM/DD/YY      XX    XXXXXXXXXX


ITEM    QUANTITY  QUANTITY  QUANTITY
NUMBER  ORDERED   SHIPPED   BACK ORDERED  DESCRIPTION
XXXXXX  XXXXXX                            X-------------------X
```

Figure 2-4. Form for Designing Reports



INPUT/OUTPUT Record Description

File No. ____

Record Name TRANSACTION HOLD System ____ Page ____ of ____

File Name Translog Date ____

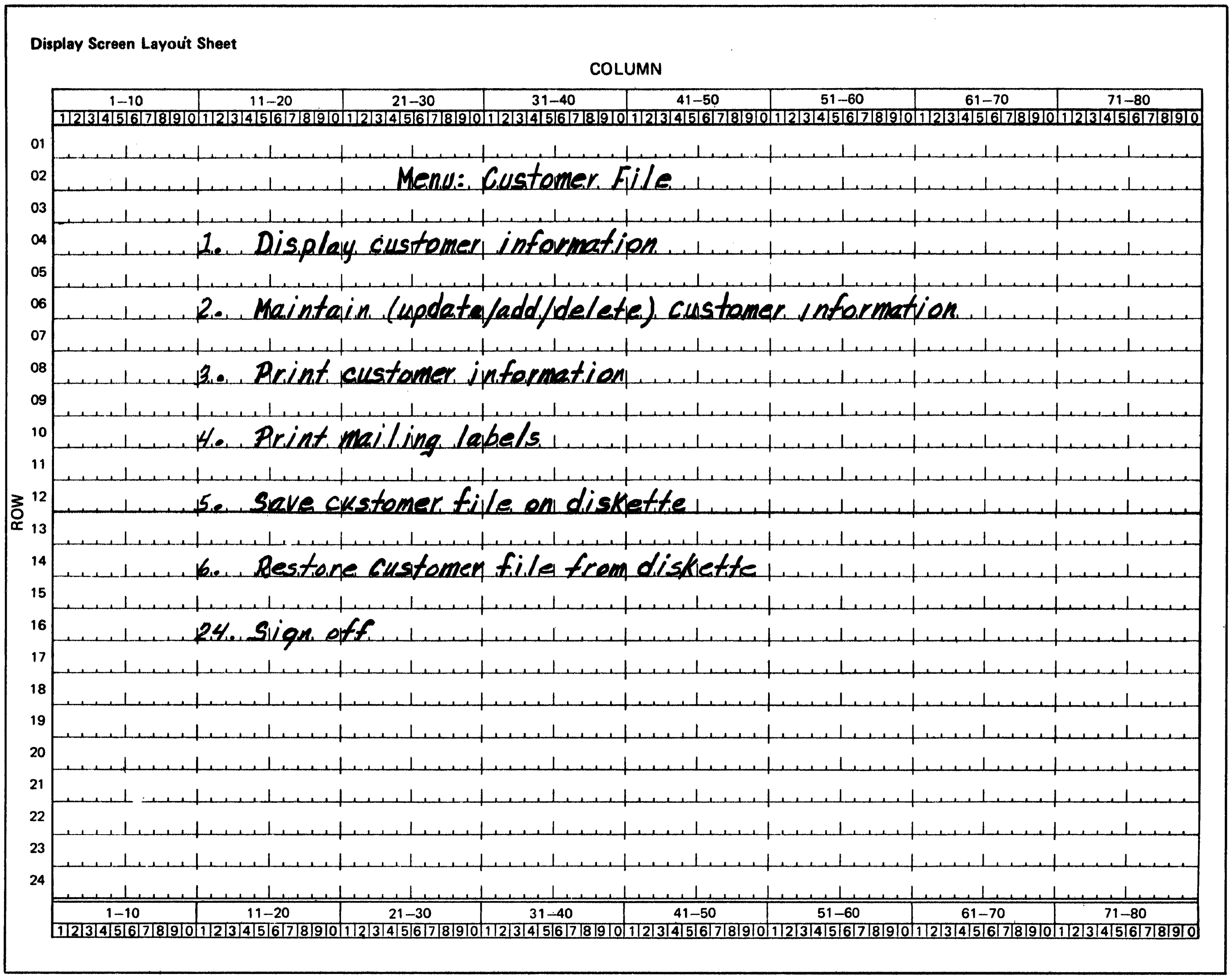
File Organization Direct Sequence Record Code, Customer Number Prepared by A. Smith

Record Length 128 Key Customer Number Key Length 6

Created by ORDITM Used by ORDITM, ORDPRT Updated by ORDPRT

Values	Field Description	Field Name	Length	Decimal Position	Format	Location	
						From	To
	Relative Record Number		128		N	1	128
*Customer Record Information							
CV	Record Code	OCODE	2		A	1	2
D or blank	Delete Code	ODELET	1		A	3	3
	Customer Number	CUSNO	6		N	4	9
	Order Number	ORDNO	6		N	10	15
	Customer Name	CNAME	25		A	16	40
	Customer Address	CADDR	25		A	41	65
	City	CCITY	22		A	66	87
	State	CSTATE	2		A	88	89
	Zip Code	CZIPCD	5		N	90	94
	Salesman Number	CSLSNO	2		N	95	96
	Purchase Order Form	CPONO	10		A	97	106
*Ship to Record Information							
CS	Record Code	OCODE	2		A	1	2
	Delete Code	ODELET	1		A	3	3
	Customer Number	CUSNO	6		N	4	9
	Order Number	ORDNO	6		N	10	15
	Ship-To Name	SNAME	25		A	16	40
	Ship-To Address	SADDR	25		A	41	65
	City	SCITY	22		A	66	87
	State	SSTATE	2		A	88	89
	Zip Code	SZIPCD	5		N	90	94
*Line Item Record occurs 1 to 3 times for each Customer Record							
	Record Code		2		A	1	2
	Delete Code		1		A	3	3
	Customer Number		6		N	4	9
	Order Number		6		N	10	15
	Order Line Number		2		N	16	17
	Item Number		6		N	18	23
	Item Description		20		A	24	43
	Quantity Ordered		6		N	44	49
	Price		6		N	50	55
	Amount Extended		8		N	56	63
	Warehouse Location		5		A	64	68

Figure 2-5. Form for Designing Files



Display Screen Layout Sheet

COLUMN

1–10 | 11–20 | 21–30 | 31–40 | 41–50 | 51–60 | 61–70 | 71–80

ROW

01	
02	Menu: Customer File
03	
04	1. Display customer information
05	
06	2. Maintain (update/add/delete) customer information
07	
08	3. Print customer information
09	
10	4. Print mailing labels
11	
12	5. Save customer file on diskette
13	
14	6. Restore customer file from diskette
15	
16	24. Sign off
17	
18	
19	
20	
21	
22	
23	
24	

1–10 | 11–20 | 21–30 | 31–40 | 41–50 | 51–60 | 61–70 | 71–80

Figure 2-6. Form for Designing Displays and Menus

Step 4. Program Design

In the program design step, you design and code the application programs, jobs, and procedures.

Chapter 16, "Programs," has guidelines for designing programs.

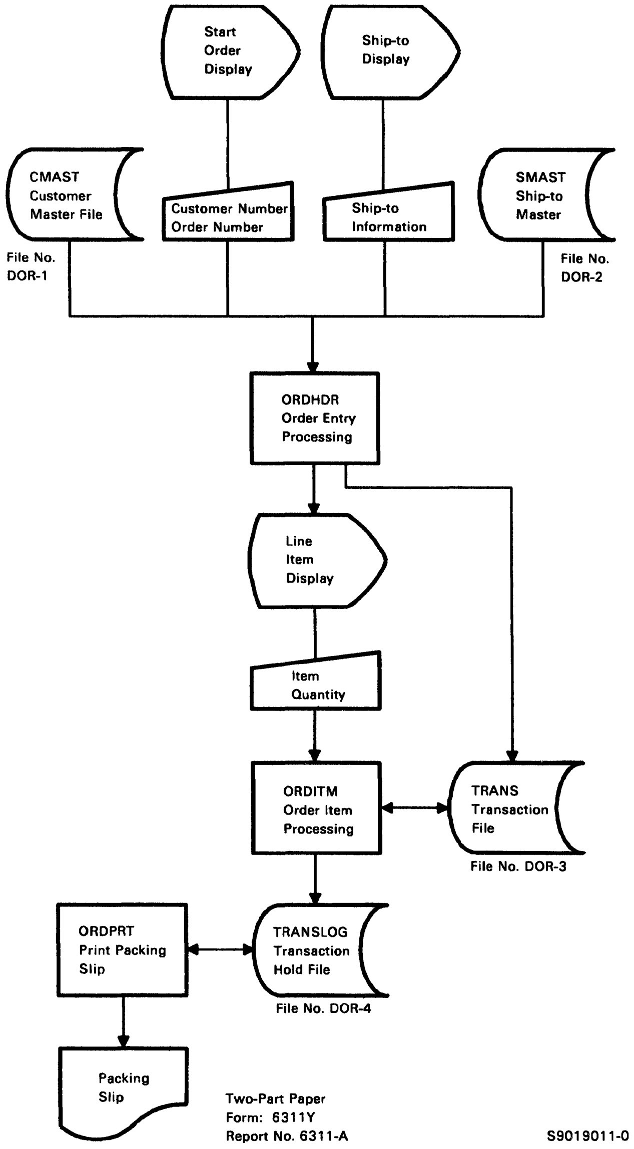
Chapter 17, "Jobs and Job Processing," has guidelines for designing jobs.

Chapter 18, "Procedures," has guidelines for designing procedures.

Documenting Programs

Documentation of your programs can include:

- A definition of the function (such as order entry)
- A definition of the input (such as displays and files)
- A definition of the output (such as displays, files, and reports)
- Logic of the current program (See Figure 2-7 for an example.)
- A history of the revisions that have been made
- People to contact for further information



TITLE ORDER ENTRY APPLICATION

	DATE
AUTHOR	PAGE 1 OF 1

Notes:

Run Frequency: Daily
Volume: Approximately 500 transactions per day

ORDHDR (OR-01)

Run Time: 1 hour

Accepts customer number, order number, and ship-to information, and writes header customer and ship-to records to the transaction (TRANS) file.

ORDITM (OR-02)

Run Time: 30 minutes

Accepts item number and quantity information, and writes an order record to the transaction hold (TRANSLOG) file.

ORDPRT (OR-03)

Run Time: 10 minutes

Generates picking slips, and updates a control record in the TRANSLOG file.

Figure 2-7. Example of Program Logic Documentation

Step 5. Testing

Testing is what you do to ensure that your application:

- Accepts input correctly
- Processes the input correctly
- Produces output correctly
- Is easy to use
- Presents meaningful error messages for the users
- Runs as fast as you expected
- Has acceptable response times for the display stations users

Allow Time for Testing

Be sure to allow plenty of time to test your application. For applications that have several complex programs, you might need more time to test than you do to design and code.

Divide the Test into Manageable Tasks

Whereas you design an application top-down, from general functions to specific programs that do each function, you should test the application from the bottom up. That is, you first test that each program works, then you test that the functions work. Some functions require only one program; therefore, you would be able to test the function and the program at the same time.

Test simple programs within simple functions first. For example, an inquiry program within an inquiry function is usually a good place to start.

Each test should build upon the preceding one. For example, if a program reads three different record types, you should test that the program can read one, then two, then all three record types.

After you test each application function, test the application as a whole in the environments it will be used (for example several people using it at the same time). When you are confident that the entire application works by itself, run it with other applications. These steps can help you find concurrent processing errors such as file sharing problems or incorrect input received from another application.

Let Users Define Test Cases

Testing, like the other steps in application design, needs user participation. Let the users help define the test cases. The users should be able to define typical situations that test each application function.

Use Realistic Test Data

Use realistic test data; for example, use actual customer numbers, item numbers, or orders. Label test data and save it for future reference. You will want to retest the application if you uncover errors or if you change functions after the application is installed and operational.

Test data should include (1) normal and expected conditions and (2) error and unexpected conditions.

Normal and Expected Conditions

Test that the application works when everything goes as it should; for example, without operating errors, without program errors, and with correct input. For example, you might test these order entry functions under normal conditions:

- Open a new account
- Update a new account
- Close an existing account
- Update many existing accounts
- Generate a picking slip report

Error and Unexpected Conditions

To test these conditions use incorrect data or operating procedures. For example:

- Use nonexistent account numbers
- Update a closed account
- Enter data that has incorrect dates or totals
- Use combinations of data that have multiple errors or data that has both correct and incorrect values
- Run a program with wrong input

Use Sufficient Amounts of Test Data

In addition to using realistic test data, you should test that the application can handle the expected volumes of data. You might, for example, test how many orders can be processed in an hour or in a work day.

Document the Test

Keep a record of test data used for input, an explanation of each test, and descriptions of the expected results from each test.

Step 6. Conversion and Installation

In nearly all cases, your application replaces current methods, even though they are partly or entirely manual methods. At some time, the current methods must be replaced by the new application. There are two basic methods of converting to the new application:

- The new application runs **in parallel** with the existing methods. The new application is carefully watched and evaluated. When the new application performs as intended, the old methods are discontinued.

- The new application **immediately replaces** the existing methods as soon as the application is tested.

The parallel method is, of course, more expensive and takes more of the application users' time than immediate replacement because information must be entered twice and results must be compared with one another. But the parallel method may be safer because you have a backup if the new application fails.

Immediate replacement may be the only way to convert your interactive programs or it may be the better way to convert if you have a relatively simple application.

When you use the immediate replacement method, be prepared to handle the transition period when neither the old method nor the new method provides the latest information. This transition time might be only a few hours, but you should plan how to conduct business as usual during it.

Regardless of the method you choose, you need to convert the information in its present form to a form usable by your application. This usually involves creating master files and entering information in them (for example a customer file or an inventory file). The information must also be checked for accuracy and corrected if necessary. Of course, if you have thousands of customers or parts, this initial entry and verification can take quite a while. A schedule is helpful for planning what happens and when during conversion and installation.

You need not convert and install all application functions at the same time. For example, you might install inquiry functions first and install more complex functions such as online file maintenance later.

Step 7. Operation

In the operation step, you (1) plan and provide what users need to run the application and then (2) turn the tested, installed application over to the users.

What do the users need to run your application? They may need some or all of the following:

- On-the-job training.
- Run books (how to start the application, the steps for normal operation, error conditions and messages, how to end the application, how to interpret the output, samples of all forms).
- Instructions for doing their jobs if the application stops working.
- Scheduling instructions. (Should this application be used before or after other applications? In what order should the application functions be run?)
- Backup and recovery instructions (for example, what data to save, when to save it, and how to recover from errors).
- Passwords if password security is active.

The system operator, as well as the application users, might need instructions from you. For example:

- When to start or stop the application
- What to do if application users have problems
- How and when to back up the application programs and data

A reminder: you need not write all of your instructions on paper. The system lets you code help displays, which are online explanations of users' menus and displays. Users can see your instructions at the touch of the Help key. Chapter 12, "Menus and Menu Design," has information about creating help for menus.
Chapter 13, "Displays," has information about creating help for displays.

Chapter Review

Again, the application design steps are:

1. Definition

2. General design

3. Detailed design

4. Program design

5. Testing

6. Conversion and installation

7. Operation

The most important points to remember are:

- Take time to do a thorough definition so that you understand what the users want and what the application should provide.

- Work with the users from the beginning to the end of the project.

- Plan and document each aspect of your application.

Using the Remaining Chapters

You can use the remaining chapters in any order as you need them. Each one has useful information about how the system works, how to design application components, and how to program the system. The chapters sometimes summarize a topic.

Chapter 3. Printed Output

The purpose of this chapter is to:

- Describe ways in which the system prints data
- Describe print spooling, which can be used to efficiently manage the printing of data
- Suggest ways to design printed output
- Provide printer performance considerations
- Provide programming tips and techniques regarding printed output
- Describe merging text and graphics in the printed output

How the System Handles Printer Output

Printed output on the system is done either by printer data management or by the system list function. Printer data management handles most of the printed output. System list mainly handles the output created by the SSP system utilities (for example, the $MAINT utility program).

The following list shows those programs that use printer data management:

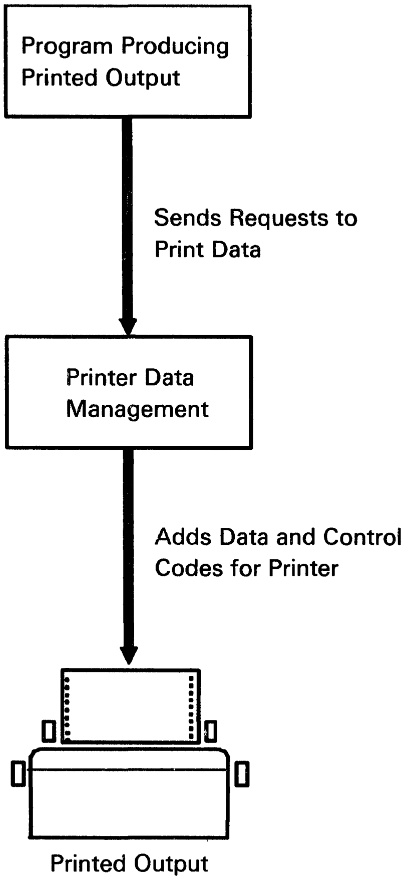
User-written programs
Print key
SSP data communications programs
Sort
Utilities programs (SEU, DFU, SDA, and WSU)
Development Support Utility (DSU)
Business Graphics Utilities/36 (BGU/36)
RPG II compiler
COBOL compiler
FORTRAN compiler
Assembler
BASIC
DW/36 (DisplayWrite/36)
Query/36

The following list shows those procedures and utility programs that use system list:

BLDMENU ($BMENU)
BUILD ($BUILD)
CATALOG ($LABEL)
EDITNRD ($SINR)
FORMAT ($SFGR)
HISTORY ($HIST)
LISTDATA ($COPY)
LISTFILE ($BICR, $COPY, and $MAINT)
LISTLIBR ($MAINT)

Printer Data Management Output

Printer data management handles most of the output from the system. The following diagram shows how printer data management handles printed output.


Program Producing Printed Output
Sends Requests to Print Data
Printer Data Management
Adds Data and Control Codes for Printer
Printed Output

S9019012-0

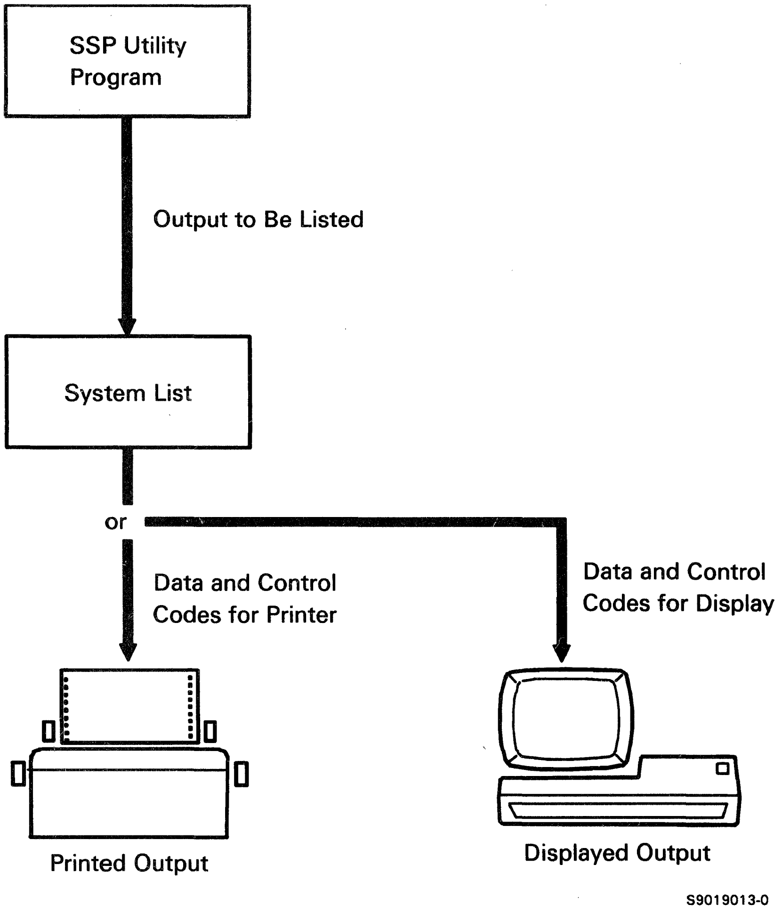
Printer data management takes the print requests from the program, inserts the proper control codes for the printer being used, and sends that information to be printed. This causes your output to be printed with the paging and spacing that your program requested.

The printed output from a program can be sent to a specific printer by using a PRINTER OCL statement. If a PRINTER OCL statement is not specified, printed output is sent to the session printer for the display station. You can use the STATUS SESSION control command to display the ID of your session printer. To change your session printer, you can use the PRINT procedure. This will change the printer only until you sign off the display station.

When you sign on, the session printer is determined by the value that was specified during system configuration. The configured session printer is shown by the STATUS SESSION control command. To change your configured session printer, use the SET procedure. This will change the printer until another SET procedure is run or the system is reconfigured; that is, the change will still be in effect the **next** time you sign on the display station. If you want to permanently change the default printer assignment, use the CNFIGSSP procedure.

System List Output

The output from SSP utility programs is handled by a special system program called **system list**. System list allows output to go to a printer or a display station. The following diagram shows how the system processes system list output.



System list takes the output listing requests from the utility program, determines whether the output is to be printed or displayed, and sends that information to the appropriate device. The printer or display station that is receiving the system list output is called the **system list device**.

When you first sign on, the system list device is the same as the session printer. To display the current system list device, use the STATUS SESSION control command. To change the system list device, use the SYSLIST procedure. This procedure changes the system list device until:

- Another SYSLIST procedure is processed.
- A PRINT procedure or the FORMS OCL statement is processed.
- You sign off the display station.

The *System Reference* manual has more information about using the SYSLIST procedure.

Print Spooling

Because main storage processing is much faster than printing, the system spends a lot of its time waiting for the printer to print data already processed. To help eliminate this waiting, you can use print spooling for all your printers. Print spooling is the capability of the SSP to store printed output on disk (in an area called the spool file) for later printing.

Advantages

Print spooling has several advantages over normal printing:

- Programs can run faster because they do not have to wait for the printer to print each line of data.
- More than one program that sends output to the same printer can be run at the same time. These programs do not have to wait for the printer to become available. (Without print spooling, only one program could use a printer at any one time. Other programs that direct output to that printer would have to wait.)
- Programs producing output can be run even if the printer is not working. The data is saved on disk and can be printed when the printer is working again.
- Multiple copies of the same output can be produced without running the program multiple times.
- Different priority levels can be assigned to printed output, so that a report with a higher priority can be printed before less important output.
- Output can be grouped, then printed by the type of form used (for example: invoices, narrow paper, or regular paper).
- Printing can be restarted if printing errors occur without running the program again.
- Output can be redirected to another printer.
- Printing output from the spool file makes more efficient use of the main storage processor, the printer, and the communications lines (for remote printers).
- Separator pages can be printed for each job.

How To Select Print Spooling

You decide whether to use print spooling during system configuration. The manual *Changing Your System Configuration* has more information about system configuration. The default is to use print spooling.

If you select print spooling, you can later cancel it by using the IPL override displays. You can also use the SPOOL-NO parameter of the PRINTER OCL statement to avoid print spooling for just one print file as opposed to canceling spooling for the whole system.

Control of Print Spooling

The system operator has control over all printers and all print entries in the spool file. The system operator can, for example, start or stop the printers as well as change the number of copies to print.

Subconsole operators have control over their designated printer(s) as well as all print entries in the spool file to be printed on their designated printer. Display station operators have some control over the print entries they create. For example, they can change the number of copies to print.

The commands used to control print spooling are shown under "Controlling or Displaying Print Spooling Information" on page 3-23.

How Print Spooling Works

When print spooling is active, a system program called the **spool intercept routine** intercepts each line to be printed and stores it on disk in the **spool file**. When a printer is ready to print the output, another system program called the **spool writer** gets lines from the spool file and prints them. Figure 3-1 shows this process.

In this example, three programs are running on the system. Programs 1 and 2 each print one report; program 3 prints two reports. Each report is placed in separate print file (the printed output from a program is called a **print file**). Each of the four reports is handled through a separate intercept buffer, and each is a separate entry in the spool file. Three printers are being used, program 1 uses printer P1, program 2 uses P2, and program 3 uses P3.

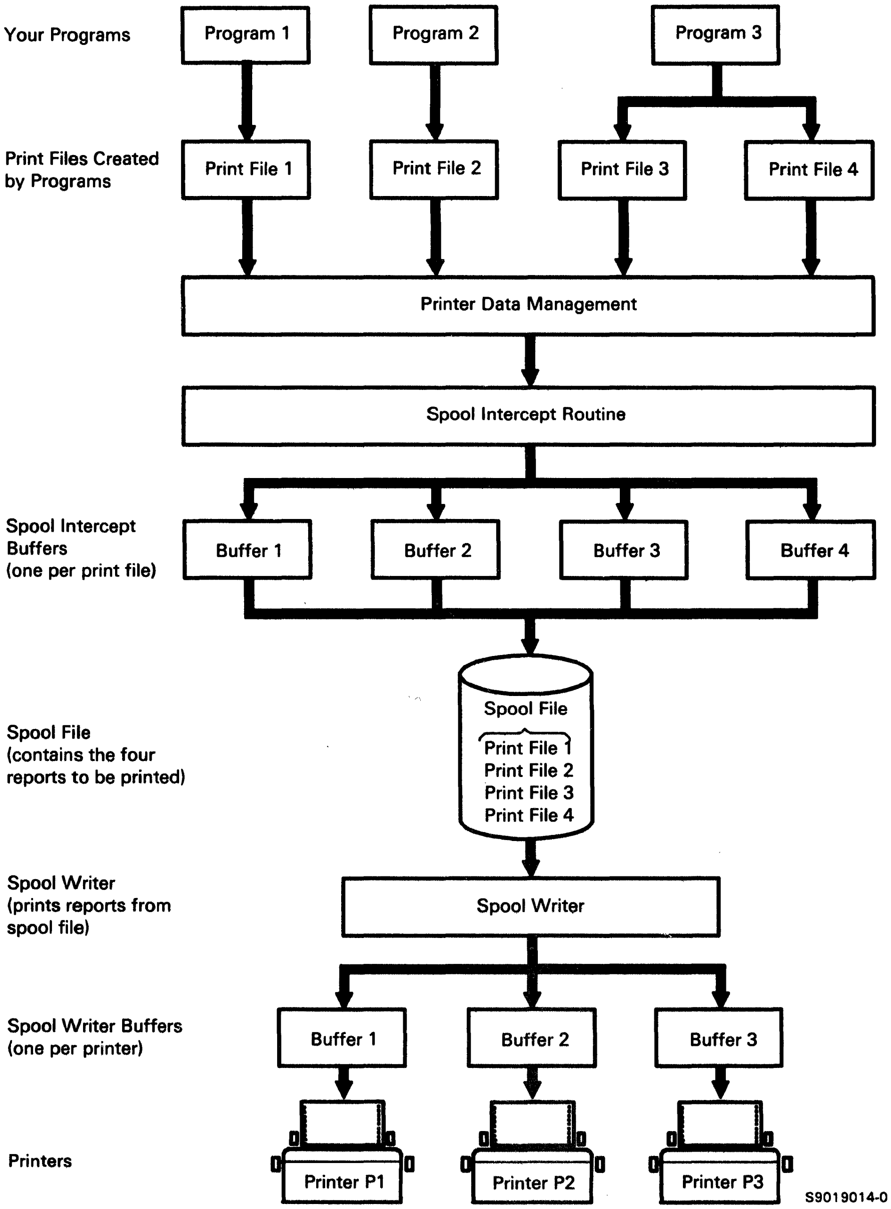


Figure 3-1. Print Spooling Overview

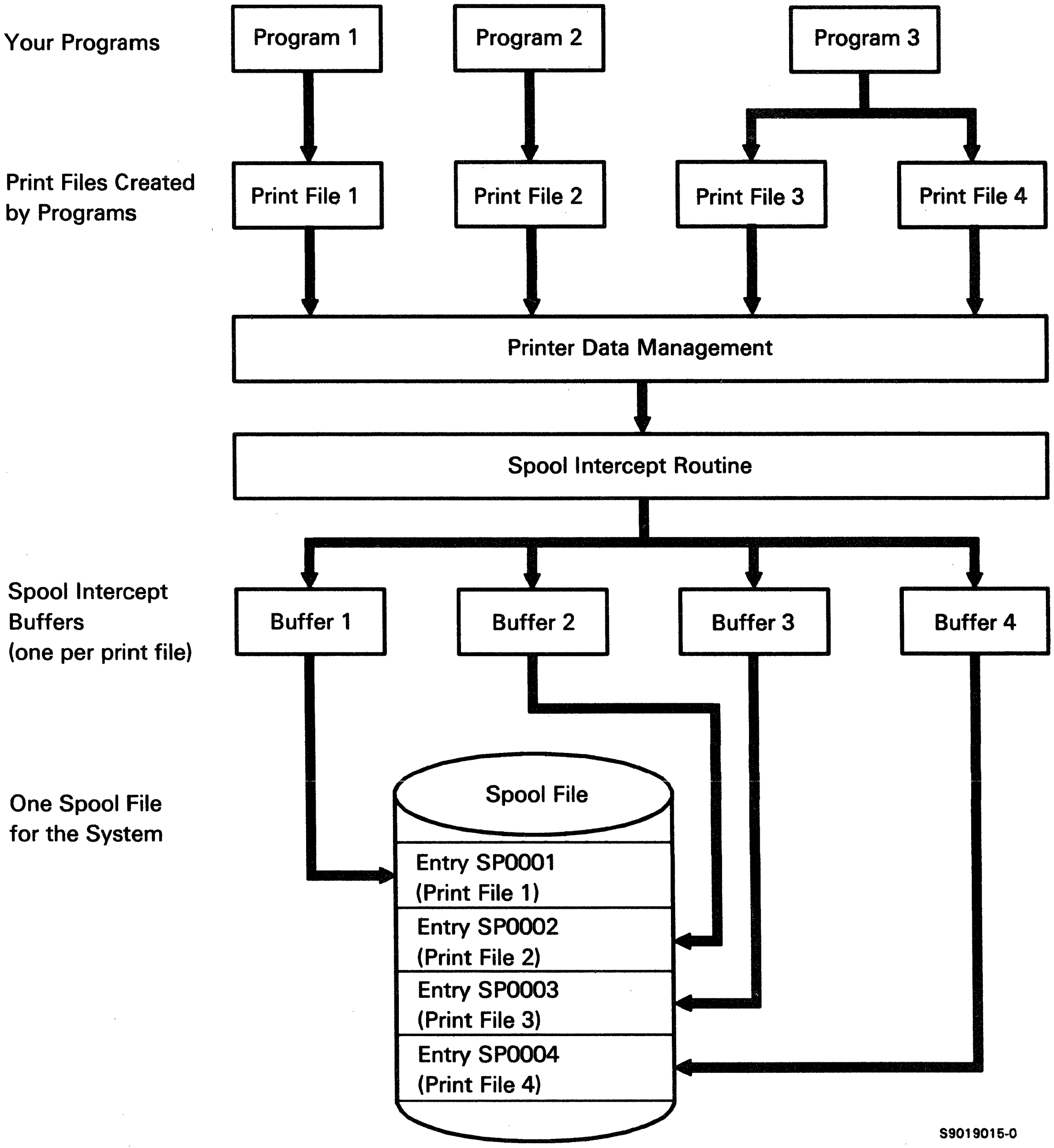
The following sections describe the different parts of print spooling shown in Figure 3-1.

Spool Intercept Routine

The printed output from your program is called a **print file**. Your programs pass the print files to printer data management one line at a time. The spool intercept routine is the part of the SSP that takes output from printer data management and places it into **spool intercept buffers**.

The SSP assigns each print file its own spool intercept buffer, and the spool intercept routine handles each print file separately. A unique spool identification consisting of the characters SP followed by four decimal digits is assigned to each print file. There is no limit to the number of print files that can be processed at the same time by the spool intercept routine.

The following diagram shows how the spool intercept routine handles the output from more than one program.


Your Programs
Program 1
Program 2
Program 3
Print Files Created by Programs
Print File 1
Print File 2
Print File 3
Print File 4
Printer Data Management
Spool Intercept Routine
Spool Intercept Buffers (one per print file)
Buffer 1
Buffer 2
Buffer 3
Buffer 4
One Spool File for the System
Spool File
Entry SP0001 (Print File 1)
Entry SP0002 (Print File 2)
Entry SP0003 (Print File 3)
Entry SP0004 (Print File 4)

S9019015-0

Spool Intercept Buffer

A spool intercept buffer is created whenever a print file is opened. As the program prints data, the data is temporarily placed into the intercept buffer. When a spool intercept buffer is full, the spool intercept routine writes the data to the spool file. Also, when a print file is closed (either by the program or when the program ends), the spool intercept routine places any remaining printer data in the spool file.

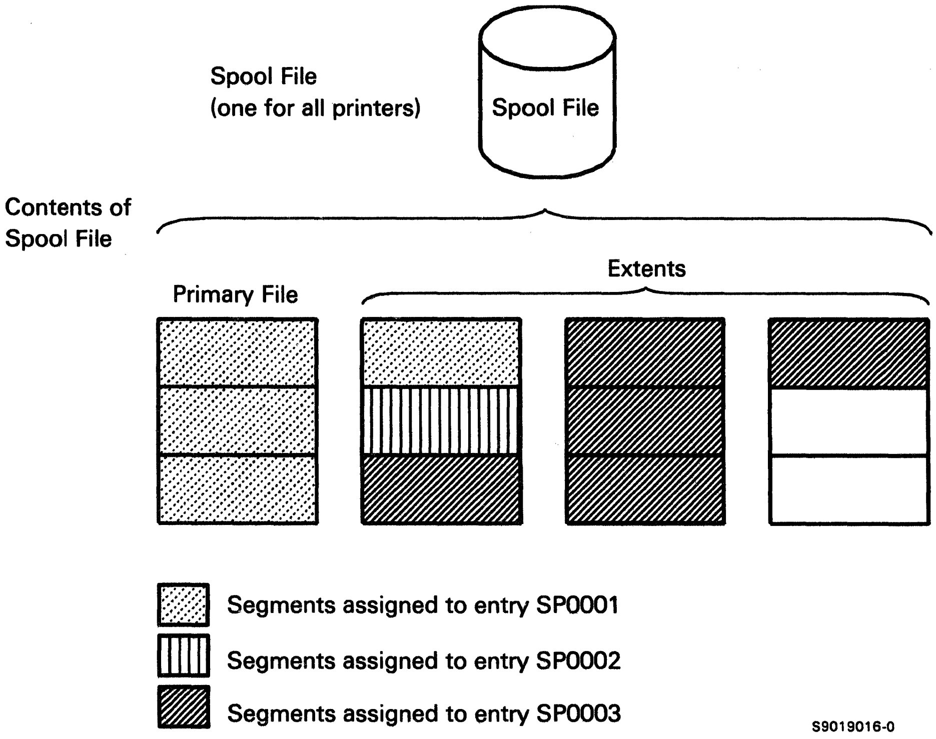
Spool Intercept Buffer Size: The size of a spool intercept buffer assigned to a print file depends on the ACTIVITY parameter of the PRINTER OCL statement. Specifying a high activity gives you a larger buffer area than if you specify a low activity. Spooled output is written to the spool file when the spool intercept buffer allocated to your program becomes full. If the buffer is too small for the amount of activity, it will fill up quickly, and printer output will have to be written to the spool file very often, possibly increasing the processing time for your job and other jobs on the system. If the buffer is too large, you may needlessly increase the amount of main storage your program requires. The *System Reference* manual has more information about the PRINTER OCL statement.

Spool File

The system has one spool file that is shared by all printers on the system. The spool file is on disk and consists of a primary file and up to five additional areas on disk called **extents**. The size of the spool file (the primary file) is specified during system configuration, and can be from 12 to 12,800 blocks. The default size of the spool file is based upon the disk size of the system. Each extent is the same size as the primary file. The extents are created only when they are needed; that is, when the primary file is full. When an extent becomes empty, it is removed.

The primary spool file and extents are divided into **segments**. The size of these segments is specified during system configuration, and can be from 1 to 16 blocks. The system starts by allocating one segment to a print file; as the segments are filled with printer output, more segments are allocated to the print file.

The following example shows a 12-segment spool file containing three entries: SP0001, SP0002, and SP0003. Entry SP0001 is contained in four segments; entry SP0002 is contained in one segment; and entry SP0003 is contained in five segments. Two segments are unused.


Spool File
(one for all printers)
Spool File
Contents of
Spool File
Extents
Primary File
Segments assigned to entry SP0001
Segments assigned to entry SP0002
Segments assigned to entry SP0003
S9019016-0

Spool File Size: The system automatically calculates the default size of the spool file and segment sizes needed based on the amount of disk space. In many cases, the size calculated by the system is the best size. However, if you want to override the system defaults, the following are some things to consider:

- Use the system default values whenever possible. If you frequently get the message that the spool file is full, consider increasing the size of the spool file by using the CNFIGSSP procedure.

- The recommended size for the spool file segments is based on the size of the typical print file. Large segments reduce the work the system does creating additional segments because fewer segments are needed. However, the amount of unused space in the last segment may be large. Smaller segments make more efficient use of spool file space by reducing the amount of unused space in the last segment of a print file.

For more information about the spool file size and segment size, see the *Changing Your System Configuration* manual.

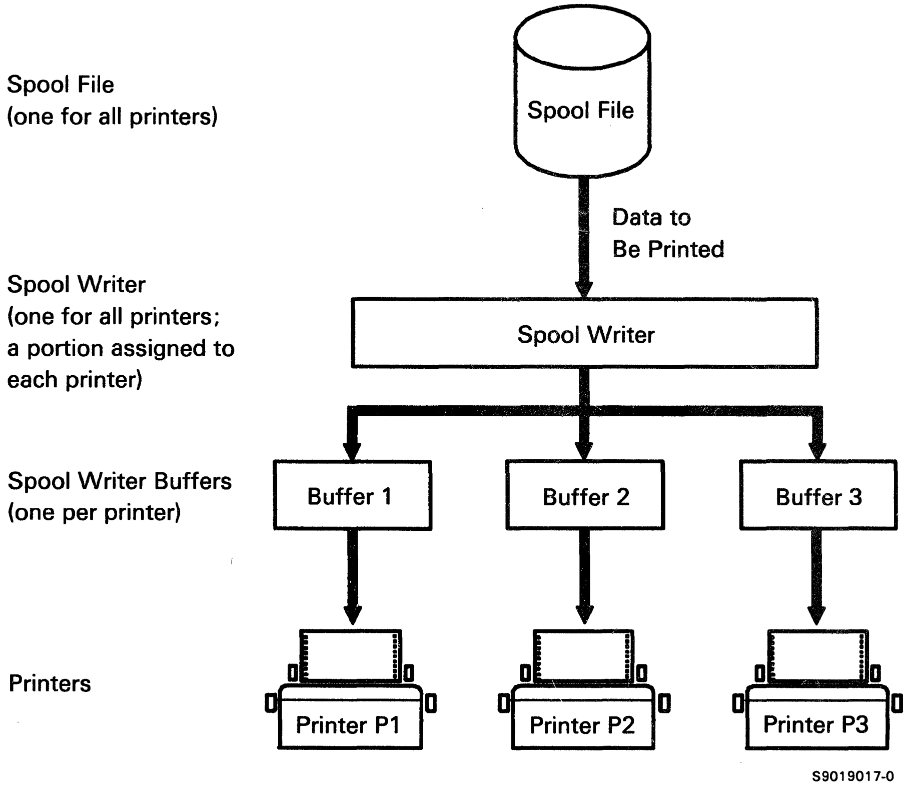
Spool File Placement on Disk: If your system has more than one disk drive, you can specify on which disk the spool file is to be placed. If you are using the spool file frequently and have other programs running on the system, there may be an increased demand for disk use. To help balance disk activity, you can specify a preferred disk location for the spool file during system configuration. You can use the system measurement facility (SMF) to measure the disk activity. SMF is described in the *SMF Guide.*

Spool Writer Program

The spool writer is the part of the SSP that prints output that has been stored in the spool file. One spool writer is shared by all the printers on the system. The spool writer is a program that gets data from the spool file and places that data into a **spool writer buffer**. The data is then printed from this buffer; each printer has its own buffer. The size of these buffers is determined during system configuration.

Although the spool writer is one program used for all the printers, you can control the operation of the portion of the program assigned to each printer. For example, you can stop the portion used by printer P3 and leave the portions for printers P1 and P2 running. The STATUS WRT control command shows information about the portion of the spool writer assigned to each printer.

The following diagram shows how the spool writer handles printer output stored in the spool file.



Before the spool writer can begin printing output for each printer, the spool writer for that printer must first be started. During system configuration, you can specify that the spool writer is to start automatically. If you choose this automatic start option, the spool writer is started when you perform system IPL. See the *Changing Your System Configuration* manual for more information about automatically starting the spool file.

If you choose not to have the spool writer started automatically, the START PRT or RESTART PRT control command must be entered to start the spool writer. These START PRT control commands can be entered by the system operator or the Subconsole operator. Once the spool writer has been started, it prints output whenever a spool file entry is available to be printed.

Spool Writer Buffer Size: During system configuration, you can select the size of the spool writer buffer to be used for a printer. See the *Changing Your System Configuration* manual for more information about changing the spool writer buffer size.

Separator Pages

The spool writer can also print one to three separator pages between spool file entries. You can also choose to have no separator pages printed. The separator pages help you to identify the printer output. Each separator page contains:

- Name of the procedure that started the job
- User ID of the operator who ran the job (this may be blank if a MRT program printed the output)
- Name the system assigned to the job
- Name of the print file
- Printer ID of the printer
- Number of pages in the print file

You can get separator pages by specifying an option during system configuration. See the *Changing Your System Configuration* manual for more information about changing the number of separator pages for a printer. You can also use the CHANGE SEP control command to change the number of separator pages (from the system console or a subconsole).

The line length on separator pages is normally 80 characters. However, if you are justifying the right margin on the 5219 Printer and if the line length for your text is less than 80 characters, the characters at the end of the line on the separator page are not printed. For example, if you are justifying text on the 5219 Printer using a line length of 72 characters, only the first 72 characters on each line of the separator page are printed.

Designing Printed Output for Your Programs

This section describes:

- Considerations to use when you are deciding what reports need to be printed
- How to design printed reports
- Performance considerations to use for printers
- Assigning forms numbers to the different types of forms you use

Printed Output Considerations

When you are determining the output of a program, you should consider what the person using the information needs. Printing the output may not always be the best method to use; displaying the output is often a better choice. The information used as output, whether displayed or printed, should be useful and should reflect the needs of the person using the output.

Use printed output for:

- Information that is not needed immediately or often
- Information that is not needed by several people (although you can print multiple copies if you want)
- Information that does not change frequently
- Large volumes of output
- Information that must be sent to several locations (Diskettes would also be a good choice if the other locations have a computer system.)

Also, a formatted report may not be needed in all cases; that is, a Print key listing of a display could serve as the output.

See Chapter 13, "Displays," for displayed output considerations.

Designing Your Printed Reports

A well-designed report should be easy to read and easy to handle. When designing a report, a document called a printer spacing chart will be useful. The printer spacing chart is a blank form that resembles a page to be printed. You can use the *IBM Print Chart*, GX20-1816, or an equivalent chart.

Each vertical column represents one print position. A typical printed page has 132 print positions. However, depending on the type of printer you have, you can print up to 198 positions on a page. Each horizontal line represents one print line. A typical printed page has 66 lines (at 6 lines per inch). However, depending on the type of printer you have, you can print four or eight lines per inch. The number of print positions and lines per page also depends on what size paper you use.

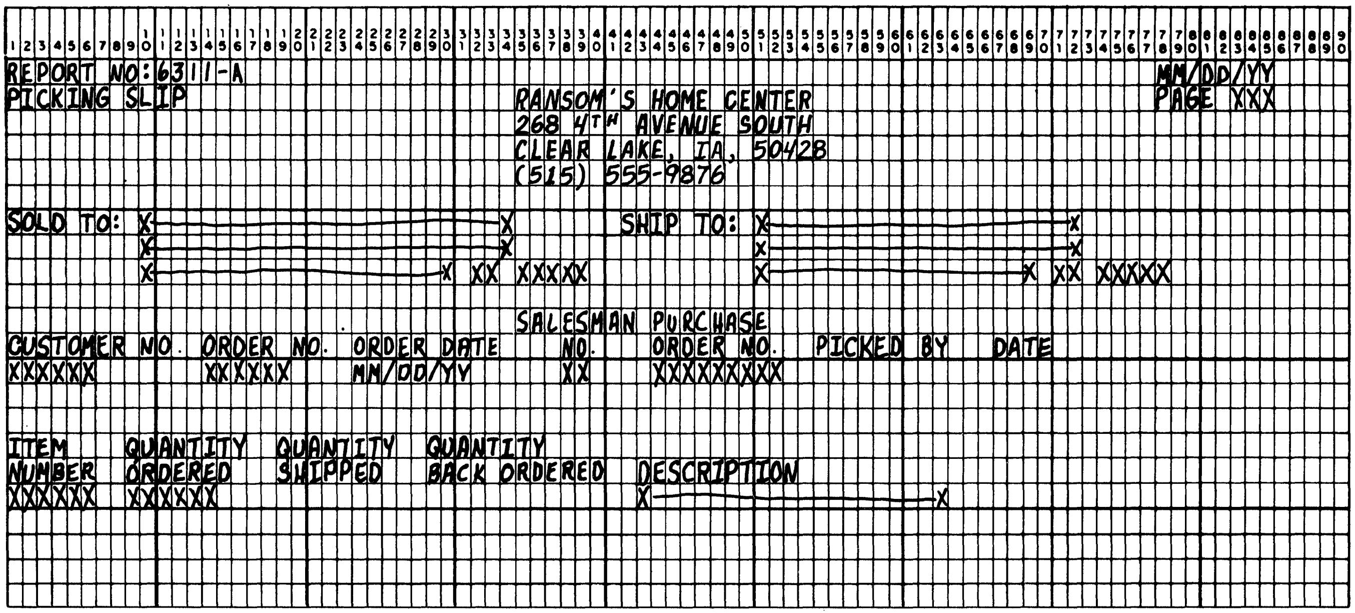
To design a report, you select the print positions on the printer spacing chart where you want information to print, and place Xs or sample data in the selected positions. Report titles and column headings should be used to identify the information on the report.

The goals in designing your reports on a printer spacing chart are to:

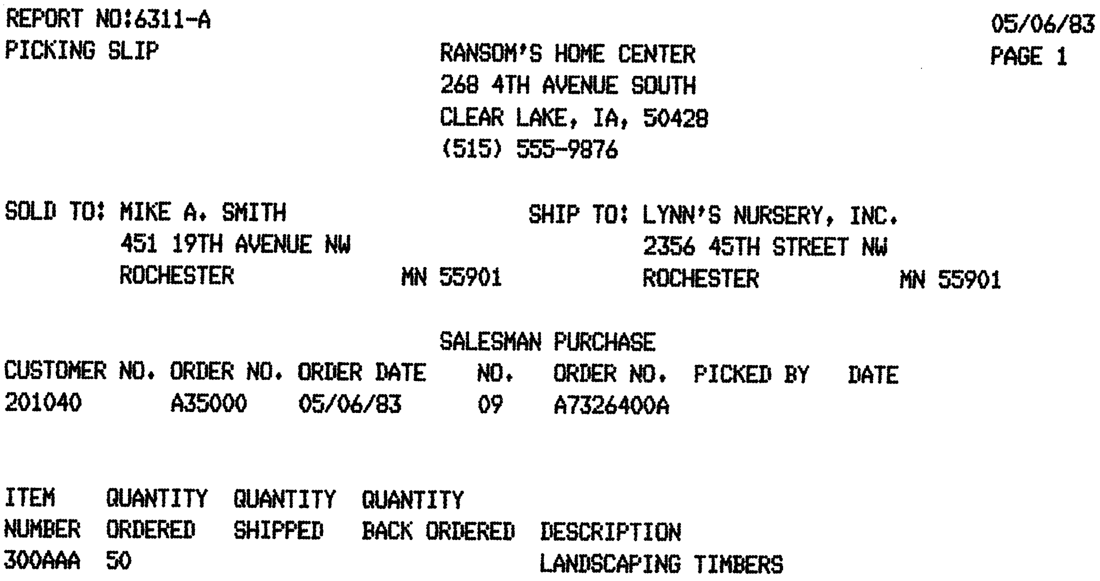
- Ensure that you know how the report will look when it is printed.

- Have a model of the report to help you code the printing sections of your program. The chart can be used to identify the lines and columns for printer output.

The following example shows a completed printer spacing chart along with the computer-produced output.



```
REPORT NO:6311-A                                                    MM/DD/YY
PICKING SLIP                  RANSOM'S HOME CENTER                  PAGE XXX
                              268 4TH AVENUE SOUTH
                              CLEAR LAKE, IA, 50428
                              (515) 555-9876

SOLD TO: X--------------------X      SHIP TO: X------------------X
         X--------------------X               X------------------X
         X-----------------X XX XXXXX         X---------------X XX XXXXX

                              SALESMAN PURCHASE
CUSTOMER NO. ORDER NO. ORDER DATE    NO.    ORDER NO.  PICKED BY   DATE
XXXXXX      XXXXXX    MM/DD/YY       XX     XXXXXXXXX

ITEM   QUANTITY QUANTITY QUANTITY
NUMBER ORDERED  SHIPPED  BACK ORDERED DESCRIPTION
XXXXXX XXXXXX                         X-----------------X
```



```
REPORT NO:6311-A                                                05/06/83
PICKING SLIP                  RANSOM'S HOME CENTER              PAGE 1
                              268 4TH AVENUE SOUTH
                              CLEAR LAKE, IA, 50428
                              (515) 555-9876

SOLD TO: MIKE A. SMITH                  SHIP TO: LYNN'S NURSERY, INC.
         451 19TH AVENUE NW                      2356 45TH STREET NW
         ROCHESTER          MN 55901             ROCHESTER          MN 55901

                              SALESMAN PURCHASE
CUSTOMER NO. ORDER NO. ORDER DATE    NO.    ORDER NO.  PICKED BY   DATE
201040          A35000    05/06/83      09     A7326400A

ITEM   QUANTITY QUANTITY QUANTITY
NUMBER ORDERED  SHIPPED  BACK ORDERED DESCRIPTION
300AAA 50                             LANDSCAPING TIMBERS
```

When you are designing the output report, use the following guidelines:

- Leave enough space on the edges (margins) of the reports that must be stapled or bound.
- Separate fields on the report by at least one space.
- Group information items that are similar.
- Number all pages of a report. Print spooling allows you to restart the printing of a report. If you have to restart the printing of a report, you specify that the printing is to begin at a specified page, rather than reprinting the entire report.
- Provide meaningful headings for the data on a report. Abbreviations, codes, and special symbols should be avoided. If you need more space than is available on the printer spacing chart, use several lines for long headings, rather than abbreviating them.

Printer Performance Considerations

The system supports several printers. Each printer has unique forms design considerations.

For information about the physical dimensions of printer forms, refer to the documentation for your printer.

Considerations for Dot Matrix Printers

The characters are made up of dots on a dot matrix printer. With some dot matrix printers, it takes the print head several passes to complete a line. This head movement takes time, therefore, you should design forms to reduce the head movement. Refer to the documentation for your printer to determine if these considerations affect your printer.

- Consider the differences in print density and line length. Greater print density means slower print speed in lines per minute but faster speed in characters per minute. Shorter lines mean greater lines per minute because the print head does not have to move a great distance to complete a line.
- Repetitive printing of the **same** character, such as an asterisk (*) in each position of a line, requires the printer to print the line in several passes, slowing the print speed. Rather than using a **solid line** of asterisks for a separator, use a line of alternating asterisks and blanks.
- Do not center one or two fields on a line if the fields do not have to be centered. Adjusting them to the leftmost portion of the line may shorten the printing time.

- Do not use a large form and print a small amount of information on it. This may be inefficient because many line returns could be required to print the data. If you keep the forms as short as possible, you might improve printing speed.

- Place fields in a horizontal line rather than spacing them vertically. Also, minimize the horizontal space between fields.

- Plan horizontal print fields so that items that are not always printed are printed last on the line. For example, an item description field might be a good variable-length field to print last on a line.

Considerations for Character Printers

Character printers print one character at a time by a print head that must move to the appropriate position on the line. This head movement takes time. Therefore, you might consider designing your forms to reduce the amount of head movement required. For example:

- Do not center one or two fields on a line if the fields don't have to be centered. Adjusting them to the leftmost portion of the line may shorten the printing time.

- Place fields in a horizontal line rather than spacing them vertically. Also, minimize the horizontal space between fields.

- Plan horizontal print fields so that items that are not always printed are printed last on the line. For example, an item description field might be a good variable-length field to print last on a line.

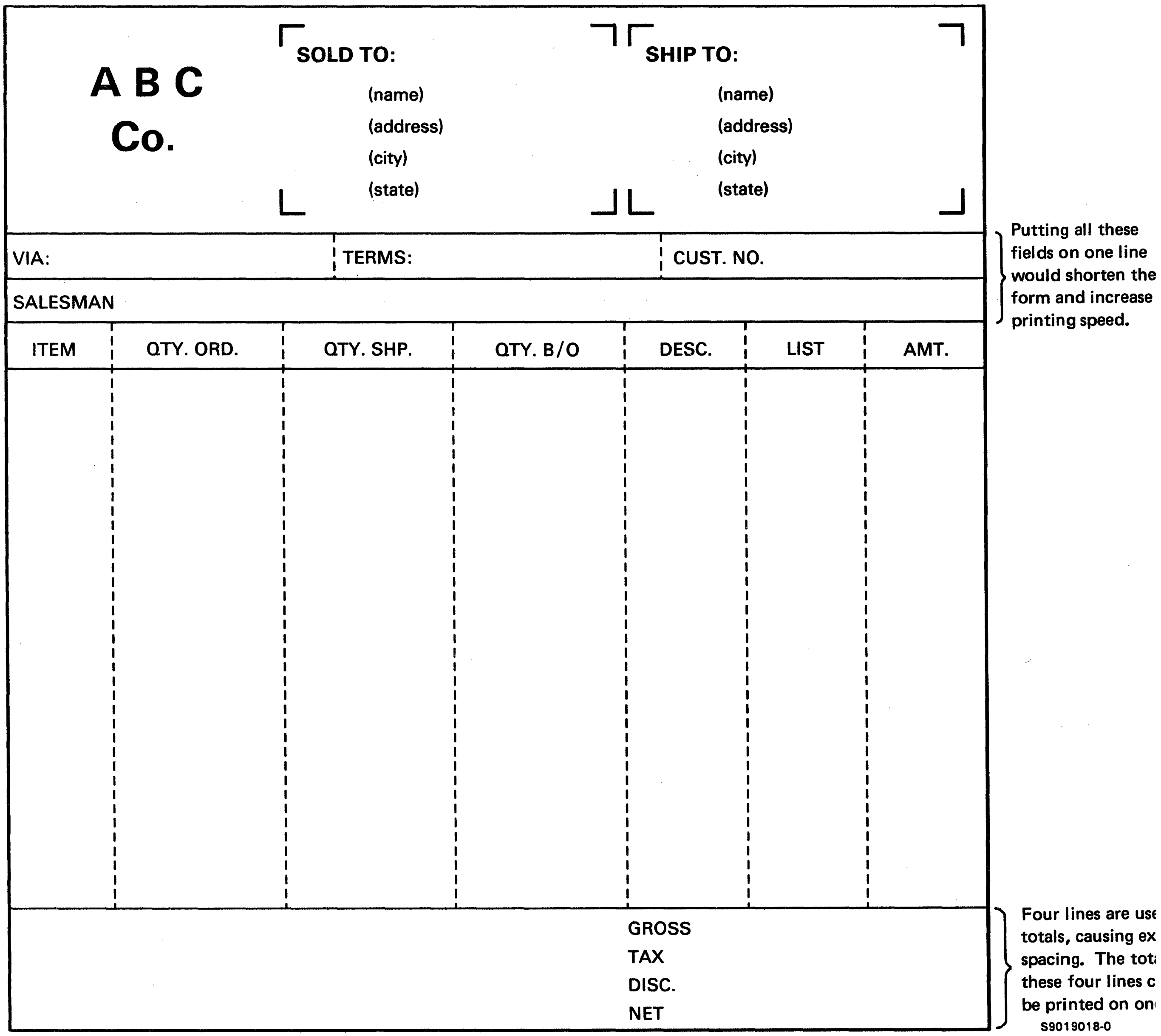
Vertical line spacing on some printers is affected by the following considerations. Refer to the documentation for your printer to determine if these considerations affect your printer.

- When 12 lines or less are specified, and the output is sent to a printer using individual sheets, the printer prints up to the number of lines specified before ejecting the sheet.

- Some printers do not print on the first line of an individual sheet. If a program attempts to print on the first and last lines of a sheet, the last line is printed at the top of the next sheet.

- The fewer the number of lines per page, the more processing time some printers take to perform the task.

Figure 3-2 shows an initial design and Figure 3-3 shows an improved design for an output form used on a 5256 Printer.



A B C Co.

SOLD TO:
(name)
(address)
(city)
(state)

SHIP TO:
(name)
(address)
(city)
(state)

VIA: | TERMS: | CUST. NO.

SALESMAN

ITEM	QTY. ORD.	QTY. SHP.	QTY. B/O	DESC.	LIST	AMT.

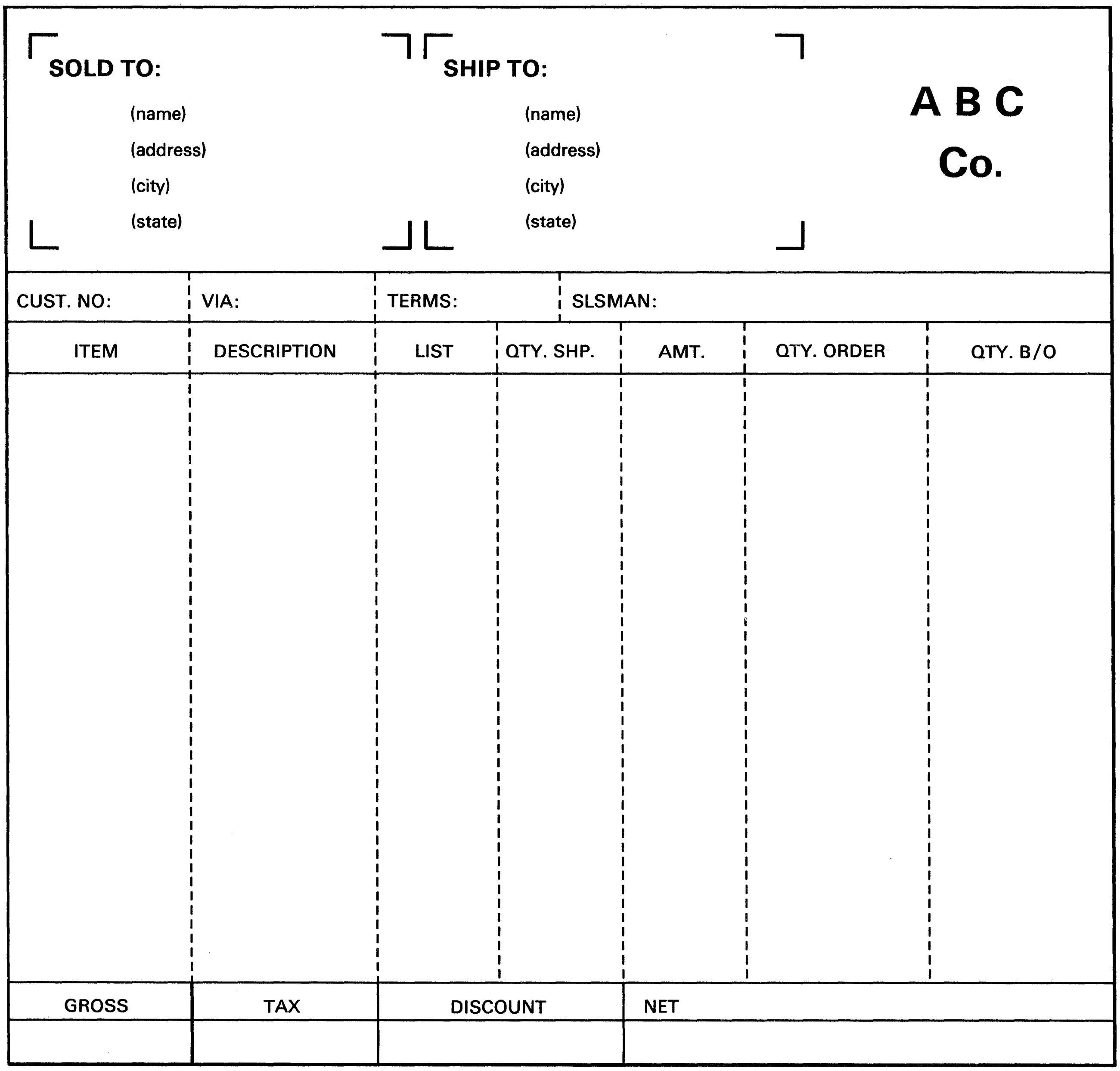
GROSS
TAX
DISC.
NET

Putting all these fields on one line would shorten the form and increase printing speed.

Four lines are use totals, causing ex spacing. The tot these four lines c be printed on on

S9019018-0

Figure 3-2. Initial Forms Design



SOLD TO:

(name)
(address)
(city)
(state)

SHIP TO:

(name)
(address)
(city)
(state)

A B C
Co.

CUST. NO:	VIA:	TERMS:	SLSMAN:

ITEM	DESCRIPTION	LIST	QTY. SHP.	AMT.	QTY. ORDER	QTY. B/O

GROSS	TAX	DISCOUNT	NET

S9019019-0

Figure 3-3. Improved Forms Design

Considerations for Line Printers

Line printers, which use a character belt, print one line at a time. Good design techniques for a line printer should, therefore, try to reduce the number of lines printed on each form. Increasing the number of characters printed per line might shorten the time required to print the form. An example of this would be to combine two lines into one print line. Consider using as wide a form as possible and as short a form as possible.

For preprinted forms such as picking slips or invoices, consider shading alternate lines; this technique can make a long list of items more readable. Design and order your preprinted forms well before you plan to use them. Request a sample of the form so that you can verify its accuracy before it is printed.

Assigning Forms Numbers

If you have different types of paper for your printed output (for example, one type of paper for general use and another type for a specialized use, such as preprinted checks), you may want to assign forms numbers to these different type of forms.

A **forms number** is a 1- to 4-character identifier you assign to each type of forms you have. For example, the standard forms that you may have for each printer could be named 0001. Preprinted check forms could be named CHEK.

The system automatically keeps track of the forms numbers that are currently specified for each printer. When you specify a certain forms number for printing and if the forms numbers do not match, the operator who controls that printer is prompted to change the paper in that printer to the forms you specified.

See “Printing Output by Forms Number” on page 3-27 for more information about how you can use forms numbers.

Printer Control Guidelines

This section describes several functions that are available to manage or control printer output. Generally, the functions are:

- Changing the session or Print key printer, changing the system list device, and setting printer information for a display station.
- Controlling or displaying printer output.
- Displaying and copying printed output from the spool file.
- Printing output by forms type.
- Combining several print files.
- Assigning defer status to printed output.
- Assigning priorities to printed output.

Changing the Session Printer

For each display station, a printer used to receive output is specified during system configuration. This printer is called the **session printer**. The session printer is used for all printed output created while an operator is signed on. The PRINT procedure changes any one or all of the following items:

- The printer ID to be used for printed output, including system list and Print key output.
- The number of lines per page.
- The vertical print density, also called lines per inch.
- The horizontal print density, also called characters per inch.
- The forms number to be used.

The PRINT procedure changes these items for the session. When an operator signs off, the settings return to the defaults.

*Note: Not all printers support lines per inch and/or characters per inch. The **System Reference** manual has more information about the PRINT procedure.*

Changing the System List Device

When an operator signs on, the system list device is the default printer specified during system configuration. The SYSLIST procedure can change the system list device to another printer, the display station, or off (no output is listed). The *System Reference* manual has more information about the SYSLIST procedure.

Changing the Print Key Printer

When an operator signs on, the Print key printer is the default printer specified during system configuration. Use the STATUS SESSION control command to display the ID of the Print key printer. The PRINTKEY procedure can change the printer to be used for Print key output and can specify whether a heading or border is to be printed with the display image. The WORKSTN OCL statement can also be used to change the Print key printer. The *System Reference* manual has more information about the PRINTKEY procedure or the WORKSTN OCL statement.

Changing the Printer Configuration Information

The SET procedure is used to specify the following printer-related items for the display station:

- Printer ID to use for session output as well as Print key output
- Forms number
- Number of lines per page
- Print belt image
- Whether a heading and border should be printed with Print key output

These items remain in effect from session to session; that is, after the operator signs off, the values remain in effect. The *System Reference* manual has more information about the SET procedure.

Changing Printer Information in a Procedure

In a procedure, you can use the FORMS, PRINTER, or WORKSTN OCL statements to control how output is printed. For example, you can change any one or all of the following items:

- Printer ID to be used for all printed output (including Print key output).
- Number of lines per page.
- Vertical print density, also called lines per inch.
- Horizontal print density, also called characters per inch.
- Forms number.
- Number of copies to be printed.
- Whether the paper needs to be aligned before the output is printed.

The *System Reference* manual has more information about these OCL statements.

Controlling or Displaying Print Spooling Information

Print spooling is controlled by either the system operator or subconsole operators entering various control commands.

To display information about the status of the spool file, use the STATUS PRT or STATUSF PRT control command. To display information about the status of the spool writer, use the STATUS WRT control command. Control commands can be entered either at the system console or a subconsole. However, some commands can be entered from any work station.

Note: *If password security is active, operators with a security classification of system operator or higher can control print spooling from any display station. The commands entered are treated the same as if they were entered at the system console.*

The following is a list of commands you can use to control print spooling.

CANCEL PRT	Cancels one or more spool file entries.
CHANGE COPIES	Changes the number of copies of output to be printed.
CHANGE DEFER	Changes the defer attribute of a spool file entry. This indicates whether an entry can begin printing before all the data has been completed by the program.
CHANGE PRTY	Changes the priority of the spool writer for a printer.
CHANGE SEP	Changes the number of separator pages used to separate spool file entries for a printer.
CHANGE PRT	Changes the order in which spool file entries are printed.
CHANGE FORMS	Changes the forms number to be used for a spool file entry.
CHANGE ID	Changes the printer ID assigned to one or more spool file entries.
HOLD PRT	Holds selected spool file entries to prevent them from being printed.
RELEASE PRT	Releases selected held spool file entries to allow them to be printed.
RESTART PRT	Restarts the printing of a spool file entry.
START PRT	Starts the spool writer so that entries can be printed.
STOP PRT	Stops the spool writer so that entries cannot be printed.

Operating Your System and *Using Your Display Station* manuals have more information about these control commands.

Controlling Printed Output with the Print Queue Manager

The print queue manager is part of the base SSP and resides in #LIBRARY. The print queue manager provides an interface that allows DisplayWrite/36 (DW/36) to access and change information about entries on the spool queue and job queue.

A request is made to the print queue manager through the print queue. Information is returned in a print file.

The print queue manager makes a request to the spool queue manager through the queue file. Information is returned in a spool print file to the print queue manager.

The print queue manager makes a request to the job queue manager through the job queue. Information is returned in the job queue file to the print queue manager.

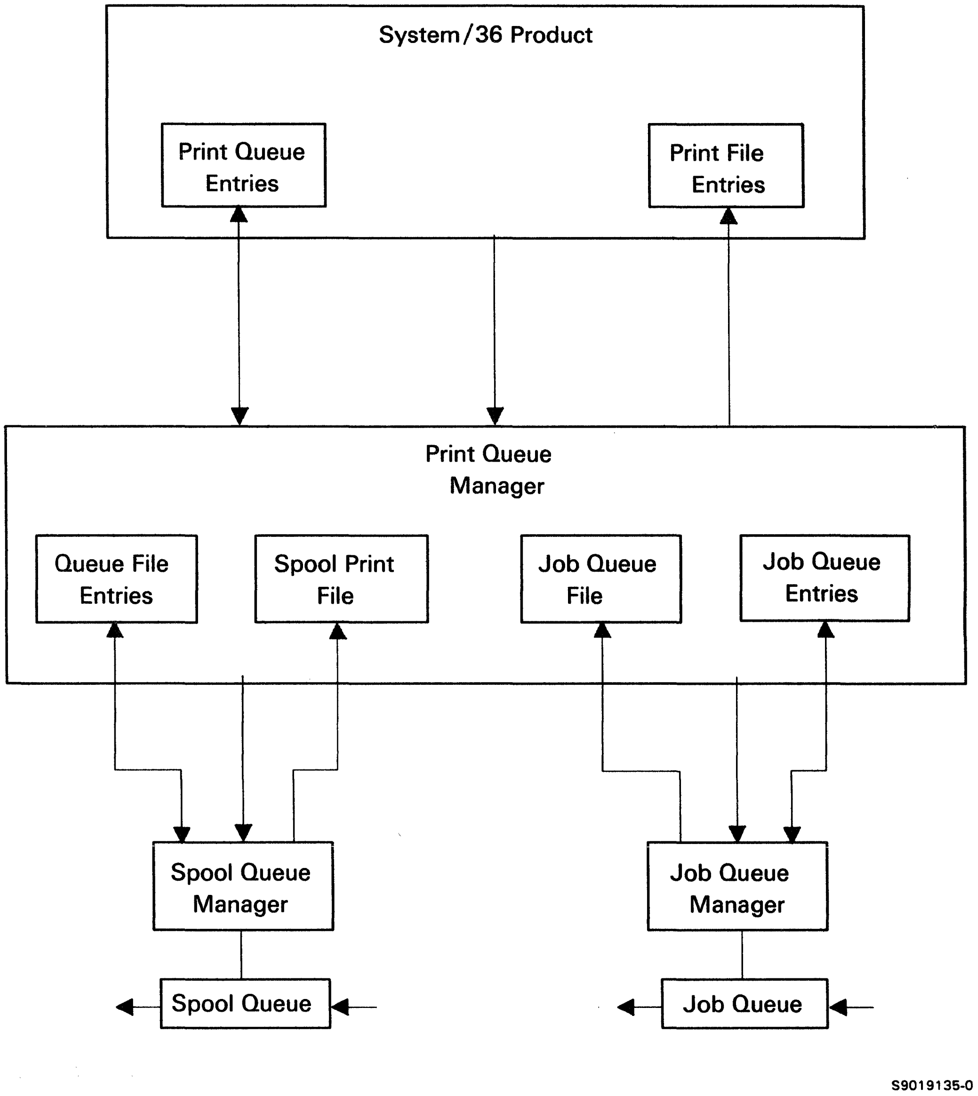


Figure 3-4. Print Queue

The following is a list of commands you can use to control the print queue manager:

Function	Definition
SEARCH	Provides the requested number of print files (provided they are available).
CANCEL	Cancels one or more print files from the spool or job queues.
HOLD	Holds one or more print files from the spool or job queues.
RELEASE	Releases one or more print files from the spool or job queues.
MOVE	Moves one or more print files from the spool or job queues.

Consider the following when moving print files:

- Print files on the spool queue can be moved to any position on the spool queue.
- Print files on the job queue can be moved to any position on the job queue.
- Print files on the spool queue may not be moved to the job queue.
- Print files on the job queue may not be moved to the spool queue.

You can specify the type of information to be used to qualify the requested function. A qualifier list may be used to subset the list of print files returned to the user. The valid qualifiers are:

- Printer ID
- User ID
- Forms ID
- Print File

There are four possible function modifiers that can be specified.

Modifier	Definition
Spool	Causes the requested function to be performed only on the print files that are on the spool queue.
JobQ	Causes the requested function to be performed only on the print files that are on the job queue.
Forward	Causes a search to continue forward from the position of the last search requested (search function only).
Backward	Causes a search to continue backward from the position of the last search requested (search function only).

Copying and Displaying Spool File Output

Operators can copy and display printed output from the spool file by using the COPYPRT procedure. The COPYPRT procedure allows operators to:

- Copy reports from the spool file to a disk file.
- Display printed reports at a display station before they are printed.
- Print reports copied to the disk file. You can also print selected pages of a report.

This procedure allows users to look at reports and decide which pages of the report they want to print. You can use this procedure to avoid printing unnecessary reports and wasting paper.

Before operators use the COPYPRT procedure, they should make sure that the specified report is held on the spool file by using the HOLD PRT control command or the PRIORITY-0 parameter of the PRINTER OCL statement.

Note: The COPYPRT procedure does not copy spool print records that were created in transparent mode. Transparent mode allows for user programs to send their own data stream to the printer. COPYPRT does not process transparent mode because it does not know what is contained in the data stream. Therefore, transparent mode is not encouraged but there is an operation code for it in the $DTFP macro.

The disk file used with the COPYPRT procedure can be saved on diskette and then printed later. The *System Reference* manual has more information about the COPYPRT procedure.

Printing Output by Forms Number

Normally, when output is printed, it is printed in a first-in, first-out basis, with no regard to forms number. If, for example, two or more jobs use different forms numbers, the operator controlling a printer has to change the forms very often. The system allows operators to control the printing of output using different forms numbers.

The system has two methods of printing by forms number. Using the START PRT control command, operators can print either by a specific forms number or by groups of forms numbers. This allows operators to print all reports that use the same forms number together. For example, all jobs in the spool file having special invoice forms could be printed together.

To print by groups of forms (using the FORMS parameter of the START PRT command), the spool writer prints all entries in the spool file using the forms currently mounted on the printer. When the spool writer has finished printing the entries, the spool writer prompts the operator to change forms in the printer. This can reduce the number of times an operator has to change the paper in a printer.

To print by a specific forms number, the spool writer prints only those entries that have the specified forms number. If there are no more entries in the spool file with that forms number, the spool writer does no more printing.

For example, to print reports that required forms number A112 on printer P2, an operator could enter the command:

```
START PRT,P2,A112
```

and the spool writer would print only reports that required forms number A112 on printer P2. If there were no entries in the spool file with forms number A112 for that printer, the spool writer would not print any entries.

The START PRT control command in the *Operating Your System* and *Using Your Display Station* manuals have more information about printing by forms type.

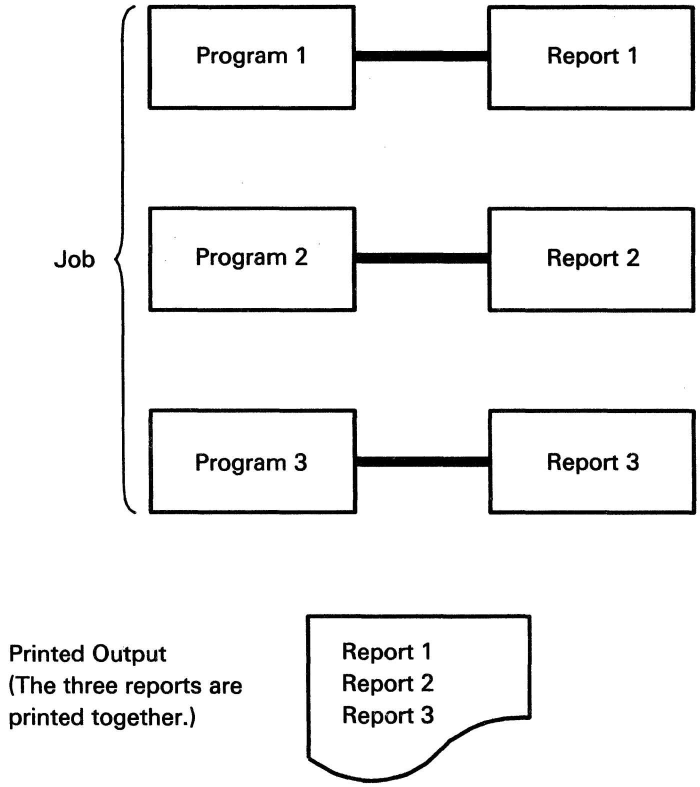
Combining Several Print Files in One Job

The system allows you to combine printer output from several programs into one large print file. The programs must:

- Run in the same job.
- Specify the same printer for the output.

The first program of the job creates the print file, and the remaining programs in the job add output to the same print file.

For example, if you have three order entry programs within a job, and each program produces one report, you can have one print file instead of three separate print files. This allows all reports produced by the order entry programs to be printed at the same time. The following diagram shows how this process works.



S9019020-0

This combining of multiple reports into one print file is controlled by using the CONTINUE parameter of the PRINTER OCL statement. The *System Reference* manual has more information about the PRINTER OCL statement.

Assigning the Deferred Status to Printed Output

You can use the DEFER parameter of the PRINTER OCL statement to specify the defer status of a print file. The defer status indicates whether a print file can begin printing before the file is completed by the program. Normally, the output is not printed until it is complete; that is, until the job ends or the program closes the print file.

By specifying DEFER-NO, the output can begin printing as soon as there is data to print. For example, if you have a 60-page report, the output can begin printing after the first page is complete. Then, by the time the program has created page 60, 20 pages of the report may have already been printed.

Using DEFER-NO might slow down the overall throughput of your printers. No print files can be printed until the print file using the DEFER-NO parameter has been printed. If you use the default (DEFER-YES), other print files can begin printing while yours is being spooled.

Assigning Priorities to Printed Output

You can use the PRINTER OCL statement to assign a priority to printed output. Printed output can be assigned a priority of 0 through 5. The highest priority is 5; that is, any output with a priority of 5 is printed first. Normally, when output is printed, it is assigned a priority of 1. A priority of 0 means that the output will be held on the spool file until an operator releases the output by entering the RELEASE control command. The PRINTER OCL statement in the *System Reference* manual has more information about assigning priorities to printer output.

Controlling Printer Functions from Your Applications Program

A PRPQ is available from IBM which allows you to control the following printer functions from your RPG, COBOL, or Assembler programs:

- Lines per inch
- Characters style (font)
- Characters per inch
- Color
- Emphasis
- Forms character (for drawing boxes)
- Page rotation
- Print quality
- Source drawer
- Other functions, by specifying a printer data stream

This PRPQ is called the Intelligent Printer Data Stream Advanced Functions, PRPQ P84094 (5799-CGK) for 5360 and 5362 system units, and PRPQ P84095 (5799-CGL) for 5364 system unit.

Merging Text and Graphics

You can include a graph anywhere in data printed by DW/36, RPG, COBOL, and any other high-level language. The base SSP can invoke the graphics functions of the intelligent printer data stream (IPDS) and allows the merging of a graphics disk file in the output. You can also print a graphic file without including it in other output. These functions work only with output printed on an IPDS printer. The following section discusses:

- How the graphics files are created
- How to merge text and graphics
- How to print just a graphics file
- Programming considerations

Creating Graphics Files

Business Graphics Utilities/36 (BGU/36) is one way of creating a graphics file. BGU/36 is an interactive utility that allows you to design simple business and scientific graphs quickly. BGU/36 also supplies a procedure that allows you to make a graph using data in a report that was made by your application.

The forms generation utility is another way to create a graphics file. This utility reads source specifications describing a form and either prints the form or saves the drawing specifications on a graphics disk file.

The forms generation utility is part of PRPQ P84094 (5799-CGK) for 5360 and 5362 System Units and PRPQ P84095 (5799-CGL) for 5364 System Unit.

Merging a Graphics File with Text

To merge a graphics file with other printed output, you must use a special control record. The control record can be anywhere in the print data produced by a program. The program can be an application program written in any language, DW/36, or DSU. The format of this control record is:

```
#$@INCLGRPH filename,x,y,w,l
```

Notes:

1. *There should be only one blank between "#$@INCLGRPH" and the file name.*
2. *If you use this control record in a DW/36 document, justification should not be active for the section of the document where the control record resides.*

filename The name of the graphics disk file to be included. Date differentiated files are not supported. Only the file with the most current date is used.

x The distance (in inches) from the left edge of the page to the left edge of the graphics area on the page. The default is zero (0).

y The distance (in inches) from the top of the page to the top of the graphics area. The default is zero (0).

w The width of the graphics area in inches. The default is the width of the current page (usually 13.2 inches).

l The length of the graphics area in inches. The default is the length of the current page.

Where the graphics data will be printed on the page is defined by x, y, w, and l. The upper left corner is defined by the x and y. The size of that area is defined by the w and the l.

The graphics data in the file is scaled to fit the graphics area defined by the w and the l.

The x, y, w, and l are specified in any of the following forms:

```
x            x.
xx           xx.
xx.x         x.x
xx.xx        x.xx
```

Note: *Leading zeros are allowed.*

Printing a Graphics File

A utility, $DPGP, is provided to print a graphics file without including that file in other output. This utility is mainly used to print a BGU/36 graphics file because BGU/36 does not directly support IPDS printers.

A PRTGRAPH procedure is provided to use the $DPGP utility. The *System Reference* manual has more information about the PRTGRAPH procedure.

Programming Considerations

$DPGP does not issue any error messages. If there is something wrong (file not on disk, graph file not found, or invalid file name) then the #$@INCLGRPH record is printed instead. It is treated as normal data and printed as is.

If the control record is used in an application program, the characters #$@INCLGRPH should not appear together, in the source program. If they do appear together, and the compiler output is printed on an IPDS printer, the include will be performed at the time the compiler output is printed. To prevent this from happening, break up the #$@INCLGRPH into two character strings and have the program concatenate them when it executes.

The #$@INCLGRPH control record should be in a print record by itself. If anything else is in the print record, it may be considered as a parameter. The parameters should immediately follow each other, separated by commas with no intervening blanks.

Because the include function may be performed by the spool writer, the file should not be protected by resource security. If the file must be protected, include it in a print file that is not spooled.

Included files are deleted from the printer's storage at the start of each page. Includes done on a page must all fit together in the printer's storage. If the printer runs out of storage, you will get an error message.

Chapter 4. Disk Storage

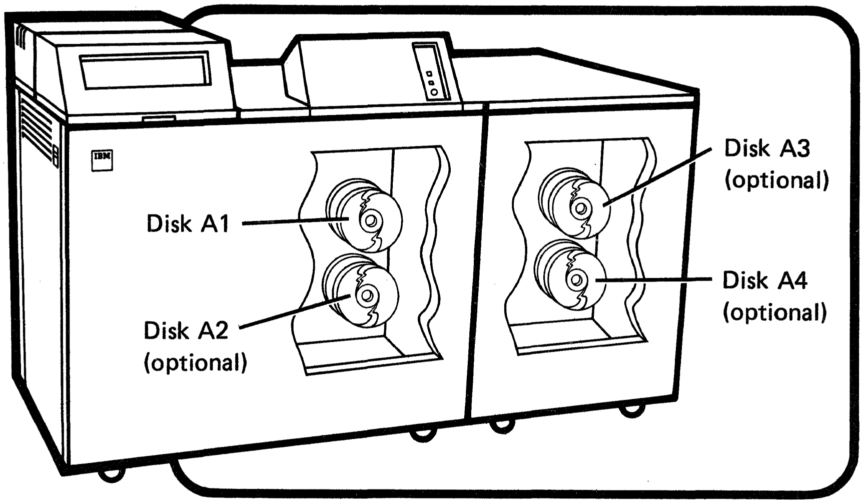
The purpose of this chapter is to:

- Describe disk storage.
- Describe what is stored on disk.
- Provide you with programming tips and techniques regarding disk processing.

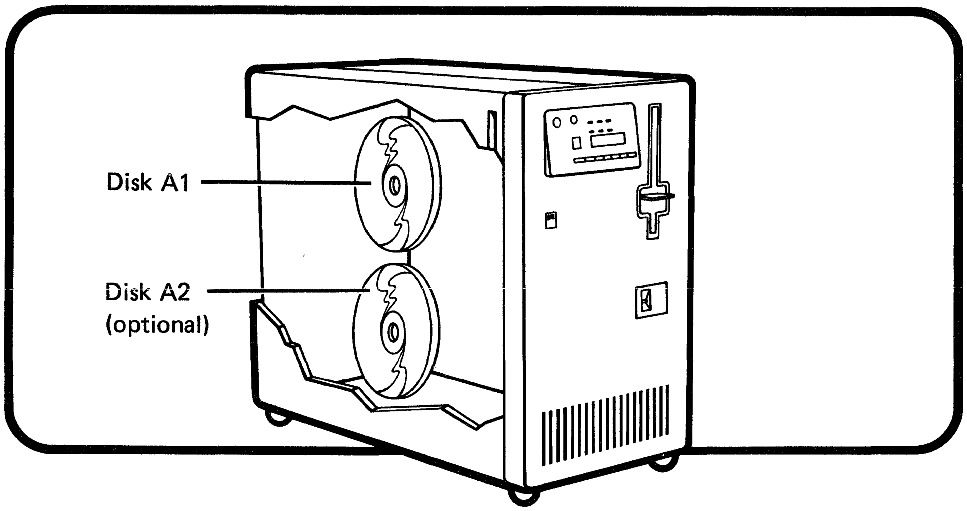
Disk Storage Concepts

A **disk** is a storage device made of flat, circular plates with magnetic surfaces. Programs, files, libraries, and system work areas that are used by the system to process programs are stored on disk. A **disk drive** is the mechanism that reads and writes information on disk. Disk drives are also called **spindles**.

The system can have from one to four disks (indicated by A1 through A4, respectively). The disks are inside the system unit and are not removable.

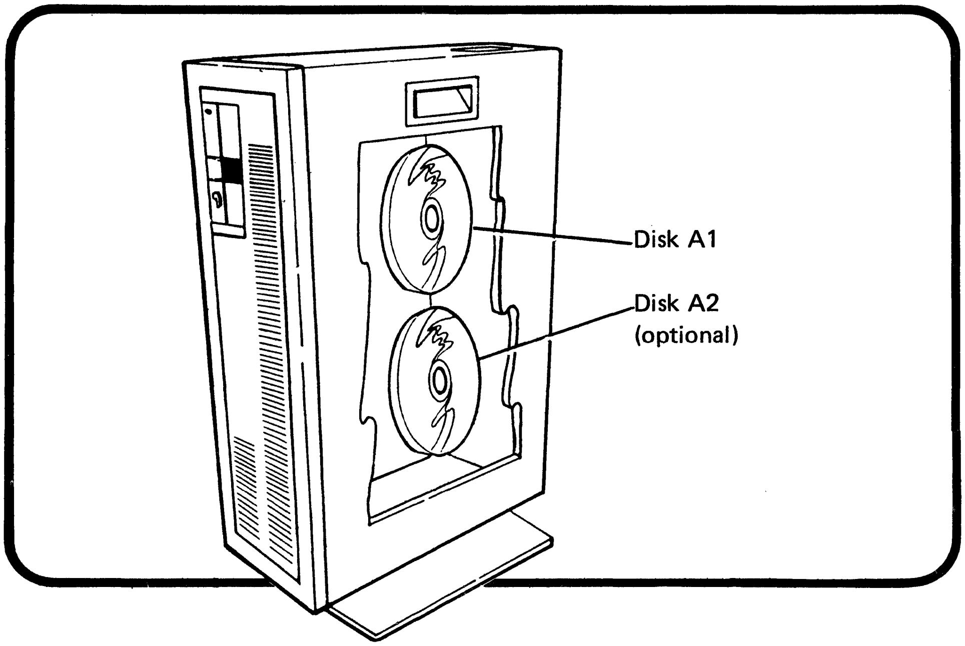


5362 System Unit:



Two disks can be externally attached to the 5362 System Unit and are removable.

5364 System Unit:



The system can have the following combinations of disk drives:

5360 System Unit:

Total Disk Capacity in Megabytes	Number of Drives	Drive Sizes in Megabytes
30	1	30
60	2	30
200	1	200
400	2	200
600	3	200
800	4	200
716	2	358
758	3	200 and 358
1074	3	358
1116	4	200 and 358
1432	4	358

5362 System Unit:

Total Disk Capacity in Megabytes	Number of Drives	Drive Sizes in Megabytes
30	1	30
60	1	60
90	2	30 and 60
120	2	60

5362 System Unit with externally attached disk drives:

Total Disk Capacity in Megabytes	Number of Drives	Drive Sizes in Megabytes
260	2	60 and 200
290	3	30, 60, and 200
320	3	60 and 200
460	3	60 and 200
490	4	30, 60, and 200
520	4	60 and 200

5364 System Unit:

Total Disk Capacity in Megabytes	Number of Drives	Drive Sizes in Megabytes
40	1	40
80	2	40

In addition to the identifiers A1 and A2, the identifier F1 is also used to refer to disk storage in certain operations that are performed with files, libraries, and folders on disk, regardless of which disk they are on or how many disks your system has. For example, the system utility program that lists all files and libraries on disk is run by the following procedure:

```
CATALOG ALL,F1
```

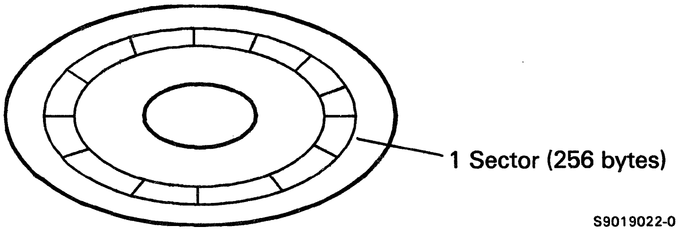
Physical Organization

To specify certain disk operations, you should be familiar with the way data is stored on disk.

Blocks and sectors are the basic units of measurement used to describe disk storage. For example, when you reserve space on the disk for a library, you specify the amount of space in blocks.

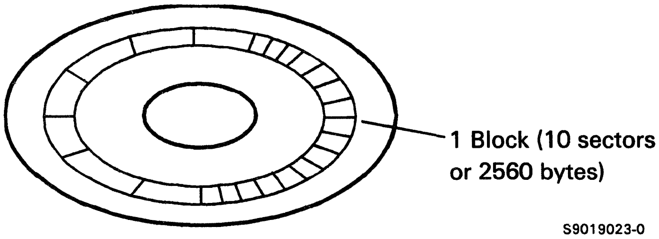
Sectors

Each disk is divided into 256-byte units called sectors. A sector of data is the smallest amount of data that can be either read from or written to the disk.



Blocks

A block consists of 10 sectors or 2560 bytes. Blocks are important because all disk space is measured in blocks. For example, when you create a library, you specify its size in blocks.



You can allocate space for disk files by either blocks or records. When you allocate a file by records, the actual amount of disk space allocated is rounded up to the next full block.

Disk Capacities

The following charts show the number of blocks of disk space available for each disk.

5360 System Unit Disk Capacity in Megabytes	Total Number of Blocks
30	12,049
60	24,098
200	78,204
400	156,408
600	234,612
800	312,816
358	140,218
716	280,436
758	296,626
1074	420,654
1116	436,844
1432	560,872

5362 System Unit Disk Capacity in Megabytes	Total Number of Blocks
30	12,049
60	24,098
90	36,147
120	48,196

5362 System Unit with External Drives Capacity in Megabytes	Total Number of Blocks
260	102,334
290	114,383
320	126,432
460	180,570
490	192,619
520	204,668

The first 650 blocks of the first disk are reserved for the system's use.

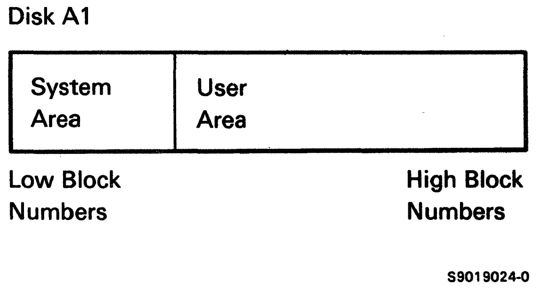
What Is Stored on Disk

This section describes the different types of information that are stored on disk.

Disk A1 is logically divided into two areas: the system area and the user area.

System Area Contents	User Area Contents
Control storage library System library System work files Task work area History file Service log	Spool file Job queue Trace files Dump files User ID file Resource security file Program product libraries User files User libraries User folders

The system area starts at the lowest block numbers.



If your system has more than one disk drive, the other drives contain only user areas.

The following sections describe each type of information on the disk, starting with the system area.

System Area Contents

Control Storage Library

The control storage library contains the part of the SSP that allows the system to perform IPL (initial program load) when you press the Load key on the system control panel. The information in this library also controls the running of the entire system; that is, it controls how the disks are accessed and how information is transmitted to the printers and display stations.

System Library (#LIBRARY)

The system library contains the SSP programs that the system uses to run your jobs (for example, disk data management and printer data management).

The system library also contains SSP procedures and SSP utility programs. These perform common functions such as copying, saving, or restoring files. The system library also contains the system help support.

System Work Files

The system work files maintain a record of information about the system. They are also used when jobs are being run. These files are:

- The master configuration record. This is a description of the program products, display stations, and printers on the system. It also contains such information as the default printer for each display station and the size of the history file.

- The disk volume table of contents (VTOC). This contains a record of all the files, libraries or folders on disk. The information includes the file, library or folder name, the creation date of files, where the file, library or folder is located on disk, and how many blocks the file, library or folder uses.

  Each time a new file, library or folder is created, the information about that file, library or folder is added to the disk VTOC. When a file, library or folder is removed from disk, the corresponding disk VTOC entry is removed.

- The diskette VTOC. This work file contains the VTOC of a diskette when the diskette drive is being used.

Task Work Area

The task work area is the portion of the disk that user programs and the system use to run jobs. The task work area contains:

- Areas for programs or buffers that are swapped out of main storage.

- Work spaces used by the system during job processing.

The size of the task work area is set during system configuration and may change depending upon the system's workload. See the *Changing Your System Configuration* manual for information about the task work area size.

History File

The history file contains the following information:

- OCL and utility control statements from user-written procedures. IBM-supplied SSP procedures do not have statements logged in the history file. (You can also specify that your procedures are not to have statements logged in the history file; see Chapter 18, "Procedures," for information about logging statements.)

- Commands contained in jobs or entered from display stations.

- All messages displayed at a display station.

- All operator responses to messages and prompts.

- The following information about each entry:

  - Display station being used

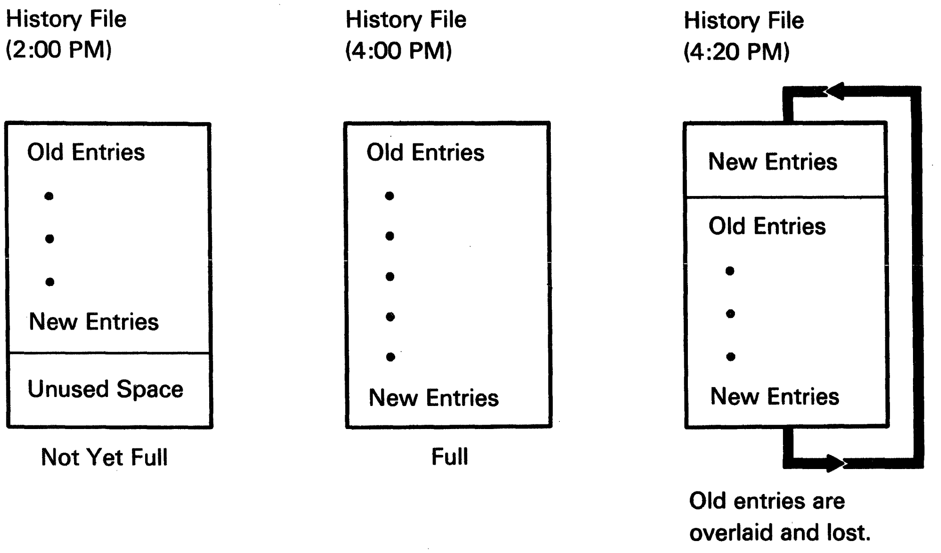
  - Operator's user ID

  - Job name

  - Time and date

The history file is an important tool you can use to review events that have occurred on the system.

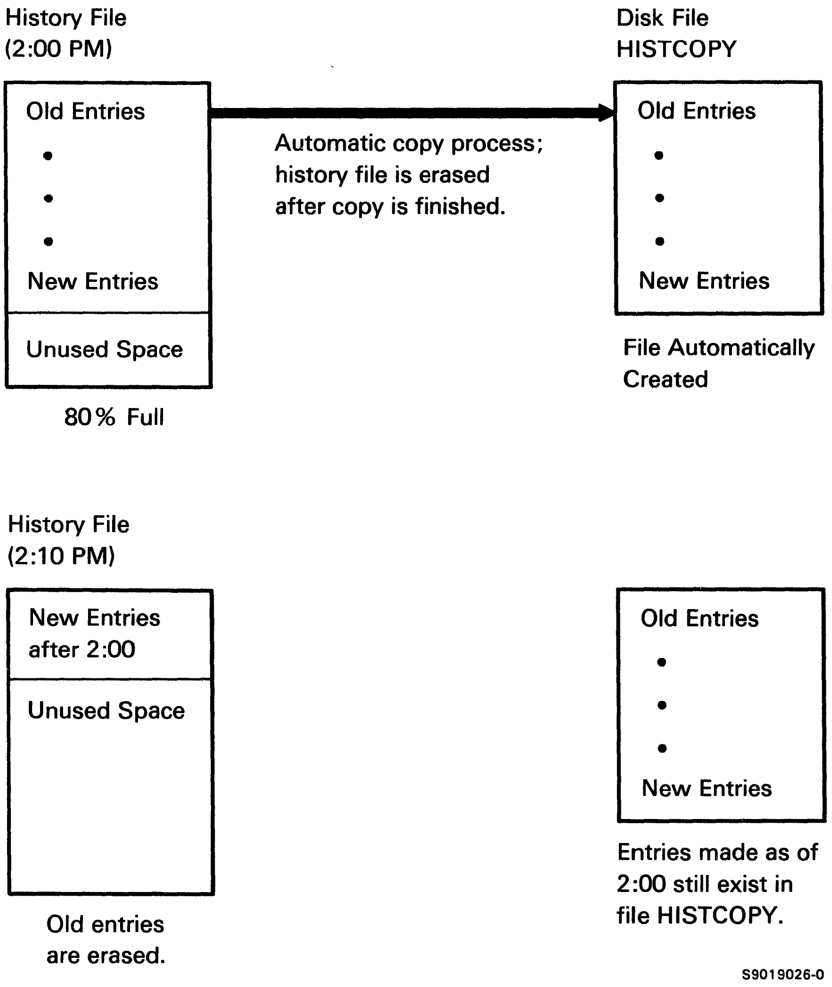
The history file is a fixed size; the size of the history file is set during system configuration. See the *Changing Your System Configuration* manual for information about changing the size of the history file. Normally, when the history file becomes full, the new entries being logged in the history file begin to overlay the older entries. The following diagram shows this process:



S9019025-0

During system configuration, you can select an option that causes the history file to be copied to a disk file (named HISTCOPY). This allows you to always have a copy of the history file for future reference.

When the history file is 80% full, entries are automatically copied to the HISTCOPY file. The following diagram shows this process:



Note the following consideration if you are using the automatic copy option. If the history file once again becomes 80% full **on the same day** and you have not deleted or renamed the HISTCOPY file, the following occurs:

- A message (SYS-1660) will be sent to the system console and to the display station that caused the history file to become 80% full.
- All processing for that display station stops until the file named HISTCOPY has either been removed or renamed. Selecting option 2 in response to the message causes the history file to wrap (new entries overlay older entries), and system processing continues.

To avoid this situation, the system automatically calls a procedure named HISTCOPY after the file named HISTCOPY is created. See "Automatic Copying of History File" on page 4-25 for more information.

Service Log

The service log is a file that contains information about when the system was serviced. The *System Reference* manual has more information about the system service log.

User Area Contents

Spool File

The spool file contains printed output that has been stored on disk for later printing. For more information about the spool file, see Chapter 3, "Printed Output."

Job Queue

The job queue is a list, in a disk file, of jobs waiting to be run by the system. For more information about the job queue, see Chapter 17, "Jobs and Job Processing."

Trace Files

Trace files are created when you are tracing system events. The TRACE procedure in the *System Reference* manual has more information about trace files.

Dump Files

Dump files contain selected system data areas (such as main storage or control storage), which are used to solve problems with either IBM-supplied programs or user-written programs. Dump files are automatically created when a job is canceled with a D (for dump) option or when a program ends abnormally. The DUMP procedure in the *System Reference* manual has more information about dump files,

User ID File

The user ID file contains the list of users that can use the system. The file is used by password security. For more information about the user ID file, see the *System Security Guide.*

Resource Security File

The resource security file contains the list of secured files, libraries, folders, folder members, and groups on the system. It also contains a list of users that can access those files, libraries, and folders. For more information about the resource security file, see the *System Security Guide.*

System Message File (#MESSAGE)

The system message file contains messages that have been sent using the MSG control command or the MSG OCL statement. When a message is sent to a display station or user, the message is saved in the system message file. The message is removed from the file when one of the following occurs:

- The message is displayed at a display station.
- An IPL occurs, and the message has been sent to a display station.
- An IPL occurs, the message is more than 7 days old, and the message has been sent to a user.
- The message is canceled using the MSGFILE procedure.

You can use the MSGFILE procedure to control the size and location of the message file. The *System Reference* manual has more information about the MSGFILE procedure.

Program Product Libraries

These libraries contain all the programs necessary for you to compile your programs or use the utility programs. All display formats, messages, procedures, and programs needed by the language compilers and the utility programs are stored in these libraries. Each programming language or utility has its own separate library. The following is a list of these libraries:

Program Product	Library Name
BASIC	#BLLIB
BASIC help text	#BLHPLIB
COBOL	#COBLIB
RPG II	#RPGLIB
Assembler	#ASMLIB
FORTRAN IV	#FORTLIB
Data file utility (DFU)	#DFULIB
Screen design aid (SDA)	#SDALIB
Source entry utility (SEU)	#SEULIB
Work station utility (WSU)	#WSULIB
Development Support Utility (DSU)	#DSULIB #DSULB1 #DSULB2
Business Graphics Utilities/36 (BGU)	#BGULIB
DW/36 (DisplayWrite/36)	#TULIB
Personal Services/36	#OFCLIB
Office Management Service (OMS)	#TMSLIB
Text Management Service (TMS)	#TMSLIB
Advanced Printer Function (APF)	#APFLIB
Query/36	#QRYLIB
PC Support/36	#IWLIB
Character generator utility (CGU)	#CGULIB
Ideographic sort program	#SRTXLIB
Local Area Networking	#LANLIB

The sizes of these libraries are shown in the *Changing Your System Configuration* manual.

User Files

User files contain the data your programs need. For more information about files, see Chapter 8, "Files."

User Libraries

User libraries contain the programming information for your jobs. For more information about libraries, see Chapter 9, "Libraries."

User Folders

Folders contain members created by IDDU (the interactive data definition utility), DW/36, and Personal Services/36. For more information about folders, see Chapter 10, "Folders."

Programming Guidelines for Disk

This section describes several functions that are available to manage or control the disk. Generally, the functions are:

- Placing files and libraries on disk
- Reorganizing disk space
- Using disk cache

Placing User Files and User Libraries on Disk

The system can automatically place user files and libraries on disk, or you can control their location by specifying a disk spindle or block number when you create the file or library. The location of your files and libraries on disk can affect the performance of your system. Generally, having the system automatically place the files and libraries is the most efficient way.

User folders are automatically placed on disk by the programs that create them.

Automatic Placement on Disk

If you have two or more disk drives on your system, you can let the system automatically place files, libraries, and folders on the disk that is used the least. By placing them on the disk that has the least amount of use, the use of the disks is balanced, and the performance of the system can be improved.

The system has two sets of counters for each disk. These counters measure the number of disk reads, disk writes, and scans for that particular disk. One set of counters, which is reset every hour, measures current system activity and is used by the system to place scratch files and job files. The system also uses another set of counters, which measures past system activity, to place resident files, libraries, and folders.

For example, if some disks are particularly busy and a new file is created, the system attempts to place the file on the disk that is used the least.

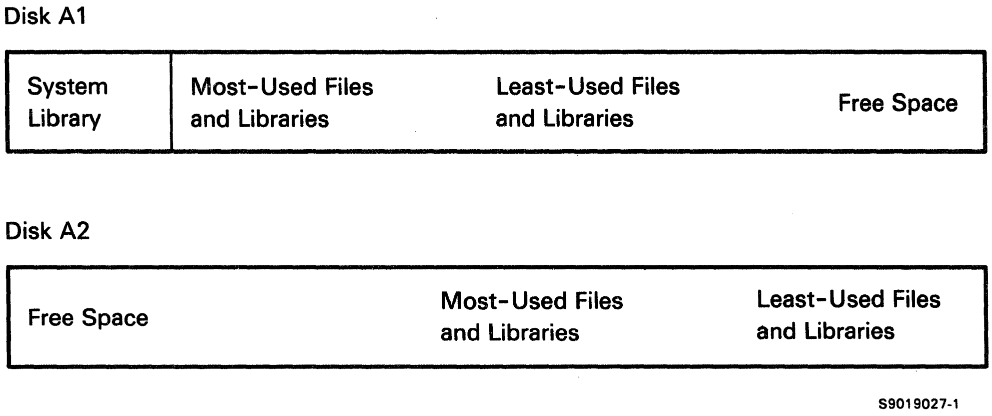
By placing the file on the least used disk, performance of the system can increase because more input/output activity will be directed to the disk that is least used. This helps balance the work each disk has to do while jobs are processing.

If you have two or more disks and decide not to let the system automatically place your files, libraries, and folders on disk, you should:

- Minimize the time the system has to search the disk for data.
- Place files, libraries, and folders on disk so that your programs process the data on disk as efficiently as possible.
- Try to balance the use of the disks.

The system measurement facility (SMF) can help you measure the use of your disks. The *System Measurement Facility Guide* has more information about running SMF.

Example: The following diagram shows some suggestions for placing files, libraries, and folders on the system if you have two disks.


Disk A1
System Library
Most-Used Files and Libraries
Least-Used Files and Libraries
Free Space
Disk A2
Free Space
Most-Used Files and Libraries
Least-Used Files and Libraries
S9019027-1

To reduce the time the system spends searching for information on disk, you should place your most-used files, libraries, and folders following the system library on disk A1. Following the most-used, place your least-used files, libraries, and folders.

On disk A2, you should place the least-used files, libraries, or folders at the high block number locations. Then place the most-used files, libraries, or folders next, leaving the continuous disk space at the low block numbers. Thus space is available when large files, libraries, or folders are created.

To move folders, you can use the MOVEFLDR procedure. The *System Reference* manual has more information about the MOVEFLDR procedure.

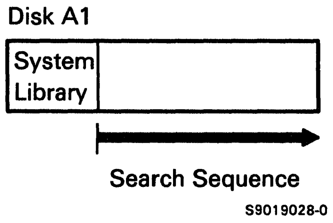
Placement By Spindle Preference

If your system has two or more disk drives, you can specify the disk identifier (A1, A2, A3, or A4) for the disk that is to contain a particular file or library. If the file or library will not fit on the disk you selected, the system tries to place it on another disk. The system begins searching for disk space on the least used disk.

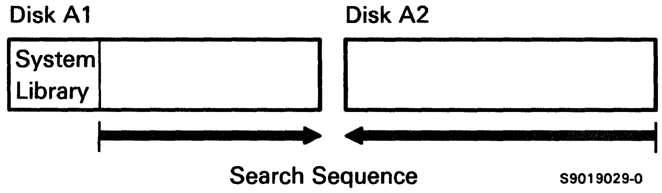
If the system places your file or library on another disk, the system keeps a record of the disk you preferred. If enough space becomes available on the preferred disk, the system moves the file to that disk (for example, during the COMPRESS or RESTORE procedure).

The system searches the disks in a particular direction: either from high blocks to low blocks or from low blocks to high blocks, depending upon the number of disks in the system.

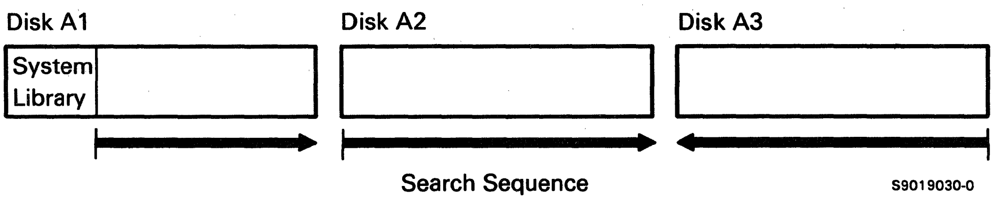
For example, if you specify that you want a particular file or library placed on disk A1, the system begins searching for continuous disk space beginning at the end of the system library (#LIBRARY) and ending at the last block number of disk A1.



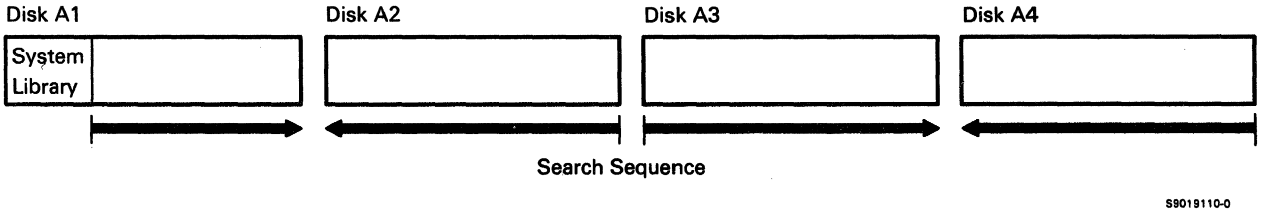
For a system with two disks, the search sequence is:



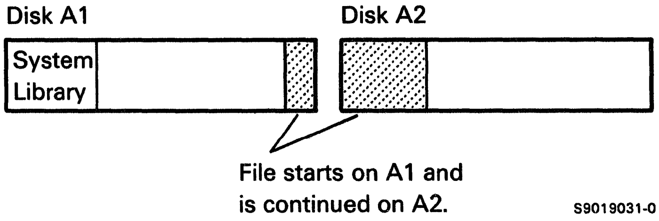
For a system with three disks, the search sequence is:



For a system with four disks, the search sequence is:



If you have two or more disks and there is not enough continuous disk space to place all of a file on one disk, the system can place part of the file on two disks. For example:



Placement By Block Number Location

You can specify the block number on disk where you want your file or library placed. You can use the CATALOG procedure to determine where blocks of disk space are available. Files and libraries cannot be placed in block number locations that are already being used by the system.

If you are creating extendable files using block number locations and the file is extended, the system might move the file to another area on disk. This will change the original block number location. The COMPRESS procedure may also move a file from its original block number.

The following diagram shows the beginning and ending block number locations for the disks used by the system.

For the 5360 System Unit:

Disk Capacity (in megabytes)	Disk Drive A1 Block Numbers	Disk Drive A2 Block Numbers	Disk Drive A3 Block Numbers	Disk Drive A4 Block Numbers
30	0 to 12,048			
60	0 to 12,048	12,049 to 24,097		
200	0 to 78,203			
400	0 to 78,203	78,204 to 156,407		
600	0 to 78,203	78,204 to 156,407	156,408 to 234,611	
800	0 to 78,203	78,204 to 156,407	156,408 to 234,611	234,612 to 312,815
358	0 to 140,217			
716	0 to 140,217	140,218 to 280,435		
758 (358+400)	0 to 78,203	78,204 to 156,407	156,408 to 296,625	
1074	0 to 140,217	140,218 to 280,435	280,436 to 420,653	
1116 (716+400)	0 to 78,203	78,204 to 156,407	156,408 to 296,625	296,626 to 436,843
1432	0 to 140,217	140,218 to 280,435	280,436 to 420,653	420,654 to 560,871

For the 5362 System Unit:

Disk Capacity (in megabytes)	Disk Drive A1 Block Numbers	Disk Drive A2 Block Numbers	Disk Drive A3 Block Numbers	Disk Drive A4 Block Numbers
60	0 to 24,097			
90	0 to 12,048	12,049 to 36,146		
120	0 to 24,097	24,098 to 48,195		

For the 5362 System Unit with externally attached disk drives:

Disk Capacity (in megabytes)	Disk Drive A1 Block Numbers	Disk Drive A2 Block Numbers	Disk Drive A3 Block Numbers	Disk Drive A4 Block Numbers
260 (60+200)	0 to 24,097	24,098 to 102,333		
290 (30+60+200)	0 to 12,048	12,049 to 36,146	36,147 to 114,382	
320 (60+60+200)	0 to 24,097	24,098 to 48,195	48,196 to 126,431	
460 (60+200+200)	0 to 24,097	24,098 to 102,333	102,334 to 180,569	
490 (30+60+200+200)	0 to 12,048	12,049 to 36,146	36,147 to 114,382	114,383 to 192,618
520 (60+60+200+200)	0 to 24,097	24,098 to 48,195	48,196 to 126,431	126,432 to 204,667

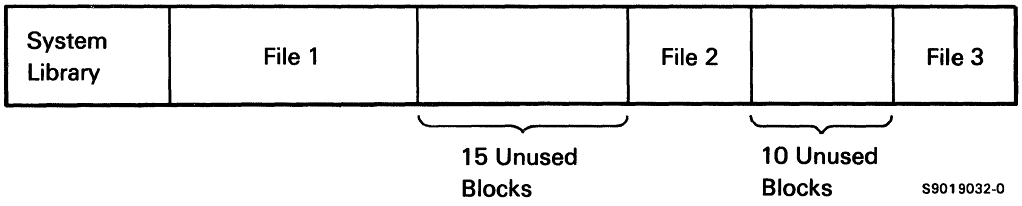
The first 650 blocks of the first disk (drive A1) are reserved for the system's use.

Reorganizing Disk Space

Reorganizing is the collection of free space on each disk to create continuous free space for new files, libraries, or folders.

You can create new files, libraries, or folders, only when enough continuous space on the disk exists for each new file, library, or folder. When you are creating and deleting files, there is a chance that the disk space will be fragmented and that there will not be enough continuous space to create a new file, library, or folder. For example, you want to create a file that takes 20 blocks of space on disk A1 and the disk space looks like the following (assume that A1 is the only disk):

Disk A1



The disk contains two areas of unused space: one is 15 blocks; the other is 10 blocks. Neither area is large enough to contain a 20-block file.

In this example, before you can create a file, library, or folder, you must do one or more of the following:

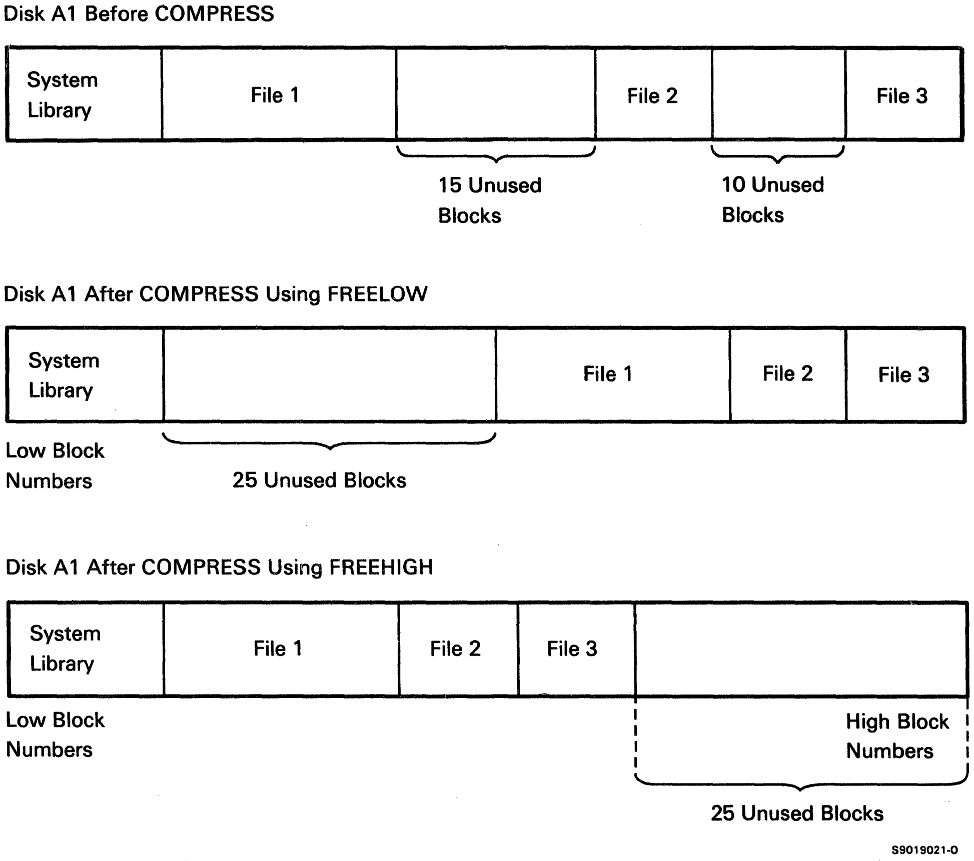
- Delete some of the files, libraries, or folders by using the DELETE procedure.
- Move the files, libraries, or folders together and collect all unused space into one area, by using the COMPRESS procedure.
- Remove some files, libraries, or folders from the disk. You could relocate them on another disk, copy them to diskettes, or copy them to tape.

If you do not want to use the DELETE procedure to make more disk space available, use the COMPRESS procedure. The COMPRESS procedure gathers all free space within the user area of each disk into an area either at the beginning of the disk or at the end of the disk.

You can specify one of two options when you run the COMPRESS procedure:

COMPRESS Procedure Options	**Description**
FREELOW	All available free space is collected at the beginning of the disk (at the **low** block numbers).
FREEHIGH	All available free space is collected at the end of the disk (at the **high** block numbers).

The following diagrams show the two options of the COMPRESS procedure:



Disk A1 Before COMPRESS

Disk A1 After COMPRESS Using FREELOW

Disk A1 After COMPRESS Using FREEHIGH

The *System Reference* manual has more information about running the COMPRESS procedure.

Disk Cache

Disk cache is a main storage buffer between the disk storage devices and the user's main storage buffer. To reduce the time the system spends getting information from disk, you can create a disk cache. With part of the existing storage devoted to the cache and without additional storage, a decrease in performance is possible. When the CACHE procedure is used to stop the cache, the storage is available for general use.

Disk cache is divided into pages. Cache size and cache page size are specified in kilobytes, and can be changed at the system console using the CACHE procedure.

If significant sequential disk processing is expected, the cache page size can be made larger to keep the number of disk accesses smaller. If significant random disk processing is expected, the cache page size can be made smaller to reduce the time the system has to search for data. The default cache page size is large enough for a wide variety of system activity. The following discusses disk cache considerations.

Disk Cache Considerations

When using the disk cache, consider the following:

- A read operation from an area **not** in the cache causes a whole page of disk data to be read into the cache.

- A read operation from an area that **is** in the cache retrieves the data from the cache rather than from the disk.

- A write operation to an area that **is** in the cache updates the cache as well as the disk data.

- A write operation to an area that is **not** in the cache does not put that data into the cache.

The *System Reference* manual has more information on the CACHE procedure.

Programming Guidelines for Disk Storage

This section describes several functions that are available to manage or control disk storage; for example, listing the files and libraries stored on disk and listing, copying, or erasing the history file.

Listing the Disk Volume Table of Contents

You list the table of contents of the disk by using the CATALOG procedure. The listing shows the names and locations of the files and libraries on disk, as well as the locations of unused space. The *System Reference* manual describes the CATALOG procedure.

Measuring Disk Activity

To measure the disk activity, you use the system measurement facility (SMF). You use three procedures (SMFSTART, SMFSTOP, and SMFPRINT) to run SMF. You can also enter the SMF procedure command to display a menu that allows you to choose the SMF procedure you want to run. SMF is described in the *SMF Guide*, which tells you how to use the printed results of SMF as a guide to deciding how to manage the disks.

Changing the Size of System Files

You can change the size of some of the system files (for example, the task work area or the history file), by using the CNFIGSSP procedure. The *Changing Your System Configuration* manual has information about changing the sizes of these files.

Service Log Procedures

To add an entry to the system service log, you use the SERVLOG procedure. To list the entries contained in the service log, you use the DUMP procedure. Both these procedures are described in the *System Reference* manual.

Listing, Copying, or Erasing History File Entries

You can display, print, copy, or erase entries in the history file by using the HISTORY procedure. You can select entries to be processed by user ID, display station, procedure name, time, and date. You can copy the history file to a disk file in either of two formats:

- The original history file format. This allows you to list that file later (using the HISTORY procedure) or to save those entries on diskette (using the SAVE procedure).
- The format of files created by the COPYPRT procedure. This allows you to use the history file as input to another program. You can also display or print the entries (using the COPYPRT procedure).

These procedures are described in the *System Reference* manual.

Automatic Copying of History File

During system configuration, you specify whether the history file should be automatically copied. The history file is copied to a disk file named HISTCOPY.

As part of the automatic history file copy process, a procedure named HISTCOPY is automatically started by the system after the file HISTCOPY has been created. You can change or edit this procedure so that it does what you want; for example, you can have it rename the file HISTCOPY. The HISTCOPY procedure is stored in the system library. The *System Reference* manual has more information about the HISTCOPY procedure.

Example of HISTCOPY Procedure

This example gives each automatic copy of the history file a unique name. This example shows the statements you could add to the HISTCOPY procedure in the system library (#LIBRARY). The files are named:

```
HIST.01
HIST.02
HIST.03
  .
  .
  .
```

You could then save the files on diskette for later use.

The sample procedure is shown below. You can use the Development Support Utility (DSU) or source entry utility (SEU) to add these statements to the HISTCOPY procedure in #LIBRARY. The *SEU Guide* and *DSU Guide* have more information about SEU and DSU.

```
* HISTCOPY PROCEDURE
* When your system is configured to periodically save the History File,
* two actions occur each time the History File becomes 80% full:
* 1) The System History file is copied to the user file named "HISTCOPY".
* 2) The control is passed to this HISTCOPY PROCEDURE.
*
* The purpose of the sample HISTCOPY procedure (given below)
* is to periodically rename the current "HISTCOPY" file to
* a new file name.  The new file names are: HIST.01, HIST.02,
* up to HIST.99.
*
* Parameter 1 is used as a counter.
// EVALUATE P1=0
*
* Check the file names HIST.nn
*
// TAG LOOP
*      Increment the counter
// EVALUATE P1,2=?1?+1
*      Check for the file name.
*      If the file does exist, go back and get a new file name
// IF DATAF1-HIST.?1? GOTO LOOP
*
* File HIST.nn does not exist, rename HISTCOPY to HIST.nn
*
RENAME HISTCOPY,HIST.?1?
*
* Send a message to the system operator
*
// MSG ,File HISTCOPY was rename to file HIST.?1? on ?DATE? at ?TIME?.
```

Note that a file named HIST.02 is created only if a file named HIST.01 is found on the disk.

Chapter 5. Diskette Storage

The purpose of this chapter is to:

- Suggest how you can use diskette storage.
- Describe the types of diskettes you can use with the system.
- Describe the different formats you can use to store data on diskette.
- Provide you with programming tips and techniques regarding diskette processing.

Uses of Diskettes

Primarily, diskettes are used to create backup copies of information. You can also use diskettes for offline storage of files and libraries or to transfer information to other systems or devices.

Diskette Types and Storage Capacities

System/36 supports several types of diskettes:

- Diskette 1 is an 8-inch single-sided, single-density diskette.
- Diskette 2D is an 8-inch double-sided, double-density diskette. Double-density means that one side of a diskette 2D can store **twice** as much information as a diskette 1.
- Diskette 2HC is a 5-1/4 inch double-sided, high-capacity diskette. Diskette 2HC is also referred to as a 96 TPI (tracks per inch) diskette, 2QD (quad-density) diskette, and as a 2HD (high-density) diskette.

The diskette you use is determined by your needs and the system you have.

These types of diskettes must be initialized before they can be used. The INIT procedure is used to do this. The INIT procedure, through its FORMAT and FORMAT2 parameters, allows you to specify storage capacity.

The following table lists the storage capacities and formats of the different diskettes you can use.

Diskette Type	INIT Procedure Parameter	Number of Bytes per Sector	Number of Sectors	Total Bytes
1	FORMAT	128	1924	246,272
1	FORMAT2	512	592	303,104
2D	FORMAT	256	3848	985,088
2D	FORMAT2	1024	1184	1,212,416
2HC	FORMAT	256	3848	985,088
2HC	FORMAT2	1024	1184	1,212,416

Initializing diskettes is discussed in the section "Programming Guidelines for Diskette Processing" on page 5-10.

Depending on the sizes of your files, libraries, and folders, you can place several files on one diskette or a large file on several diskettes.

Diskette Exchange Formats

An **exchange format** is a set of rules about the content of the header record of the diskette and the physical organization of the diskette. Exchange formats are provided so you can exchange data between System/36 and other systems. The formats you choose depend on which system you exchange your data with. System/36 supports these types of exchange formats.

- Basic data exchange (1)
- H-data exchange (2D and 2HC)
- I-data exchange (1 and 2D)
- Special E-format (2D)

For information about the exchange format required to exchange data with another system, see the appropriate manual for that system.

Basic Data Exchange Format

Basic data exchange format files have requirements assuring that diskettes may be exchanged between systems capable of reading and writing diskette 1.

Basic data exchange has the following characteristics:

- The diskette sector is 128 bytes (diskette 1).
- The records are unblocked and unspanned.
- All records in the file must be the same length. The record length of the file you put on diskette must be less than or equal to 128 (diskette 1).
- The file name must be 8 characters or less.

You can use the TRANSFER procedure or the POST procedure to read and write diskettes in this format. The $MAINT utility program allows you to read and write library members in basic data exchange format (using record mode). The *System Reference* manual has more information about the TRANSFER procedure, the POST procedure, and the $MAINT utility program.

H-Data Exchange Format

The H-data exchange format files have requirements assuring that diskettes may be exchanged between systems capable of reading and writing the diskettes 2D and 2HC.

Note: H-data exchange is also called basic data exchange on System/36.

H-data exchange has the following characteristics:

- The diskette sector is 256 bytes.
- All records in the file must be the same length. The maximum record length is 256 bytes.
- The records are unblocked and unspanned.
- The file name must be 8 characters or less.

You can use the TRANSFER procedure or the POST procedure to read and write diskettes in this format.

I-Data Exchange Format

The I-data exchange format files have requirements assuring that diskettes may be exchanged between systems capable of reading and writing diskettes 1 and 2D.

I-data exchange has the following characteristics:

- The diskette sector is 128 bytes or 512 bytes (diskette 1), or 256 bytes or 1024 bytes (diskette 2D).

- All records in the file must be the same length. The record length of the file must be less than or equal to 4096.

- Records are blocked and can span diskette sectors. That is, several records and parts of records can be placed in a diskette sector, or a record can extend from one sector to another. However, records cannot span diskette volumes.

- The file name must be 8 characters or less.

You can use the TRANSFER procedure to read and write diskettes in this format. The *System Reference* manual has more information about the TRANSFER procedure.

Special E-Format

The E-general exchange format files have requirements that force the using system to examine each field in the header label. None of the characteristics can be assumed or summarized.

System/36 uses a form of E-exchange format called special E-format to communicate with some point-of-sale terminals such as the IBM 5260 Retail System. You can use the POST procedure to read diskettes in this format. The *System Reference* manual has more information about the POST procedure.

How Information Is Stored on Diskette

When you save a file or library on diskette, the system creates a file on diskette which contains the file or library that you saved. When you save a folder on diskette, the system creates one or more diskette files which contain the folder that you saved. The diskette volume table of contents (VTOC), which is similar to the disk VTOC, includes the following information about the diskette file:

- Name of the file or files.
- Type of file that was created.
- Date the file was created.
- Size of the file.
- Record length of the file.
- File's expiration date.
- Sequence number of the diskette, if the file is contained on more than one diskette.

This is only a partial list; the CATALOG procedure in the *System Reference* manual has a complete list.

Types of Diskette Files

The more common types of diskette files you can create with the system are shown in the following table:

File Type	Description
COPYFILE	Created when you use the SAVE procedure (or the $COPY utility program) to save a disk file. The format of the diskette file is unique to System/36 and System/34. The information can be exchanged only with another System/36 or System/34.
EXCHANGE	Created when you use the TRANSFER or POST procedure (or the $BICR utility program) to copy a disk file in basic data exchange format.
IFORMAT	Created when you use the TRANSFER procedure (or the $BICR utility program) to copy a disk file in I-exchange format.
LIBRFILE	Created when you use the FROMLIBR procedure (or the $MAINT utility program) to copy one or more library members.
SAVELIBR	Created when you use the SAVELIBR procedure (or the $MAINT utility program) to save an entire library. The format of the diskette file is unique to System/36.
SAVEFLDR	Created when you use the SAVEFLDR procedure (or the $TMSERV utility program) to save an entire folder. The format of the diskette file is unique to System/36.
ARCHIVE	Created when you use the ARCHIVE procedure (or the $TMSERV utility program) to save a folder member.

This is only a partial list; the CATALOG procedure in the *System Reference* manual has a complete list.

SAVEFLDR Files on Diskette

When you use the SAVEFLDR procedure, the system places the folder and its extents in separate files on diskette: one file for the folder directory and one file for each extent (for a description of these areas, see "Folder Layout" on page 10-3). In order to give each diskette file a unique file name, the system uses the following naming convention:

- The diskette file name for the directory is always the name of the folder.
- For each extent, the file name is in the form *NAMEx.yy*, where:
  - *NAME* is the first four letters of the folder name.
  - *x* is a sequence number. If this folder has the same first four letters and the same creation date as another folder stored on this diskette, this number indicates which folder was stored first.
  - *yy* is the sequence number of the folder extent. This is a number between 01 and 99; it is always 99 for the last extent in the folder. For example, if the folder has three extents, the last-created extent would be numbered 99; if the folder has only one extent, the sequence number for that extent would be 99.

In the following example, folder TXTLETR has only one extent:

File Name	Description
TXTLETR	Folder directory
TXTL0.99	First and only folder extent

In the next example, folders LETTER1 and LETTER2 are being saved on the same diskette on the same day. LETTER1 has two extents; LETTER2 has only one.

File Name	Description
LETTER1	Directory of LETTER1
LETT0.01	First extent of LETTER1
LETT0.99	Second (and final) extent of LETTER1
LETTER2	Directory of LETTER2
LETT1.99	First and only extent of LETTER2. Because the first four letters of LETTER1 and LETTER2 are the same, and because they have the same creation date, the sequence number 1 is given to LETTER2's extent file.

Diskette Data Compression

Diskette data compression can enhance the performance of the SAVE/RESTORE or SAVEFLDR/RESTFLDR procedure by compressing the duplicate character strings in your data files and folders.

If you have files and folders with many duplicate characters, diskette data compression may reduce the number of diskettes you need and the amount of time needed to process your data.

To run diskette data compression for data files, you specify the COMPRESS parameter in the SAVE procedure; for folders you specify the COMPRESS parameter in the SAVEFLDR procedure. When you restore the compressed files or folders, decompression is automatically performed.

Data compression can be performed only on 2D and 2HC diskettes initialized to FORMAT2.

Notes:

1. *If your main storage size is 128K bytes, you may not have enough storage to run diskette data compression unless you make space by reducing the system work load. For example, you could reduce the trace file size and the number of programs running.*

2. *For information about the storage requirements, see "Control Storage Requirements" in Chapter 17.*

Diskette File Expiration Dates

When you copy a file, library, or folder to diskette, one of the parameters you can specify is the number of **retention days**. This parameter specifies how long the system should protect the diskette file. The system uses the value specified for retention days to calculate the **expiration date**.

The calculation is:

Expiration =	Current +	Retention
Date	Date	Days

If you specify 999 for the retention days parameter, the file is considered permanent and will never automatically expire.

The system examines the expiration date each time you create a file on diskette. If the date has passed, the system writes over the file. For example, a diskette file named FILE1 has an expiration date of 14 July 1983. If you copy another file to that diskette **before** 14 July 1983, FILE1 is still on the diskette. However, if you copy a third file to that diskette **on or after** 14 July 1983, FILE1 will no longer be stored on the diskette; the system automatically erases FILE1 because its retention period has expired.

The FILE OCL statement for diskette files in the *System Reference* manual has more information about diskette file retention.

Programming Guidelines for Diskette Processing

This section describes procedures and techniques you can use to process diskettes.

Only IBM-supplied procedures and programs can use the diskette drive. There is no high-level language support for diskette. In order to have your programs use information on diskette, you must copy the diskette information to a disk file, run your program to use the disk file, then copy the disk file back to diskette.

Preparing Diskettes

You prepare a diskette for use by the system by **initializing** the diskette using the INIT or INITDIAG procedures. Initialization determines the number of bytes that can be stored on each sector of the diskette. Initializing also erases any information on the diskette.

You may or may not have to initialize your diskettes. If you are unsure about the format of your diskettes, use the CATALOG procedure to check the diskettes.

When you initialize a diskette with the INIT procedure, the system allows you to specify identifying information to be placed on the diskette. This identifying information is:

- A 6-character name called the **volume ID**. You specify the volume ID on system procedures to ensure that you are using the proper diskette.
- A 14-character name called the **owner ID**. This can be used to determine the owner of a diskette. The owner ID is not checked by the system procedures, but is displayed by the CATALOG procedure.

When you initialize a diagnostic (microcode) diskette with the INITDIAG procedure:

- Diskettes are 2D or 2HC and 512 bytes.
- You need not specify identifying information to be written on diskette.
- You use the COPYDIAG procedure to copy.
- You cannot use the CATALOG procedure.

The INIT and INITDIAG procedures in the *System Reference* manual have more information about diskette initialization.

Copying, Saving or Restoring Information

The following procedures are supplied by the SSP to let you copy, save, or restore information using diskettes. The *Systems Reference* manual has more information about these procedures.

Copying Information

Procedure	Description
COPYI1	Copies diskette files, an entire diskette, or several diskettes to one or several diskettes.
COPYDIAG	Copies diagnostic (microcode) to other diskettes.
FROMLIBR	Copies one or more library members to diskette. To copy library members to diskette in basic data exchange format, use the $MAINT utility program. The *Systems Reference* manual has more information about the $MAINT utility program.
POST	Copies special E-format diskettes to disk. Also copies disk files to diskette in the basic data exchange format.
TOLIBR	Copies library members from a diskette file to a library.
TRANSFER	Copies disk files to diskette in either basic data exchange or I-data exchange format. Also copies basic data exchange or I-data exchange format diskette files to disk.

Saving Information

Procedure	Description
ARCHIVE	Saves a folder member from disk to diskette
SAVE	Saves disk files on diskette
SAVELIBR	Saves an entire library on diskette
SAVEFLDR	Saves an entire folder on diskette
SAVENRD	Saves a network resource directory on diskette

The system issues an error message during a save operation if a diskette is unusable or if there is an empty-slot condition.

The system considers a diskette unusable if it has:

- Active files (see note)
- Extended VTOC in header label
- Physical damage
- Incorrect format (or does not recognize it)
- Incorrect label
- Incorrect geometry (bytes per sector)
- Incorrect volume ID

The system assumes an empty-slot condition when:

- The system is a single-slot system and the slot is empty.
- The system is a multiple-slot system and there is no diskette in the slot.
- The diskette is inserted wrong.
- The loading mechanism cannot remove the diskette from the magazine.

You need not start the job over again during a save operation if any of these conditions occur or if you have a single-slot system. Recovery information for each condition is listed below.

Unusable diskette condition: When the system issues the error message, you can press the Attn key and request a command display. Then run one of the following procedures to prepare the diskette: the CATALOG, the INIT, or the DELETE procedure.

Empty-slot condition: When the system issues the error message, insert a usable diskette and retry the save operation. You may or may not get an error message. If you do not get an error message, the save operation will continue. If you do get an error message, press the Attn key and request a command display. Then run one of the following procedures to prepare the diskette: the CATALOG, the INIT, or the DELETE procedure.

Single-slot systems: If you do not already have the next diskette prepared when the system issues a message requesting the next diskette, you can press the Attn key and request a command display. Then run one of the following procedures to prepare the diskette: the CATALOG, the INIT, or the DELETE procedure.

Once you have prepared the diskette, you can continue the save operation.

Note: *A diskette that contains active files is generally unusable for a save operation unless:*

1. *All the diskette VTOC entries have been used.*

2. *All the data sectors on the diskette have been used.*

3. *A SAVE ALL operation is being performed.*

4. *It is not the first diskette in the save operation.*

Restoring Information

Procedure	Description
RESTLIBR	Restores an entire library from diskette to disk.
RESTFLDR	Restores an entire folder from diskette to disk.
RESTNRD	Restores a network resource directory from diskette to disk.
RESTORE	Restores disk files from diskette to disk.
RETRIEVE	Restores a folder member from diskette to a folder on disk.

Listing Information from Diskette

To list the names of the files, libraries, and folders on diskette, use the CATALOG procedure.

To list a disk file that was copied to diskette by the SAVE procedure or the $COPY utility program, use the LISTDATA procedure.

If you want to list a library, a basic data exchange file, an I-exchange file, or any other type of diskette file, use the LISTFILE procedure.

The *System Reference* manual has more information about these procedures.

Removing Information from Diskette

To remove information from a diskette, use the DELETE procedure. The *System Reference* manual has more information about this procedure.

Allocating the Diskette Drive to a Job

You can use the ALLOCATE OCL statement to dedicate the diskette drive to a job. For example, you have a procedure that saves three files (the diskette has a volume ID of VOL001):

```
SAVE FILE1,,,VOL001
SAVE FILE2,,,VOL001
SAVE FILE3,,,VOL001
```

Normally, you lose control of the diskette drive between the SAVE procedures. That is, after FILE1 is saved but before FILE2 is saved, another procedure on the system could use the diskette drive. This would make your SAVE FILE2 procedure wait until the other procedure ends.

You can use the ALLOCATE OCL statement to retain control of the diskette drive throughout the three SAVE procedures:

```
// ALLOCATE UNIT-I1
SAVE FILE1,,,VOL001
SAVE FILE2,,,VOL001
SAVE FILE3,,,VOL001
```

To avoid allocating the diskette drive longer than necessary, you should use the DEALLOC OCL statement to deallocate the diskette drive. For example, your procedure would save three files and then run another type of job that did not use the diskette drive. You would use the DEALLOC OCL statement to allow other jobs to use the diskette drive.

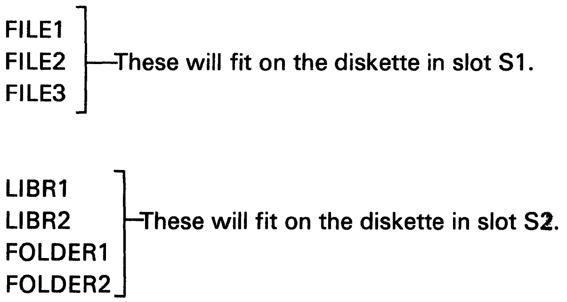
```
// ALLOCATE UNIT-I1
SAVE FILE1,,,VOL001
SAVE FILE2,,,VOL001
SAVE FILE3,,,VOL001
// DEALLOC UNIT-I1
*
// LOAD PROG1
// RUN
```

In this example, if the DEALLOC OCL statement is not specified, the job retains the diskette drive until the procedure ends. While PROG1 is running, other jobs would needlessly be prevented from using the diskette drive. The *System Reference* manual has more information about the ALLOCATE and DEALLOC OCL statements.

Creating a Sequential Set of Files on Diskette

If your system has a diskette magazine drive, you can use the ALLOCATE OCL statement to read or write a sequential set of files on diskette.

Normally, when you save or restore files, libraries, and folders from a set of diskettes, the system starts with the location you specify in the procedure. Assume that you want to save three files, two libraries, and two folders. Also, assume that you have two empty diskettes in slots S1 and S2:


FILE1
FILE2
FILE3
These will fit on the diskette in slot S1.
LIBR1
LIBR2
FOLDER1
FOLDER2
These will fit on the diskette in slot S2.

S9019126-0

When you use the SAVE, SAVELIBR, or SAVEFLDR procedure to save a file, library, or folder on diskette, the location parameter defaults to S1.

If you enter the following procedures:

```
SAVE FILE1,,,VOL001
SAVE FILE2,,,VOL001
SAVE FILE3,,,VOL001
SAVELIBR LIBR1,,VOL001
SAVELIBR LIBR2,,VOL001
SAVEFLDR FOLDER1,,VOL001
SAVEFLDR FOLDER2,,VOL001
```

the files, libraries, and folders are saved on diskette. However, after the diskette in slot S1 becomes full, the system keeps checking that diskette for unused space. This means that when the system saves LIBR1, LIBR2, FOLDER1, and FOLDER2, it wastes time by looking for unused space on the full diskette in slot S1.

Also, if you were using a diskette magazine (which can contain up to 10 diskettes), a lot of time would be wasted if the system were saving several files that used all 10 diskettes. The system would check each diskette for unused space before it saved the next file, library, or folder.

You can use the ALLOCATE OCL statement to instruct the system to continue the operation from its previous ending location. For example:

```
// ALLOCATE UNIT-I1,CONTINUE-YES,AUTO-YES
SAVE FILE1,,,VOL001
SAVE FILE2,,,VOL001
SAVE FILE3,,,VOL001
SAVELIBR LIBR1,,VOL001
SAVELIBR LIBR2,,VOL001
SAVEFLDR FOLDER1,,VOL001
SAVEFLDR FOLDER2,,VOL001
```

Now, the system begins with the diskette in slot S1 (remember, it's the default) and automatically goes to the other slots as needed. That is, after the diskette in slot S1 becomes full, the system no longer checks that slot until the job ends or the DEALLOC statement is processed. The *System Reference* manual has more information about the ALLOCATE OCL statement.

Changing the AUTO/NOAUTO Settings for Procedures

The AUTO/NOAUTO parameters of the diskette procedures specify whether the system should automatically check the next diskette magazine slot for a diskette. The default for the IBM-supplied procedures is AUTO; that is, the system automatically checks the next slot for a diskette.

The ALLOCATE OCL statement allows you to override the AUTO/NOAUTO parameters of procedures. You can also override the AUTO parameter of the FILE OCL statement for diskette files. For example, the SAVE procedure AUTO/NOAUTO parameter defaults to AUTO. You can use the ALLOCATE OCL statement to prevent the SAVE procedure from automatically advancing to the next diskette:

```
// ALLOCATE UNIT-I1,AUTO-NO
SAVE FILE1
```

This causes the system to use only the first slot (S1). The *System Reference* manual has more information about the ALLOCATE OCL statement.

Chapter 6. Magnetic Tape Storage

The purpose of this chapter is to:

- Suggest how you can use magnetic tape storage.
- Describe the capacities of tape reels and cartridges.
- Describe the different formats you can use to store data on tape.
- Provide you with programming tips and techniques regarding tape processing.

Using Tape Storage

Primarily, tapes are used to create backup copies of information. You can also use tapes for:

- Offline storage of files, libraries, and folders.
- Transfer of information to other systems or devices (8809 Tape Drive only).

Tapes must be initialized before they can be used. You can use the TAPEINIT procedure to do this.

Tape Lengths and Storage Capacities

The 8809 Tape Drive records data at 1600 bytes per inch over 9 tracks on the tape. The system supports tape reel sizes from 15.9 cm (6.25 in.) to 26.7 cm (10.5 in.).

The IBM 6157 Tape Drive records data at 8000 bytes per inch over 9 tracks on the tape. The system supports one tape cartridge size.

The following tables list the storage capacities of the different tapes on reel and tapes on cartridge you can use.

Length of Tape on Reel	**Reel Size (Diameter)**	**Megabytes that Can Be Stored (Approximate)**
92 m (300 ft.)	15.9 cm (6.25 in.)	5
183 m (600 ft.)	19.1 cm (7.5 in.)	10
366 m (1200 ft.)	21.6 cm (8.5 in.)	20
732 m (2400 ft.)	26.7 cm (10.5 in.)	40

Length of Tape on Cartridge	**Megabytes that Can Be Stored (Approximate)**
450 ft.	40
550 ft.	48.9

Tape Formats

The tape format describes the contents and characteristics of information stored on the tape. The system initializes tapes differently depending on your system unit and tape drive. This section describes the types of formats you can use.

Tape Format for 8809 Tape Drive

The system initializes tapes on reel in two formats: IBM standard labeled and nonlabeled.

Using IBM standard labeled tapes is recommended. This allows you to easily determine the names of files stored on tape, as well as other information. Also, most system procedures can only operate with IBM standard labeled tapes (for example, SAVE and SAVELIBR).

If you use nonlabeled tapes, **you** have to keep track of the information on the tape, because no labels are provided to identify the files on the tape. These SSP procedures can be used with nonlabeled tapes: TAPECOPY and LISTFILE.

Tape Format for IBM 6157 Tape Drive

The system initializes tapes on cartridge in one format: IBM standard labeled.

This allows you to easily determine the names of files stored on tape, as well as other information.

The TAPECOPY procedure ($TCOPY program) does not support the IBM 6157 Tape Drive (UNIT-TC). All tape files are written in fixed block record format (RECFM-FB). The block length (BLKL) must be a multiple of 512 bytes.

How Information Is Stored on Standard Labeled Tapes

When you save a file or library on a standard labeled tape, the system creates tape marks (TM), and header and trailer labels on the tape before and after the file or library data. The following example shows where these are located on the tape:

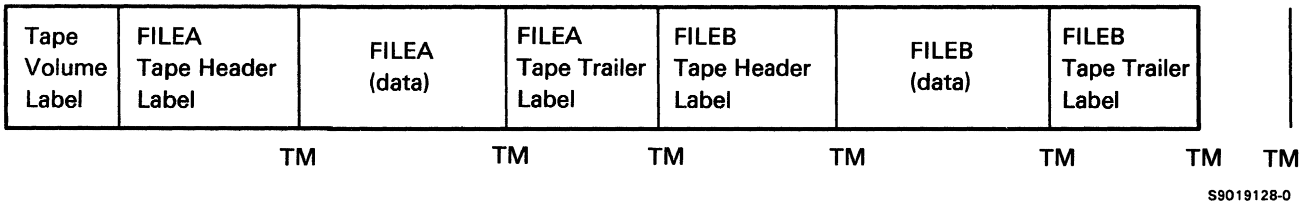


Figure 6-1. Standard Labeled Tape Processing

Header and trailer labels may include the following information about the tape file:

Header 1 Label (Present on all tapes)

- Name of the file
- Date the file was created
- File's expiration date
- File's sequence number (SEQNUM) from the beginning of the tape or the beginning of a sequence of tapes
- Volume sequence number of the tape, if the file is contained on more than one tape (such a file is called a multivolume file)

Header 2 Label (Present on tapes used with 6157 Tape Drive)

- Record length of tape file
- Record format of tape file
- Block length of tape file

User Header Label (Present on System/36 tapes only)

- Type of file that was created
- Size of file

This is only a partial list; the CATALOG procedure in the *System Reference* manual has more information.

How Information Is Stored on Nonlabeled Tapes

When you copy a file to a nonlabeled tape, the system does not create header or trailer labels. You have to keep track of the information on the tape. Each file is assigned a sequence number (SEQNUM) and a tape mark (TM). The following example shows how information is located on the tape.

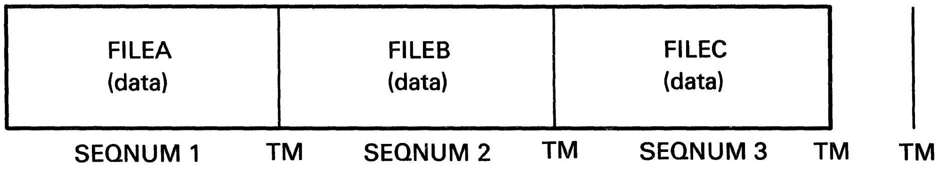


Figure 6-2. Nonlabeled Tape Processing

Note: *Some nonlabeled tapes have a leading tape mark (TM). In order to get the first data file, SEQNUM 2 must be specified to read FILEA data.*

Tape Files

The types of tape files you can create with the System/36 are shown in the following table. Except where noted, the formats of these tape files are unique to this system; the information can be exchanged only with another System/36.

File Type **Description**

COPYFILE Created when you use the SAVE procedure (or the $COPY utility program) to save a disk file.

EXCHANGE Created when you use the TAPECOPY procedure (or the $TCOPY utility program) to save a disk file. This type allows information to be exchanged with another system, not necessarily a System/36.

Note: This type of file is not supported on the IBM 6157 Tape Drive.

LIBRFILE Created when you use the FROMLIBR procedure (or the $MAINT utility program) to copy one or more library members.

SAVELIBR Created when you use the SAVELIBR procedure (or the $MAINT utility program) to save an entire library.

SAVEFLDR Created when you use the SAVEFLDR procedure (or the $TMSERV utility program) to save an entire folder.

ARCHIVE Created when you use the ARCHIVE procedure (or the $TMSERV utility program) to save a folder member from disk to tape.

Exchanging Tape Files with other Systems

To exchange tape files with other IBM systems, you should use IBM standard labeled tapes.

If you exchange tape files with another system, it is possible the tape has been written with volume/file security. More information about security access on standard labeled tapes is found later in this chapter.

Note: Exchanging tape files with other systems is not supported on the IBM 6157 Tape Drive.

You can specify any of the following processing methods when the system reads tapes. These methods are specified in the TAPECOPY procedure. The *System Reference* manual has more information about TAPECOPY.

- IBM standard label processing. Specifies that the header labels are to be used to read the tape.

  Note: Files that have only a header 1 label can be processed if the record processing information is supplied.

- Nonlabeled tape processing. Specifies that the tapes have no labels and that marks on the tape indicate where the files are stored.

- Nonstandard label processing. Specifies that the tapes have labels, but the labels are not IBM standard labels. Only one file can be read from a nonstandard labeled tape.

  The nonstandard labels are ignored, and the tape is read starting from the first mark on the tape to the second mark on the tape. Thus, only the first file is read.

- Bypass label processing. Specifies that the tape has IBM standard labels but that label information is to be ignored. Instead, sequence numbers on the tape are to be used to process the file.

  You can use this method when you do not know the volume ID of the tape or the name of the tape file.

Tape File Expiration Dates

When you copy a file or library to tape, one of the parameters you can specify is the number of **retention days**. This parameter specifies how long the system should protect the tape file. The system uses the value specified for retention days to calculate the **expiration date**.

The calculation is:

Expiration =	Current +	Retention
Date	Date	Days

If you specify 999 for the retention days parameter, the file is considered permanent and will never automatically expire.

When you initialize a tape and you specify date checking, the system examines the expiration date of the **first** file on an IBM standard labeled tape before initializing the tape. If the file has expired, the system writes over the expired file, and any other files that may be on the tape. Also, when the system continues writing a file from one tape to another and files exist on the continued reel, the expiration date of the first file on the continued reel is checked. If the first file has expired and other subsequent files have not expired, **all** the files can be written over.

The FILE OCL statement for tape files in the *System Reference* manual has more information about tape file retention.

Securing Tapes: Write-Enable Ring and Write-Protect Plug

IBM 8809 Tape Drive: Write-enable rings are plastic rings that can be inserted into the tape reel to allow information to be written on the tape.

If you remove the write-enable ring from a tape reel, the information on the tape can only be read; the information is protected from being written over. This is the only way to ensure no active tape files are written over.

IBM 6157 Tape Drive: Write-enable/write-disable is controlled by a write-protect plug on the tape cartridge. The write-protect plug must be set to write-enable to allow information to be written on the tape.

If the plug is set to write-disable, the information can only be read; the information is protected from being written over.

Security on Standard Labeled Tapes

Security access on standard labeled tapes is controlled by a hex code of '00', '40', or 'F0' in the tape volume label (VOL1). If one of these codes is not found, you must be a security officer to continue using the tape. The system checks volume security when processing standard labeled, nonlabeled, nonstandard labeled, or bypass labeled tape, provided a standard volume label exists.

Security access to the files on standard labeled tapes is controlled by a 0, 1 or a 3 in the header label (HDR1). A zero (0) indicates no security access to the files. If the code is a 1, you must be a security officer to read or write to the files. If the code is a 3, you can read the files but must be a security officer to add to the files. File security will be checked when processing standard labeled, or bypass labeled tapes only.

Programming Guidelines for Tape Processing

This section describes procedures and techniques you can use to process tapes.

Only IBM supplied procedures and programs can use the tape drive. There is no high level language support for tape access. In order to have your programs use information on tape, you must copy the tape information to a disk file, run your program to use the disk file, then copy the disk file back to tape.

Preparing Tapes

You prepare a tape for use by the system by **initializing** the tape with the TAPEINIT procedure, which is described in the *System Reference* manual. Initializing can also be used to erase any information on the tape.

You may or may not have to initialize your tapes. If you are unsure about the format of your tapes, use the CATALOG procedure to check the tapes.

When you initialize a tape with the TAPEINIT procedure, the system allows you to specify identifying information to be placed on the tape. This identifying information is:

- A 6-character name called the **volume ID**. You specify the volume ID on system procedures to ensure that you are using the proper tape. It is a good practice to assign unique volume IDs to each tape reel.
- A 14-character name called the **owner ID**. This can be used to determine the owner of a tape. The owner ID is not checked by the system procedures, but is displayed by the CATALOG procedure.

Copying, Saving, or Restoring Information

The following procedures are supplied with the SSP to let you copy, save, or restore information using tapes. Except where noted, these procedures require standard labeled tapes. The *System Reference* manual has more information about these procedures.

Procedure	**Description**
ARCHIVE	Saves folder members from disk on IBM standard labeled tape.
BLDLIBR	Creates a new library on tape or tape cartridge.
FROMLIBR	Copies one or more library members from a library on disk to an IBM standard labeled tape.
JOBSTR	Copies a tape or tape cartridge file that contains one or more procedure members and source members to a specific library. You can also specify the name of the procedure to be run after the members in the tape or tape cartridge file are copied.
RESTLIBR	Restores an entire library from tape to disk.
RESTFLDR	Restores an entire folder from tape to disk.
RESTNRD	Restores a network resource directory from tape to disk.
RESTORE	Restores former disk files from tape to disk.
RETRIEVE	Restores folder members from tape to a folder on disk.
SAVE	Saves disk files on IBM standard labeled tape.
SAVELIBR	Saves an entire library on IBM standard labeled tape.
SAVEFLDR	Saves an entire folder on IBM standard labeled tape.
SAVENRD	Saves a network resource directory on tape.
SECREST	Restores the user identification file or the resource security file from tape to disk.
SECSAVE	Saves the user identification file or the resource security file on tape or tape cartridge.

TAPECOPY Copies disk files to tape in exchange format. Also copies exchange format tape files to disk. This procedure allows you to use both standard labeled tapes and nonlabeled tapes. You can use standard label processing or nonlabeled tape processing while reading or writing tape files. Nonstandard label processing or bypass label processing can only be used while reading tape files.

Note: TAPECOPY is not valid for the IBM 6157 Tape Drive.

TOLIBR Copies library members from a tape file to a library on disk.

If two IBM 8809 Tape Drives are are attached to your system, you can run the following procedures at the same time:

- TAPEINIT
- TAPECOPY
- FROMLIBR
- TOLIBR
- LISTDATA
- LISTFILE
- ARCHIVE
- RETRIEVE

If two IBM 8809 Tape Drives are attached to the system, you cannot run the following procedures at the same time:

- SAVE
- RESTORE
- SAVELIBR
- RESTLIBR
- SAVELIBR
- SAVEFILE
- RESTFILE

Listing Information from Tape

To list the names of the files, libraries, and folders on an IBM standard labeled tape, use the CATALOG procedure.

To list a file that was copied from disk by the SAVE procedure or the $COPY utility program (a COPYFILE), use the LISTDATA or LISTFILE procedure. To list any other type of tape file, use the LISTFILE procedure.

To list information about files on nonlabeled or non-IBM standard labeled tapes, use the DUMP procedure.

The *System Reference* manual has more information about these procedures.

Removing Information from Tape

To remove information from a tape, use the TAPEINIT procedure. The *System Reference* manual has more information about this procedure. You cannot remove a specific file from tape.

Allocating the Tape Drive to a Job

You can use the ALLOCATE OCL statement to dedicate the tape drive to a job.

Assume that you have a procedure that saves three files (the tape has a volume ID of VOL001):

```
SAVE FILE1,,,VOL001,T1
SAVE FILE2,,,VOL001,T1
SAVE FILE3,,,VOL001,T1
```

Normally, you do not retain control of the tape drive between the SAVE procedures. That is, after FILE1 is saved but before FILE2 is saved, another procedure on the system could use the tape drive. This would make your SAVE FILE2 procedure wait until the other procedure ends.

You can use the ALLOCATE OCL statement to retain control of the tape drive throughout the three SAVE procedures:

```
// ALLOCATE UNIT-T1
SAVE FILE1,,,VOL001,T1
SAVE FILE2,,,VOL001,T1
SAVE FILE3,,,VOL001,T1
```

To avoid allocating the tape drive longer than necessary, you should use the DEALLOC OCL statement to deallocate the tape drive. For example, if your procedure would save three files and then run another type of job that did not use the tape drive, you would use the DEALLOC OCL statement to allow other jobs to use the tape drive.

```
// ALLOCATE UNIT-T1
SAVE FILE1,,,VOL001,T1
SAVE FILE2,,,VOL001,T1
SAVE FILE3,,,VOL001,T1
// DEALLOC UNIT-T1
*
// LOAD PROG1
// RUN
```

In this example, if the DEALLOC OCL statement is not specified, the job would retain the tape drive until the job ends. While PROG1 is running, other jobs would needlessly be prevented from using the tape drive.

Notes:

1. *Asynchronous communications lines and/or diskette cannot be allocated at the same time as the 6157 Tape Drive on a 5362 System Unit.*
2. *An 8809 Tape Drive cannot be allocated at the same time as the 6157 Tape Drive on a 5360 System Unit.*
3. *To receive greater benefit from the use of the ALLOCATE OCL statement and the LEAVE parameter with the 6157 Tape Drive, the same tape must be in the drive and the drive door must not have been opened. statement and the LEAVE parameter.*

The *System Reference* manual has more information about the ALLOCATE and DEALLOC OCL statements.

Creating a Sequential Set of Files on Tape

You can use the ALLOCATE OCL statement and the LEAVE parameter in a procedure to read or write a sequential set of files on tape.

Note: *To receive greater benefit from the use of the ALLOCATE OCL statement and the LEAVE parameter with the 6157 Tape Drive, the same tape must be in the drive and the drive door must not have been opened. the drive door must not have been opened to receive greater benefit from the use of the ALLOCATE OCL*

The *System Reference* manual has more information on the ALLOCATE procedure.

Normally, when you save or restore files, libraries, and folders from a set of tapes, the system starts with the location you specify in the procedure.

For example, if you have the following statements in a procedure:

```
* Procedure to save 2 files, 2 libraries, and 2 folders
SAVE FILE1,,,VOL001,T1
SAVE FILE2,,,VOL001,T1
SAVELIBR LIBR1,,VOL001,,,T1
SAVELIBR LIBR2,,VOL001,,,T1
SAVEFLDR FLDR1,,VOL001,T1
SAVEFLDR FLDR2,,VOL001,T1
```

the files, libraries, and folders are saved on tape. However, the tape is rewound to its beginning (for the 6157 Tape Drive, the drive-in-use light does not remain on between jobs). This means that when the next job step starts, the tape has to be positioned to the end of the last file before the next file can be saved. This rewinding and repositioning can waste a lot of time.

You can use the LEAVE parameter to instruct the system to leave the tape positioned after the end of the file just processed. The next job step using tape may take advantage of the LEAVE state and the tape is not rewound (for the 6157 Tape Drive, the drive-in-use light remains on). For example, if you have these statements in a procedure:

```
* Procedure to create a sequential set of files on tape
// ALLOCATE UNIT-'T1/T2'
SAVE FILE1,,,VOL001,T1,AUTO,,LEAVE
SAVE FILE2,,,VOL001,T1,AUTO,,LEAVE
SAVE ALL,,GRPB,,VOL001,GRPB,T1,,AUTO,LEAVE
SAVELIBR LIBR1,,VOL001,,AUTO,T1,,LEAVE
SAVELIBR LIBR2,,VOL001,,AUTO,T1,,LEAVE
SAVEFLDR FLDR1,,VOL001,T1,,AUTO,LEAVE
SAVEFLDR FLDR2,,VOL001,T1,,AUTO,UNLOAD
// DEALLOC UNIT-'T1/T2'
```

the system begins with the first file and automatically continues with the other files and libraries.

The tape continues processing from the previous job step's ending location. If the tape has auto advanced to the next tape unit and the LEAVE parameter is used, the system will keep track of the previous job step's ending unit and tape position. Processing will continue from there. The last job step causes the tape to be unloaded so that it can be removed from the drive. (For the 6157 Tape Drive, the drive-in-use light will be turned off during processing of all // DEALLOC OCL statements.)

Note: The UNLOAD parameter is defaulted to REWIND on the 6157 Tape Drive.

When you are using the LEAVE parameter to read files from IBM standard labeled tapes, the tape's volume ID, the file name, and (if specified) the file's creation date are examined to ensure that the proper file is being read.

If you are using the LEAVE parameters to write to a standard labeled tape, the system will check to make sure the tape is positioned at the end of the tape before writing to the tape.

If a sequence number is specified, the tape is searched from the current position toward the specified sequence number, checking that the sequence number can be written to the tape. If a tape file exists at that sequence number, it is written over with the new tape file.

Notes:

1. *If you try to write over files on the IBM 6157 Tape Drive, a diagnostic message is issued.*
2. *For the IBM 6157 Tape Drive, if the sequence number request is less than the current position, the tape will rewind and search from the beginning of the tape. If the sequence number requested is greater than the current position, the tape is searched forward from the current position.*

If the next job step is reading a standard labeled tape file in bypass label processing, the tape is searched from its current position to the specified sequence number.

If the next job step is reading or writing to a nonlabeled tape file and no sequence number is specified, the tape is read or written to without any checking. If a sequence number is specified, the tape is rewound and then the sequence number located.

If the next job step is reading a nonstandard labeled tape file, the tape is rewound and the first file's data is read.

The procedure control expressions VOLID and DATAT cause the tape to rewind and then search from the beginning of the tape.

You should also allocate the tape drive to your job when you are using the LEAVE parameter. This ensures that your use of the tape drive is not interrupted by another job, thus preserving your tape's position. See "Allocating the Tape Drive to a Job" on page 6-12 for information about allocating the tape drive to a job. LEAVE information is maintained by the system from job step to job step but is ***not*** passed from job to job.

Chapter 7. Designing Records

A **record** is a collection of fields that is treated as a unit. This chapter describes how to design records, and describes the types of data you can store in records.

Identifying Required Fields

The applications determine what data is needed in the records. That is, the records must contain all the fields required for input to or output from applications that use the file.

In addition, the records might require any of the following fields:

- Keys or relative record numbers
- A status byte to allow for deletions
- A record code to identify the record
- Fields for exception reports (for example, credit limit or minimum stock)
- Fields to aid in analysis (for example, number of accesses to this record, date of last update, number of returns, or pending order)
- Fields reserved for expansion

Naming Fields

Meaningful field names can make your program logic clearer to a user who is not familiar with your program.

Usually, high-level languages (such as RPG II or COBOL) limit the number of characters you can use for field names. Therefore, you usually have to use abbreviations for field names. By establishing a list of standard abbreviations, you can make sure that they are readily understood by everyone in your installation. For example, various programmers might abbreviate the word MASTER as MST, MAST, or MSTR. By consistently using one of these abbreviations, you can help clarify what the abbreviation means.

If you use descriptive field names, you usually have several files that contain fields with the same name. When these files are used in the same program, you must qualify or redefine fields that have the same name, depending on the field-naming requirements of the high-level language you are using.

You can avoid this problem by using an abbreviation of the file name as a prefix to the field name. For example, you might use MST as the prefix to all field names in your master file. By using unique prefixes such as MST for MASTER, TRAN for TRANSACTION, PAY for PAYROLL, and INV for INVENTORY, you can help eliminate misinterpretation and confusion.

The field description entries associated with any one of your files can be standardized. You can write a field description for each of your files and then store these definitions in separate source members in your libraries. You can then copy these field definitions into your program by using $AUTO / COPY, SEU, or DSU Full Screen Editor.

Note: All functions done with SEU can be done with DSU.

This approach has several advantages:

- Data definitions are standardized. The programmers in your installation use identical file definitions with the same field names and descriptions.

- You save time. The field description for each file needs to be written, entered, and debugged only once. This description can then be made available to all programmers.

- Maintenance becomes considerably easier. Programmers can easily pick up the structure of the data because they have probably worked with the same field definitions in another program. If the field definition itself needs to be updated, only one source member needs to be modified. This member is then copied into each program that used the old source member.

- The source member for each file can be used to document the contents of the file. When you build the field description for each file, you can include comments that document what each field is used for, what the valid range of values is, and whatever additional information you think is necessary.

The high-level language you use to write your program determines the maximum length you can specify for your field names. See the appropriate language manual for information about the maximum length allowed for field names in the programming language you use.

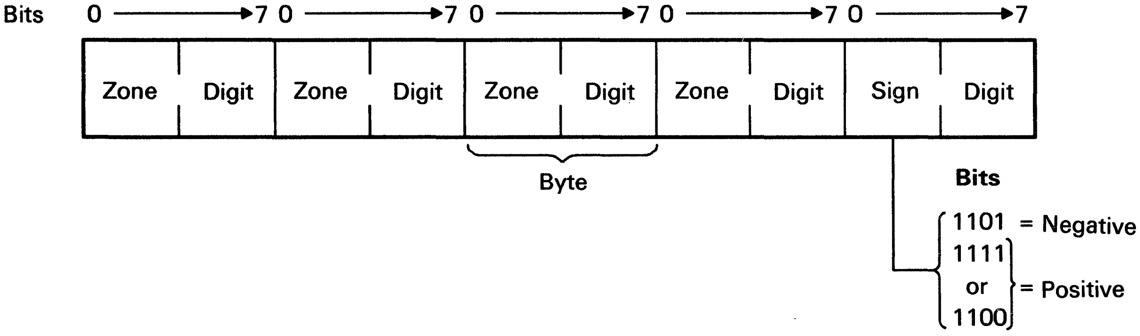
Numeric Fields

If you have numeric fields, you should determine whether the field is to be in zoned decimal format, packed decimal format, or binary format. Packed and binary formats can reduce the amount of space needed to store numeric information.

Be sure to allow the maximum length for amount fields. Otherwise, a high-order position could be lost in exceptional cases.

Zoned Decimal Format

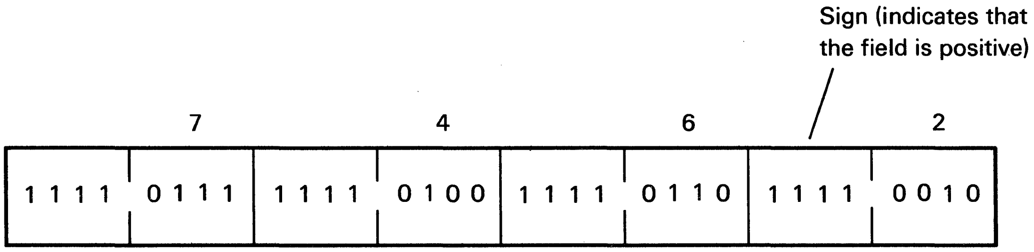
For data in zoned decimal format, each byte of storage represents a single character. Each byte of storage is divided into two parts: a 4-bit zone portion and a 4-bit digit portion. Zoned decimal format looks like this:



S9019094-1

The zone portion of the rightmost byte contains the representation of the sign. A positive sign is represented by hexadecimal F (1111) or C (1100). A negative sign is represented by hexadecimal D (1101).

The number 7462 stored in zoned decimal format looks like this:



S9019095-0

In your program, you can specify whether a field is signed or unsigned. A signed field stores the sign with the data. An unsigned field does not store the sign with the data. To process fields with a negative value, you must define the field as signed. If you do not specify that a field is signed, the system assumes that the data in the field is positive. This can cause unexpected results. For example, if you code your program so that an operation is performed if the sign of a field is negative, the operation cannot be performed if the field is unsigned.

When the zone portion of the rightmost byte is changed to represent the sign, the rightmost byte might become a character that cannot be printed or displayed. The following table shows how changing the sign affects how the rightmost byte is printed or displayed.

Sign Portion of the Rightmost Byte	**Value of the Rightmost Byte**	**How the Rightmost Byte Is Printed or Displayed**
Hex F (positive)	0 through 9	0 through 9
Hex C (positive)	0	Unprintable[1]
Hex C (positive)	1 through 9	A through I
Hex D (negative)	0	Unprintable[1]
Hex D (negative)	1 through 9	J through R

[1] These characters can be printed if your printer has a printer belt that includes additional special characters.

For example, if you place a value of +1234 in a signed zoned decimal field, it is stored in the computer as:

```
F1 F2 F3 F4
```

If you print this field, it appears as:

```
1 2 3 4
```

If you place a value of -1234 in a signed zoned decimal field, it is stored in the computer as:

```
F1 F2 F3 D4
```

However, if you print this field, it appears as:

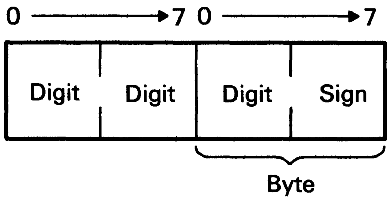
```
1 2 3 M
```

because hex D4 is the EBCDIC value of the letter M. Therefore, your program must convert the data in the field to properly print the value.

Packed Decimal Format

For data in packed decimal format, each byte of storage can represent two decimal digits. Therefore, you can store almost twice as much data in the same amount of storage by using packed decimal format rather than zoned decimal format.

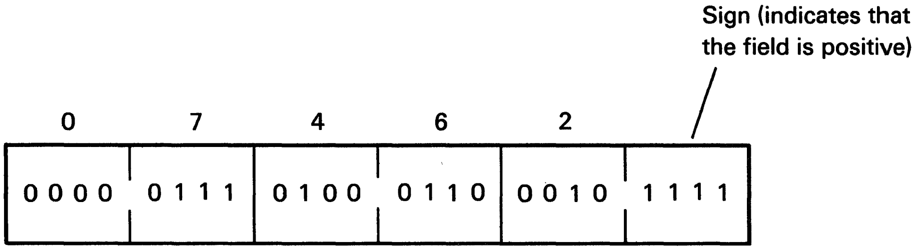
In packed decimal format, each byte of storage except the rightmost is divided into two 4-bit digit portions. Packed decimal format looks like this:



S9019096-0

The rightmost 4 bits of the rightmost byte are reserved for the sign. If you are storing a field with an odd number of digits, the field completely fills the bytes of storage reserved for it, as shown in the preceding figure. However, if you are storing a field with an even number of digits, the field is right-adjusted within the bytes of storage reserved for it. The leftmost digit portion of the leftmost byte contains only zeros.

The number 7462 stored in packed decimal format looks like this:



S9019097-0

In packed decimal format, a positive sign is represented by hexadecimal F. A negative sign is represented by hexadecimal D.

If you define a numeric field with a value of +1234 as an unsigned packed decimal field, it is stored in the computer as:

```
01 23 4F
```

If you define a numeric field with a value of -1234 as an unsigned packed decimal field, it is also stored in the computer as:

```
01 23 4F
```

However, if you define a numeric field with a value of -1234 as a **signed** packed decimal field, it is stored in the computer as:

```
01 23 4D
```

To print or display a packed decimal field, you must first translate it into its zoned decimal equivalents.

The maximum length of a packed decimal field is 15 digits (8 bytes of storage). The following table shows the number of bytes of storage required to store zoned decimal and packed decimal fields.

Digits in Digits	**Zoned Decimal Format**	**Packed Decimal Format**
1	1	1
2	2	2
3	3	2
4	4	3
5	5	3
6	6	4
7	7	4
8	8	5
9	9	5
10	10	6
11	11	6
12	12	7
13	13	7
14	14	8
15	15	8

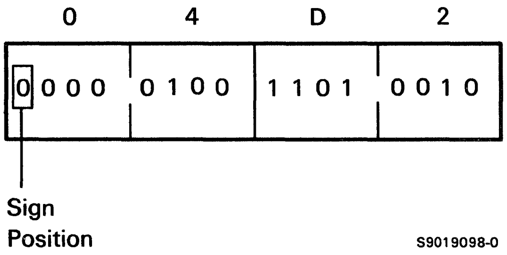
Binary Format

Data in binary format is represented in storage in binary digits; that is, as a number to the base 2. A binary field usually occupies less storage than a zoned decimal field and sometimes occupies less storage than a packed decimal field. The following table shows how many bytes of storage are occupied by binary fields of various lengths.

Number of Digits in Field	Bytes Required to Represent the Number	Bytes of Storage Occupied
1 to 2	1	2
3 to 4	2	2
5 to 6	3	4
7 to 9	4	4
10 to 11	5	8
12 to 14	6	8
15 to 18	7	8

A binary field normally reserves the leftmost bit of its representation in storage as the sign position. If the sign position contains 0, the number is positive. If the sign position contains 1, the number is negative.

For example, if a binary field has a value of -1234 and you define the field as unsigned, it is represented in storage as hex 04D2:

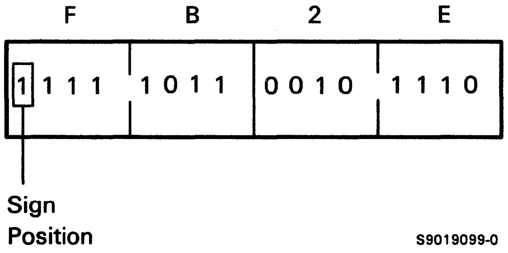


If you define a binary field as signed, it is stored as a negative number. Negative binary fields are stored in **twos complement**. To find the twos complement of a binary field, follow these steps:

1. Change the setting of each bit in the binary representation. That is, change each 0 to 1, and each 1 to 0.

2. Add 1 to the new binary representation.

For example, the value -1234 is stored as hex FB2E:



To print or display a binary field, you must first translate it into its zoned decimal equivalent.

Floating-Point Format

Data expressed in floating-point format consists of a decimal number followed by an exponent. The decimal number is called the **mantissa**. The **exponent** specifies a power of 10 that is used as a multiplier to indicate the placement of the decimal point.

The value of a floating-point number is the mantissa multiplied by the power of 10 expressed by the exponent. For example, 3E02 is the floating-point representation of 3 times 10 to the 2 power, or 300; while 3E-2 equals 0.03.

Data in floating-point format has the following form:

[±]mantissaE[±]exponent

Each character or digit occupies one byte of storage.

+ or - Signs: A plus (+) or minus (-) sign is optional before the mantissa and before the exponent. If no sign is specified, a positive mantissa or exponent is assumed. A plus (+) sign indicates positive values and a minus (-) sign represents negative values.

Mantissa: The mantissa can contain from 1 to 16 digits. The mantissa must contain one actual or assumed decimal point as a leading, embedded, or trailing symbol.

The Letter E: The letter E immediately follows the mantissa and indicates the exponent.

Exponent: The exponent immediately follows the second optional sign character. It can contain from 1 to 3 digits.

Alphameric Fields

Alphameric data can contain letters, numbers and special characters. Special characters are those characters other than alphabetic or numeric characters. For example; *, +, and % are special characters.

See the appropriate language manual for information about the lengths allowed for alphameric fields.

Key Length and Placement

A key is one or more characters used to identify the record and establish the record's order within an indexed file. The maximum key length is 120 bytes, although in some cases the limit is lower (for example, WSU, DFU, RPG II, and Query/36 all restrict the key length to 99 bytes).

A multiple-index file uses a separate key for each index. The key for the primary index of an indexed file must be made up of fields that are next to each other in the record, or contiguous. The keys for alternative indexes (for all file types) do not have be built from contiguous fields in the record. Up to three noncontiguous fields can be used as the key for an alternative index. For example, fields 1, 2, and 3 could be the key for the primary index of an indexed file; fields 1, 3, and 10 of a record could be the key for an alternative indexed file.

Files and indexes are described in Chapter 8, "Files."

Providing for Deletion of Records

When a record is deleted from a delete-capable file, the record remains in the file, but the data is unavailable for use. If you might need the data that was in a deleted record, do not use a delete-capable file. Instead, have your program place a delete code in the record. Then, when the file is processed, your program can check for this code. For example, if a customer record becomes inactive, you may want to place a delete code in the record and have the program check for this delete code. When the program finds this delete code, it can bypass the record instead of processing it. Later, if you wish to reactivate the inactive customer record, you can do so by removing the delete code.

The system does not treat records with a user-specified delete code as deleted records.

To remove deleted records from a file, you can use the COPYDATA or SAVE procedure. The *System Reference* manual has information about the COPYDATA and SAVE procedures.

Field Sizes

Field size depends on the data in the field. The data in many fields contains the same number of characters for every record. For example, every record has a six-position field for the customer number, and all six positions contain data. Other fields contain a varying number of characters. In those cases, each field should be long enough to contain the largest number of characters for that field in any record. For example, the length of each customer's name varies, but 20 positions should be long enough to contain any customer's name.

Defining Record Length

The lengths of various fields in a record may vary, but a given field should have the same length in every record, and all records in a particular file must have the same length. Record length is the sum of the field lengths, plus any extra space reserved for new fields. The maximum record length for a disk file is 4096 positions.

Allowing for New Fields

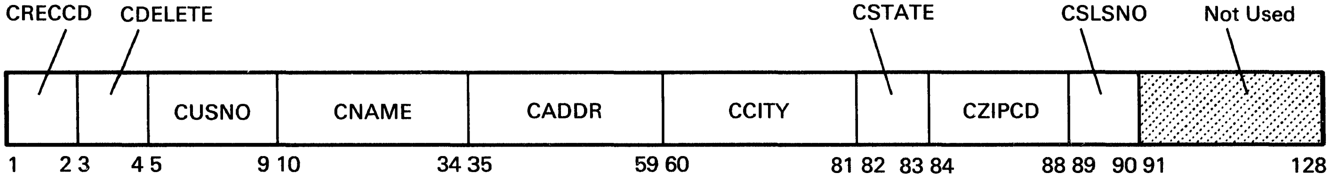
You should allow for future additions to the record. For example, a new field might be needed in the record. Your record design should allow for that possibility.

One way to allow for new fields is to make the record length longer than initially required (10 percent longer, for example). However, this method requires extra space and can decrease performance. For example, if a file contains 1000 records, and if 20 positions are set aside in each record for future additions, the file requires an additional 20,000 bytes.

Another way to allow for new fields is to have the program that reads the file add the new fields and then write the larger file back to disk.

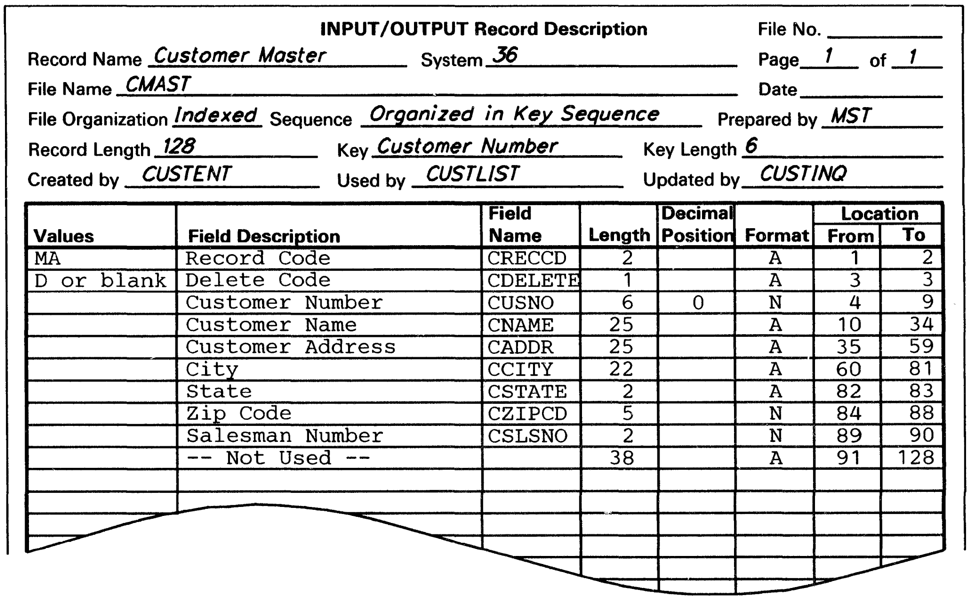
Documenting Record Layout

When you document your record layouts, your programs are easier to write. The following diagram shows the layout of a master customer record. Record layout includes the order of the fields in the record, the length of each field, and the name of each field. In this example, the field names all begin with the letter C to indicate that the record is from the customer master file.



S9019068-0

The following is one way to document this record layout:



INPUT/OUTPUT Record Description

File No. ______

Record Name *Customer Master* System *36* Page *1* of *1*

File Name *CMAST* Date ______

File Organization *Indexed* Sequence *Organized in Key Sequence* Prepared by *MST*

Record Length *128* Key *Customer Number* Key Length *6*

Created by *CUSTENT* Used by *CUSTLIST* Updated by *CUSTINQ*

Values	Field Description	Field Name	Length	Decimal Position	Format	Location	
						From	To
MA	Record Code	CRECCD	2		A	1	2
D or blank	Delete Code	CDELETE	1		A	3	3
	Customer Number	CUSNO	6	0	N	4	9
	Customer Name	CNAME	25		A	10	34
	Customer Address	CADDR	25		A	35	59
	City	CCITY	22		A	60	81
	State	CSTATE	2		A	82	83
	Zip Code	CZIPCD	5		N	84	88
	Salesman Number	CSLSNO	2		N	89	90
	-- Not Used --		38		A	91	128

S9019112-0

On the following page is a sample record description form. You can make copies of this form to use in documenting your record layouts.

INPUT/OUTPUT Record Description

File No. ____________

Record Name ______________________________ System ______________________________ Page______ of ______

File Name ____ Date ____________

File Organization __________ Sequence ______________________________________ Prepared by ____________

Record Length __________________ Key ____________________________ Key Length ____________

Created by __________________ Used by ____________________________ Updated by ____________

Values	Field Description	Field Name	Length	Decimal Position	Format	Location	
						From	To

S9019113-0

Chapter 8. Files

A **file** is a set of related records that is treated as a unit. Files contain information such as customer accounting data.

Introduction to Files

This chapter describes disk files, including:

- Using files in general
- File organizations and access methods
- File attributes
- Blocking records and index entries
- Sharing files

This introductory section gives an overview of some basic file concepts, each of which is described in greater detail later in this chapter.

Using Files in General

As you program your applications, you do the following:

- Create files
- Name files
- Specify files in programs
- Place data in files
- Remove files from disk
- Secure files

File Organizations and Access Methods

The system allows three types of file organizations: sequential, direct, and indexed. File organization refers to the way records are placed in a file.

Sequential Files: Records occur in the order in which they were entered into the file. IDDU creates only sequential files.

Direct Files: Records occur in any order. Each record has an associated relative record number, which specifies the location of a record in relation to the beginning of the file.

Indexed Files: The key and the position of each record are recorded in a separate portion of the file called an index. The key is one or more characters in the record and is used to access the record in the file.

File access methods refer to the way in which your programs use or process the file. The system allows programs to access files using the following methods:

Consecutive access allows you to access records in the order in which they appear in the file, one after the other from the first to the last.

Random access by relative record number allows you to access only the record you need by specifying the record's relative record number.

Sequential access by key allows you to access indexed files according to the sequence of the keys in the index.

Random access by key allows you to access, in an indexed file, only the record you need by specifying the key of the record.

Generalized access allows you to access a file by using a combination of the above methods.

File Attributes

File attributes indicate how long a file is to be retained, whether a file can be extended, or whether records in a file can be deleted.

The **file retention** attributes are:

Resident files are considered permanent files; for example, a customer master file is usually a resident file. These files are always on the system until they are explicitly removed. Resident files can be shared among several jobs.

Job files exist from the time they are created and remain on the system until the job that was using them ends. Job files are usually used to store temporary data that is needed only from one job step to the next.

Scratch files exist from the time they are created and remain on the system until the program that was using them ends. Scratch files are used to store temporary data for one program.

The **extendable file** attribute allows the system to automatically extend a file when you add more records than the file's current size allows. This prevents a program from ending abnormally when the file does not contain enough room to add more records.

The **delete-capable file** attribute allows your programs to use their delete statements or codes to delete records from the file. This prevents you from having to reserve a record location for a delete code, for which all your programs would have to check.

Blocking of Records and Indexes

The system allows you to block records and index entries. The record blocking factor specifies approximately how many records are to be transferred for each disk input/output operation. Record blocking is useful when you are likely to process several records at a time. By specifying a large blocking factor, you reduce the number of disk input/output operations that are required.

Index entry blocking tells the system how many index entries to read from the disk for each disk input operation. Blocking index entries can save processing time when your program is accessing consecutive index entries, because the system does not have to read the index on disk until all the index entries in main storage are processed.

If you are using the cache, it is better not to specify a blocking factor.

File Sharing

The system allows resident files to be shared by two or more programs at the same time. The system allows you to specify the sharing levels of the programs that are to share files. For example, if two programs are sharing a file, one program can be allowed to read and modify data, but the other program can be allowed only to read the file.

Using Files

This section describes several topics about files in general, including:

- Creating files
- Naming files
- Specifying a file for a program
- Placing data in files
- Removing files from disk
- Securing files

Creating a File

You can create files on disk by using:

- The system help menu for working with files. To display that help menu, enter HELP CREAFILE.

- The BLDFILE procedure. For example, if you want to create a sequential file named NEWFILE on disk unit A1 and you have enough space for one hundred 256-byte records, you would enter the following statement:

  ```
  BLDFILE NEWFILE,S,R,100,256,A1,T
  ```

- WSU or DFU, which automatically creates files. WSU creates direct files. DFU creates sequential, direct, or indexed files.

- A program that does output operations. The FILE OCL statement may have DISP-NEW or no DISP parameter specified. For example, if you want your program to create an output file named OUT1 with the label OUTPUT and you have enough disk space for 10 blocks of data, you could enter the following FILE OCL statement:

  ```
  // FILE NAME-OUT1,UNIT-F1,LABEL-OUTPUT,BLOCKS-10,
  //      RETAIN-T,DISP-NEW
  ```

- The OPEN statement in BASIC.

- The RESTORE, POST, TRANSFER, or TAPECOPY procedure to copy files from diskette or tape to disk.

- The COPYDATA procedure to copy files from disk to disk.

- The Query/36 data entry facility.

- IDDU.

You have two options when you specify the size of files being created. You can specify either:

- The number of records that the file is to contain

- The number of blocks of disk space desired for the file

For example, you may want to create a file with enough room for five hundred 256-byte records, or you may wish to create a file that has 20 blocks of disk space allocated for it.

Naming a File

The SSP requires a unique identification for each file. This identification is provided by a file name and a file label. A file name or file label can be up to 8 characters long and must begin with an alphabetic character (A through Z, #, $, or @). characters can be any combination of characters (numeric, alphabetic, and special). Do not name files ALL.

You should avoid using the following characters because these have special meanings in procedures: commas (,), apostrophes ('), question marks (?), slashes (/), greater than signs (>), equal signs (=), plus signs (+), and hyphens (-).

You can create meaningful file labels by abbreviating the name of the application that uses the file. For example:

File Label	Application or Type of File
ACCTRECV	Accounts receivable
CUSTMAST	Master customer file
CUSTORDS	Customer orders
INVTRANS	Inventory transactions
ITEMBAL	Item balance file
INVMAST	Inventory master file
SHIPMAST	Ship-to master file

Do not confuse the file name with the file label. The file name indicates how the program refers to the file. The file label indicates how the system refers to the file on disk. Both NAME and LABEL are parameters on the FILE OCL statement. The LABEL parameter must be specified only when the name of the file on disk is different from the name used in the program. If the LABEL parameter is not specified, the name specified as the NAME parameter is used. For example, in the COBOL coding and the FILE OCL statement shown below, the file name used by the program is ITEMMAST, but the file label is INVMAST (the actual name of the file on disk).

COBOL Program Segment:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
       SELECT INPUT-FILE ASSIGN TO DISK-ITEMMAST.
```

FILE OCL Statement:

```
// FILE NAME-ITEMMAST,LABEL-INVMAST
```

The *System Reference* manual has more information about the NAME and LABEL parameters on the FILE OCL statement.

File Dates

You can assign the same file label to more than one file as long as each file has a different creation date.

During system configuration, you can specify that the system prevent a new disk file (with a different date) from being created if it has the same label as an existing file. See the *Changing Your System Configuration* manual for more information.

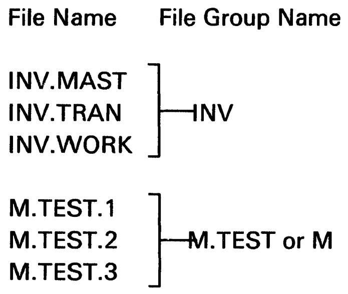
Group Files

Group files are a set of files collectively identified by a file name that contains identifiers separated by one or more periods. The characters preceding a period identify the file group.

The advantage of using group files is that they can be saved, restored, or deleted in one step by using the SAVE, RESTORE, or DELETE procedure. You should consider using group files for all files used by a particular application or portion of an application.

Another advantage of group files is that they can be secured easily. See the *System Security Guide* for more information about securing group files.

Here are some examples of file names that identify group files:



S9019124-0

The limit of 8 characters for a file name also applies to names for group files. The period counts as one of the 8 characters.

To have a meaningful group name, it should not be more than 6 characters long plus a period (.). A 7-character group name plus a period (.) would not allow more characters to differentiate between files in a group. An 8-character group name is not possible because no period (.) is allowed after the name.

Files in a Group Resource

Files, folders, and libraries can be part of a group resource. The group resource must be named and each file, library, or folder that is part of the group must be named.

For example, a group resource could be PAY: a file that is part of this group could be PAY.FILE, a library could be PAY.LIB, and a folder could be PAY.FOLD. As you can see, the group resource name and the name of the file, library, or folder are separated by a period (.). PAY is called the group identifier.

One advantage of using groups is that they can be secured easily. See the *System Security Guide* for more information about securing groups.

The limit of 8 characters for a file name also applies to names of groups. A period counts as one of the 8 characters.

Renaming a File

You can use the RENAME procedure to rename files. However, you should be careful not to rename files that other programs, procedures, or utilities depend on. For example, if you rename a file that is used by several jobs, the reference to that file name must also be changed in each job.

The *System Reference* manual has more information about the RENAME procedure.

Specifying a File in a Program

To use a file in a program, you must specify the file name and, optionally, the file label in the program. When the program is compiled, the compiler extracts the necessary file information from the file description in the program. The system uses some of this information to define the record formats to be processed and some of it to open the file.

The file description in the program contains a description of:

- The record formats, which includes the field names, data types (numeric, alphameric, or character), and field lengths
- The access method
- The file organization as specified in the high-level language program

All records in a file are fixed length and can have the same or different field descriptions. In your high-level language program, you can subdivide a field and/or redefine the field for processing. Records are described in Chapter 7, "Designing Records."

Your high-level language program must open a file before issuing requests to read, write, update, or delete records in the file. A request to open a file allows the file to be processed by the program. A request to close a file prevents further processing of the file by the program. How you issue a request to open or close a file depends on which high-level language you use. In some high-level languages, you must code the request in your program; in other languages, the open or close operation is performed automatically.

Some high-level languages allow you to specify file parameters in the program; others require a FILE OCL statement to specify these parameters. Before a file is opened, the system checks to see if a FILE OCL statement has been entered to override the file specified in the program. The file description in the program is merged with the parameters on the file-overriding FILE OCL statement.

See the appropriate language manual for more information.

Placing Data in Files

There are several ways to place data in files:

- High-level language program. For more information, see the appropriate language manual.

- WSU. The *Work Station Utility Guide* has more information.

- DFU. The *Data File Utility Guide* has more information.

- Query/36. *Getting Started with Query/36* or the *Query/36 Online Information* has more information.

- Diskette to disk. Use the TRANSFER, RESTORE, or POST procedure.

- Tape to disk. Use the TAPECOPY or RESTORE procedure. The *System Reference* manual has more information about these procedures.

- COPYDATA procedure. The COPYDATA procedure copies a file on disk to another file on disk. During the copy operation, the COPYDATA procedure can:

  - Remove deleted records

  - Include or omit specific records

  - Create the copied data in either the same or a different file organization

  - Specify a selection criteria to copy specific records

  The *System Reference* manual has more information about the COPYDATA procedure.

Printing or Displaying Files

You can use the following methods to print or display files:

- High-level language program. For more information, see the appropriate language manual.

- DFU. The *Data File Utility Guide* has more information.

- Query/36. *Getting Started with Query/36* or the *Query/36 Online Information* has more information.

- LISTDATA procedure. The LISTDATA procedure allows you to selectively list data from any file, no matter what the file organization is. This data can be printed or displayed.

  The options of the LISTDATA procedure allow you to:

  - Print or display the records in either character or hexadecimal representation.

  - Specify the maximum number of records to be printed.

  - Make the record length either larger or smaller than the original record length.

  - Include or omit selected records.

  - Specify a selection criterion for printing or displaying specific records, for example, greater than or equal to a specific value in the record.

  The *System Reference* manual has more information about the LISTDATA procedure.

- LISTFILE procedure. The LISTFILE procedure allows you to list files or libraries from disk, diskette, or tape. The *System Reference* manual has more information about the LISTFILE procedure.

Reorganizing Disk Space

When disk space is reorganized, the free space on disk is collected and files are moved to create continuous free space for new files or libraries. You cannot create new files unless there is enough continuous space on the disk for each new file. For information about reorganizing disk space, see "Reorganizing Disk Space" on page 4-21.

Securing Files

You can secure your files by using resource security. Not everyone who uses the system should be allowed to read, write, or change data in a file. For example, payroll information should not be accessible to everyone. You should decide who should be allowed to:

- Read data from the file
- Create or delete the file
- Change data in a file

The access levels you can specify for resource security allow you to decide who is the owner of the file, who can change the file, and who can only read the file. These access levels are listed below in descending order, with the highest access levels having the greatest authority. The authority that is given by a higher access level includes all the authority of the access levels below it (except, of course, for an access level of None). For example, the user with change access to a file also has update and read access to it.

Access Level **Description**

Owner The owner of a file can:

- Indicate the users of the file and their access levels.
- Create, rename, or delete the file.
- Read, write, update, or delete records in the file.

Change A user with change access to a file can:

- Create, rename, or delete the file.
- Read, write, update, or delete records in the file.

Update A user with update access to a file can read, write, update, or delete records in the file.

Read A user with read access to a file can only read records in the file.

None A user with this access is not allowed to access the file.

The *System Security Guide* has more information about how to secure files.

Removing a File from Disk or Diskette

To remove a file from disk or diskette, use the DELETE procedure. The *System Reference* manual has more information about the DELETE procedure.

File Organizations

The system allows three types of file organizations: sequential, direct, and indexed. This section describes each organization.

Sequential File Organization

In a sequential file, records are arranged in exactly the same sequence as they were placed into the file by a program that created the file. The first record is placed in the first position in the file, the second record is placed in the second position, and the rest of the records are placed sequentially in the file one after another.

Figure 8-1 shows the creation of two sequential files.

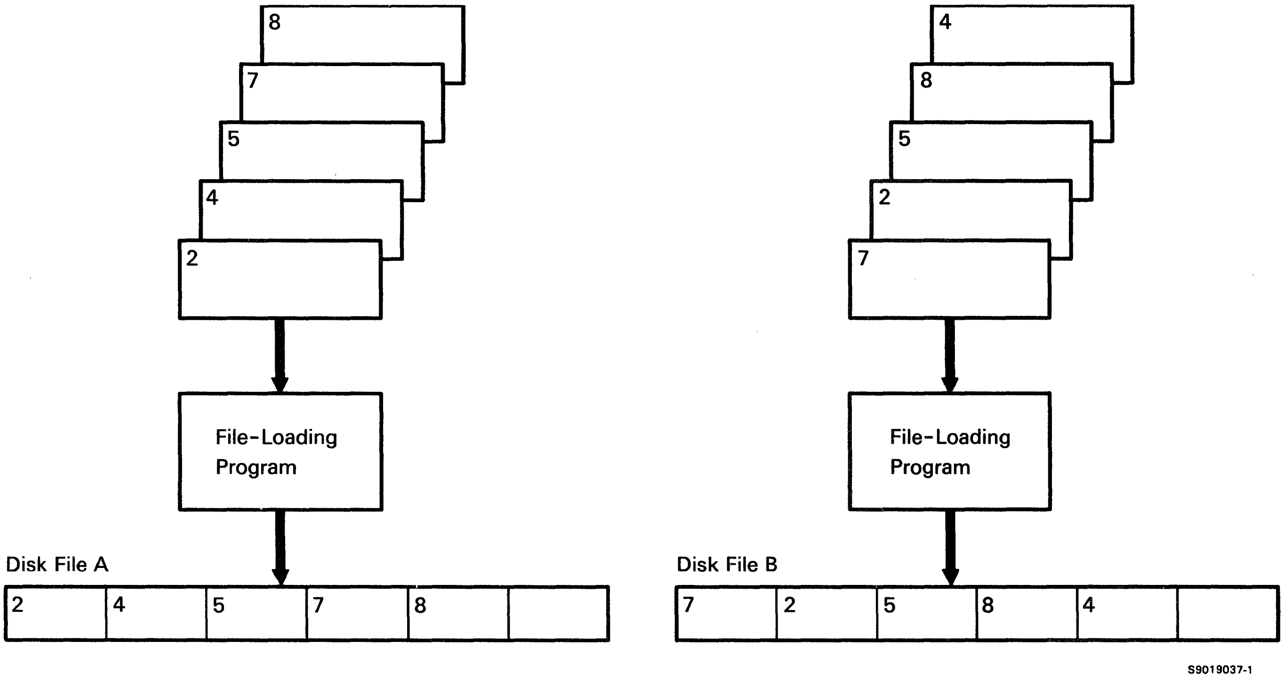


Figure 8-1. Organization of a Sequential File

The records on the left are read by the file loading program in the following sequence: 2, 4, 5, 7, 8. They are loaded onto a disk in exactly the same sequence; thus the disk file is a sequential file. The records on the right are read by the file loading program in the sequence: 7, 2, 5, 8, 4. They are loaded onto a disk in that same sequence; again fitting the definition of a sequential file.

Thus, a sequential file does not mean that the records are stored in the file in numerical sequence. Rather, it means that the records are stored in the file **in the sequence that they were written by the file loading program.**

The records in a sequential file can be loaded in an ordered sequence or in an unordered sequence. In an ordered sequence, records are arranged into ascending or descending sequence, based on the value of some control field. The records in Disk File A in Figure 8-1 are loaded in an ordered sequence. The records in the Disk File B in Figure 8-1 are loaded in an unordered sequence. Whether a file is loaded in an ordered or unordered sequence is determined by the way the file loading program is written, **not** by the sequence of the input source records. However, most file loading programs load records in the same sequence as the sequence of input records.

A sequential file takes less space than a direct file or an indexed file. Direct files typically have gaps in the file for missing records; indexed files use extra space for the index.

When a record is added to a sequential file, the record is placed at the end of the file, not between existing records. Thus, if record 6 is added to Disk File A in Figure 8-1, it is placed after record 8, not between records 5 and 7. If it is important that record 6 appear between records 5 and 7, you can run a sort program to arrange the records into a new ordered sequence.

Sequential files can be accessed consecutively, randomly by relative record number, or by the generalized access method (both consecutively and randomly). Each of these access methods is discussed under "File Access Methods" on page 8-35.

Direct File Organization

In a direct file, records are loaded into **assigned places within the file.** Normally, the file loading program uses the value of one of the fields from the input record to point to the position in the disk file where the record should be placed. This pointer is called the **relative record number.** It identifies the position of the record relative to the beginning of the file. For example, if the decimal value of a relative record is 8, that record is placed in the eighth record position in the file.

No matter what sequence the records are read by the file loading program, the file loading program places the records into the direct file at the proper location. For example, Figure 8-2 shows two sequences of input records.

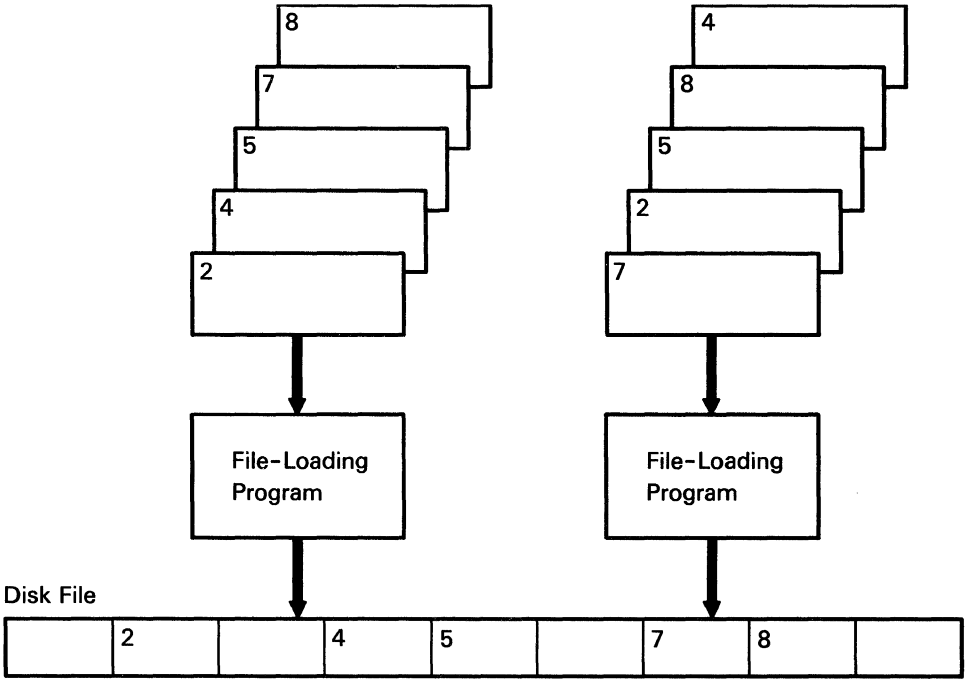


Figure 8-2. Organization of a Direct File

The input records on the left in Figure 8-2 are in the sequence: 2, 4, 5, 7, 8. The input records on the right are in the sequence: 7, 2, 5, 8, 4. However, in both cases, the file loading program places the records into the same positions in the direct file. Spaces are left for records 1, 3, and 6. When records are added to a direct file, they are placed in the spaces that were left for them. For example, record 6 would be added between records 5 and 7.

When a direct file is created, all record positions in the file are initialized to blanks (hex 40s) if the file is not delete-capable. If the file is delete-capable, unused record positions are marked as deleted records and initialized to hex FF.

Relative record numbers can be either decimal (for Assembler, BASIC, COBOL, FORTRAN IV, or RPG II programs) or binary (for Assembler programs). If the relative record number is a decimal number, the first record position in the file has a value of 1. If the relative record number is a binary number, the first record position in the file has a value of 0.

Usually, the relative record number is based on a value stored in the record. However, the relative record number can also be a computed value. In this case, an arithmetic formula is used to calculate the position in the file where the record is to be stored. Normally, these calculations are based on the values of certain fields in the record.

Direct organization can use a great deal of disk storage space if the formula you choose for calculating record positions leaves many unused record positions in the file. For example, if your calculations create relative record numbers from 1 to 1000, space is reserved in the file for 1000 records. If you use only 100 records, the space for the other 900 records is not used.

Occasionally, when you use formulas to determine relative positions, more than one record has the same calculated position in the file. These records are called **synonym records** (see Figure 8-3).

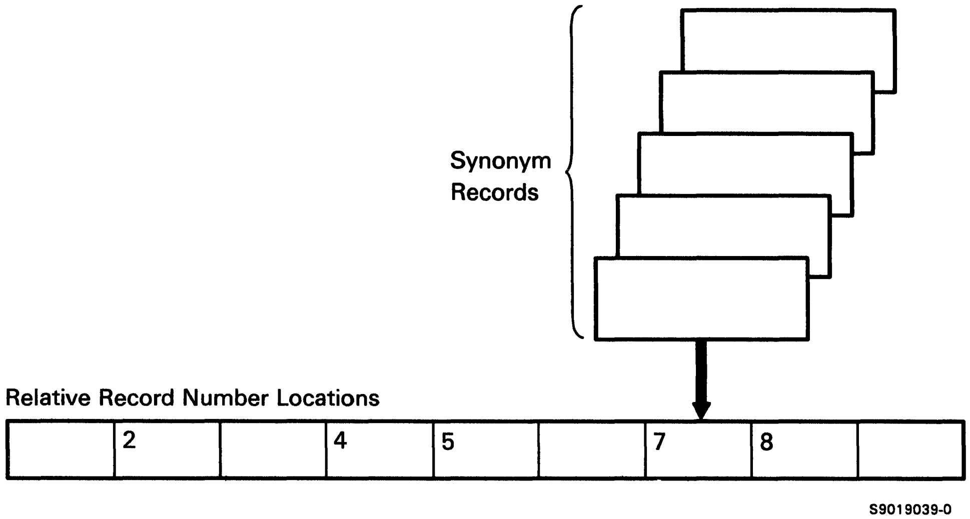


Figure 8-3. Synonym Records

If you use formulas to create direct files, you must plan how you will store and retrieve synonym records from various locations within the file, because only one synonym record can be stored in a calculated record position. For information about handling synonym records, see Appendix A, "Access Algorithms for Direct Files."

Direct files can be processed consecutively, randomly by relative record number, or by the generalized access method. Each of these access methods is discussed under "File Access Methods" on page 8-35.

Indexed File Organization

In an indexed file, the records are arranged in the same sequence as they were written by the file loading program. Therefore, the records in an indexed file are organized in the same sequence as the records in a sequential file. However, the SSP builds an **index** at the front of an indexed file. Thus, an indexed file is a file that contains two parts: an index and the records.

An index allows a program to process required records by referring to the key of the record. For example, if you have an indexed file consisting of order records containing customer number, amount ordered, and balance due, your program can use the customer number as the key to find a record for a particular customer without having to read any other records.

The index is a table containing an entry for each record in the file. Each index entry identifies a record by the value of its key and locates the record by its address in the file. The **key** (also called the **key field** or the **record key)** is the portion of the record containing information that identifies the record. For information about the rules for forming keys, see "Rules for Index Keys" on page 8-22.

Figure 8-4 shows how indexed files are organized.

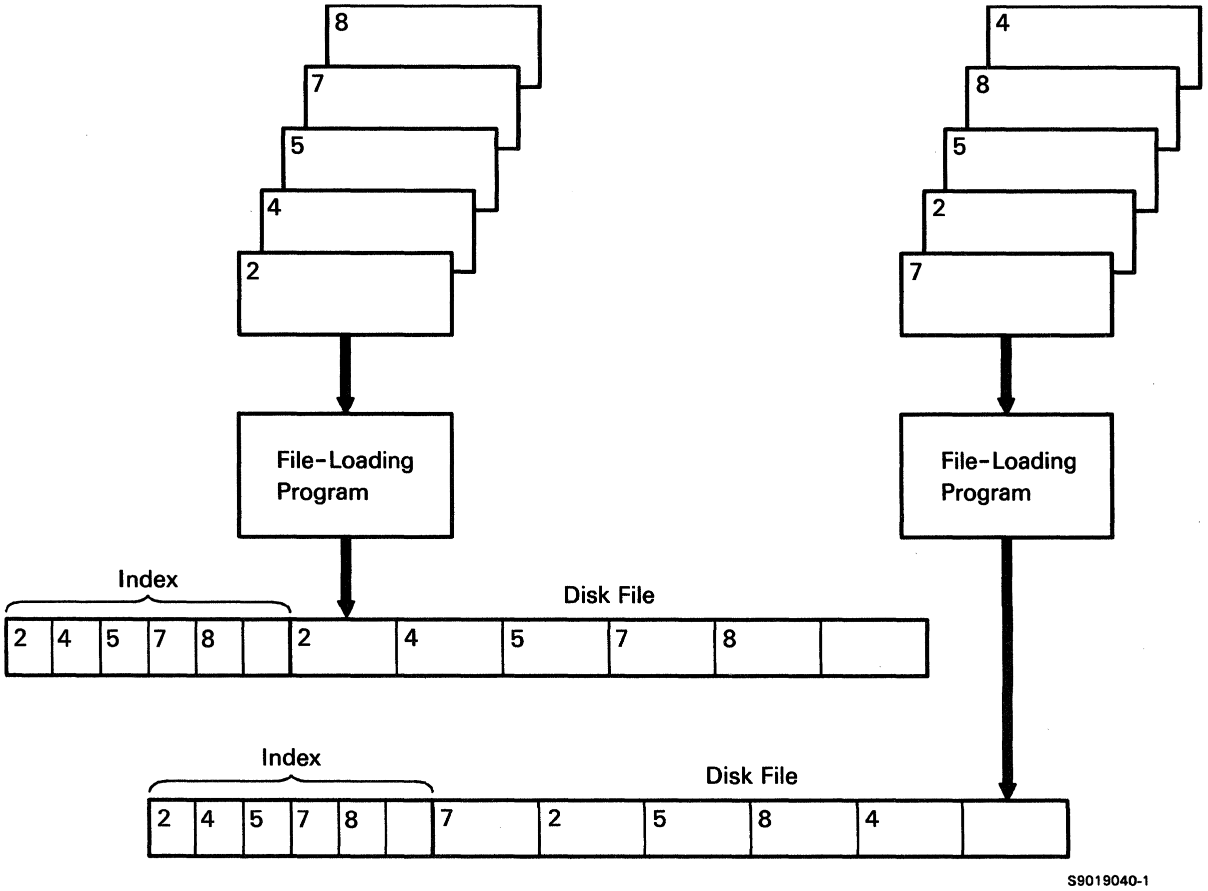
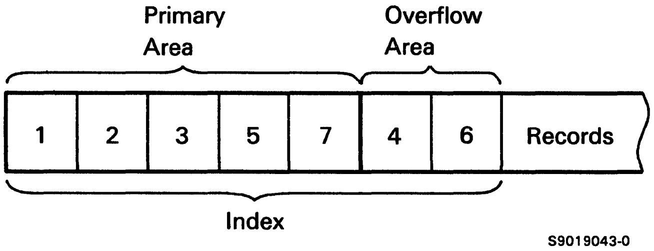


Figure 8-4. Organization of an Indexed File

Indexed files can be accessed consecutively, sequentially by key, randomly by relative record number, randomly by key, or by the generalized access method. Each of these access methods is discussed under "File Access Methods" on page 8-35.

Structure of the Index

The index is divided into a primary area and an overflow area:



The primary area contains the index entries in ascending order by the value of the key. The overflow area contains the entries for records added to the file and for records with updated keys. The system automatically keeps the entries in each area of the index in sequence. This allows you to access all the records in an indexed file sequentially by key without sorting the index.

Keysorting

When the system sorts the entire index, the entries from the overflow area are merged into the primary area. The main reason for keysorting is to increase performance when you add or update records. Indexes can be sorted when:

- A program sequentially adds records to a file that is not shared. In this case, the system may keysort the file when it is opened by the program. Index entries for records that have been added may be in the overflow area of the file after the program ends.

- The system operator enters a STOP SYSTEM or STOP SYSTEM,SORT command.

- System IPL is performed, and files are rebuilt. This topic is discussed under "IPL File-Rebuild Function" on page 19-14.

- The KEYSORT procedure is run. The system allows the keys for a shared indexed file to be sorted by the KEYSORT procedure. However, programs sharing a file whose keys are being sorted cannot access records within the file until the keysort is completed.

  The *System Reference* manual has more information about the KEYSORT procedure.

In addition, the system sorts the keys in the file when the following procedures are used to do the indicated functions:

SSP Procedure	Function
RESTORE	Restore an indexed file from diskette or tape
COPYDATA	Create an indexed file on disk
TRANSFER	Transfer a file from disk to diskette
BLDINDEX	Create a multiple-index file

Multiple Indexes for a File

A file that contains data records is called a **physical file**. After a physical file is created, you can create indexes, called **alternative indexes**, for that file without creating new data records (see Figure 8-5). You can create any number of alternative indexes for a file. A file that has one or more alternative indexes is called a **multiple-index file**. The files that you create to define each of the alternative indexes are called **alternative index** files.

Alternative indexes can be created for any file type: sequential, direct, or indexed.

When the physical file is an indexed file, the index that is built when the physical file is created is called the **primary index**.

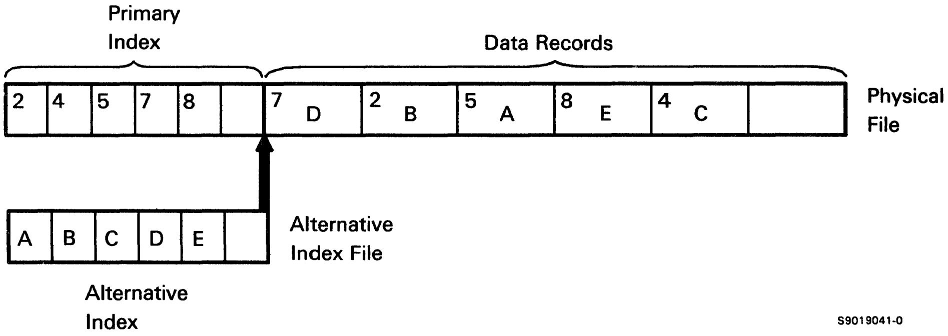


Figure 8-5. Multiple Indexes for an Indexed File

All the indexes point to the same data records in the file, but each index can use a different portion of the record as the key. The field(s) used as the key in an index can overlap the field(s) used as the key in other indexes. Therefore, you can process records from the file in various sequences, depending on which index you use. For example, for a personnel file, you can use the employee number as one key and the department number as a second key. Then you can access the records in the file by using either index. A high-level language program treats a multiple-index file the same way it treats a normal indexed file.

When changes are made to the physical file, all indexes (primary and alternative) are changed as required to reflect the change, as long as any restrictions for duplicate keys or updating keys are followed. For example, if you use the employee number as one key and the department number as a second key, any changes you make to the file while using the index based on the employee number are automatically made in the index based on the department number.

You must have a FILE OCL statement for each index that the program uses.

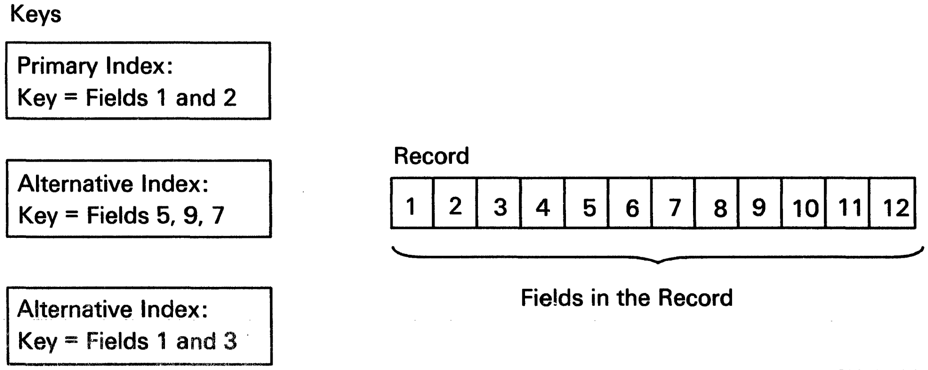
Alternative index files are created by the BLDINDEX procedure. The *System Reference* manual has more information about the BLDINDEX procedure.

Rules for Index Keys

The following rules apply to keys:

- A key can be up to 120 bytes in length, although in some cases the maximum length is lower (for example, WSU, DFU, Query/36, and RPG II all restrict the maximum length to 99 bytes).
- A key is treated as alphameric, even if the data is numeric. If a key is numeric, it should not contain a sign (+ or -).
- A key can start anywhere within the record.
- Up to three fields can be combined to form a key.
- The fields in a key must be defined so that each contains a unique set of positions in the record; for example, for a key made up of fields 2, 3, and 5, you cannot specify that field 2 contains positions 4 through 7 and that field 3 contains positions 6 through 9, because both fields contain positions 6 and 7.
- Key fields can be specified in any order, regardless of their place in the record.
- For a primary index, the fields in a key must be next to each other in the record (contiguous); for alternative indexes, they do *not* need to be next to each other in the record.

  For example, refer to the fields in the record shown below. For this record, you could use fields 1 and 2 together as the primary key; you could use fields 5, 9, and 7 together as an alternative index key; and you could use fields 1 and 3 as another alternative index key.



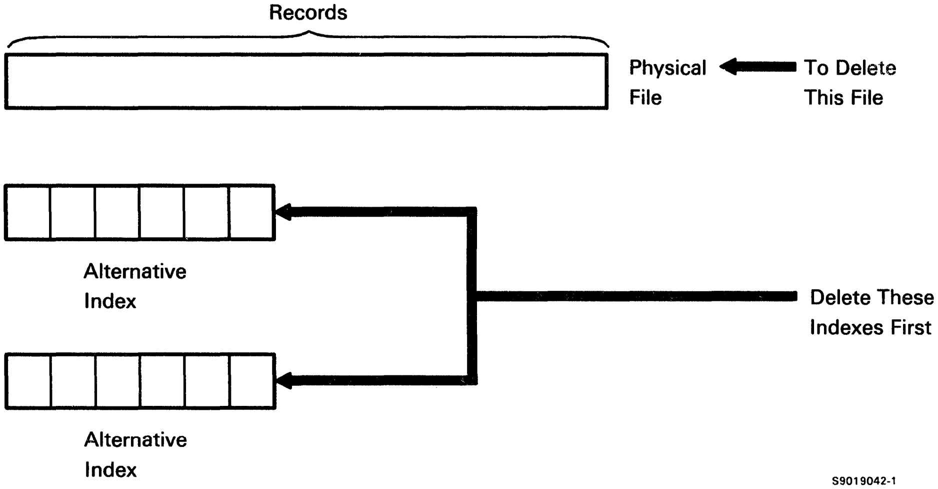
S9019118-0

Considerations for Using Multiple-Index Files: When you use multiple-index files, you should be aware of the following:

- Updates to a file can be lost if you update a file through more than one index within the same program. For more information, see "Using One File As Two Or More Logical Files" on page 8-89.

- You cannot delete the physical file if alternative indexes are specified for it. The RETAIN-S parameter of the FILE OCL statement is not allowed for the physical file or for the alternative indexes.

  To delete the physical file, you first have to delete all alternative indexes associated with the file. You can use the DELETE procedure to delete the alternative index files. You can delete a physical file and all indexes by using the file group naming convention and the DELETE procedure to delete the entire group. See "Group Files" on page 8-7 for more information about file groups.



- You cannot rewrite (overlay) an existing physical file if it has alternative indexes associated with it. Before you can overlay the data in the physical file, you must first delete the alternative indexes.

  You can never rewrite an alternative index file. You can only perform add or update operations to alternative index files.

- A multiple-index file can be specified in a procedure substitution expression that retrieves information about the file's size (?F'S'? or ?F'A'?). If the specified file is an alternative index file, the number of blocks or records allocated for the physical file is substituted.

- The date of an alternative index file is the same as the creation date of the physical file.

- When it is used by a program, a multiple-index file requires more space in the assign/free area of main storage than an indexed file requires. Extra space is required because the control blocks for all indexes (primary and alternative) must be in main storage for the SSP to use.

- On update operations, key values cannot be changed in the primary index, but key values can be changed in the alternative indexes. When you update an alternative index, consider the following:

  - In a COBOL program, you cannot change a key that has been used to retrieve the record. Therefore, the record should not be retrieved by the field to be changed.

  - In an RPG II program, between the record retrieval (CHAIN, READ, READE, or READP operation) and the record update, no other retrieval operation should be made to the file. This restriction ensures that the correct record is updated.

- When a record is updated and a key for any index is changed or when a record is added, duplicate keys can occur. The duplicate key can cause an error message to be displayed for an index that the program is not using. This error message occurs only for indexed files that do not allow duplicate keys. You indicate whether an indexed file allows duplicate keys when you create the file. For more information about specifying duplicate keys, see "Specifying Duplicate Keys" on page 8-27.

Saving Multiple-Index Files: When you use the SAVE procedure to save an indexed file that has no alternative indexes, the system saves the data and a description of the index. It does not save the index itself.

When you save an alternative index, the system saves only a description of the index. It does not save the index itself or the data in the file. Saving the physical file does not save the alternative indexes. You can save a physical file and all indexes for that file by using the file group naming convention and the SAVE procedure to save the entire group. See "Group Files" on page 8-7 for more information about file groups. When you use the SAVE procedure to save a file group, the system first saves the physical file, and then it saves the alternative indexes.

The description of the SAVE procedure in the *System Reference* manual has more information about saving multiple-index files.

Restoring Multiple-Index Files: When you use the RESTORE procedure to restore an indexed file that has no alternative indexes, the system restores the data and rebuilds the index from the description that was saved.

When you restore an alternative index, the system rebuilds the alternative index from the description that was saved. The data used for this rebuild is the data in the physical file that has the same date as the alternative indexes being restored. Therefore, when you restore the indexes individually, the physical file must be restored before any alternative indexes are restored. If the physical file is not on disk and you are restoring an alternative index for the file, an error message is displayed.

When you use RESTORE ALL to restore a multiple-index file, the system restores the physical file first and then the alternative indexes. A physical file on tape or diskette cannot be restored into an existing physical file on disk until the alternative indexes are deleted from disk.

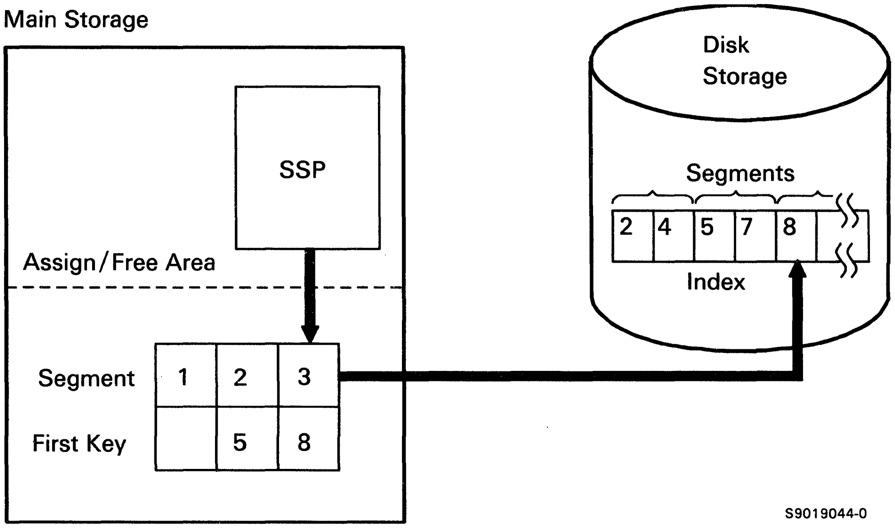
Because the system rebuilds alternative indexes when they are restored, restoring a multiple-index file can be time-consuming. The time depends on the number of records in the file, the number of alternative indexes defined for the file, the key length, and whether duplicate keys have been allowed in the file being restored.

The description of the RESTORE procedure in the *System Reference* manual has more information about restoring multiple-index files.

Storage Index

To access an indexed file randomly, the system searches the index to find the requested record. This can be time-consuming if the index is large, because the system scans the entire index for a particular entry.

In order to process indexed files faster, the system may create a storage index for indexed files that are being used by programs. The storage index resides in the assign/free area of main storage. Thus, it is an in-storage index to the file's primary index on disk.



To create a storage index, the system divides the file's primary index into segments and then uses the first key of each segment (except the first segment) as an entry in the storage index. Each storage index entry contains the lowest key field from the next segment in the file index. The size of each segment is stored at the beginning of the storage index area. By directing the program to the index segment containing the entry of the desired record, the storage index eliminates much of the needless searching of the disk.

The storage index is built when the file is first opened if the following conditions are met:

- A storage index for the file does not already exist. Only one storage index is built per file, and any programs sharing a file also share the storage index.

- Using the storage index would improve processing speed. Processing speed improves if you are accessing an indexed file randomly by key or sequentially by key within limits. It also improves if you are adding records to an indexed file that does not allow duplicate keys in the index.

- The primary portion of the disk index is large enough to justify creating a storage index for it.

The maximum size of the storage index is determined by the key length of the file, the number of records in the file, the size of the system, and the value specified on the FILE OCL statement. On systems with more than 128K bytes of main storage, the maximum default size of the storage index is 8K bytes. On systems with 128K bytes of main storage, the maximum storage index size is 2K bytes.

The building of a storage index requires additional work for the system and requires space in the assign/free portion of main storage. If your program accesses the file infrequently, the time to build the index may offset the faster processing time of the program using the storage index. For example, if you have an inquiry program that reads an indexed file only a few times, you may not want the system to create a storage index for the program. If you do not want a storage index built, specify NO on the STORINDX parameter of the FILE OCL statement when you open the file. Normally, if you access the file more than three times, you should let the system build a storage index.

If NO is specified on the STORINDX parameter of the FILE OCL statement, the program uses a storage index anyway if the storage index has already been created for the file or if a storage index is built when the file is opened by another program.

You can also specify YES on the STORINDX parameter of the FILE OCL statement if you always want a storage index to be built for all the indexes of the file. This would be useful if you are processing a file in such a way that the system does not automatically build a storage index for the file. Building a storage index improves processing time for files that have had keys changed during update operations, or for files that have records deleted.

By specifying STORINDX-YES, you also force the system to build a storage index when a particular file is opened. If you specify YES on the STORINDX parameter, the primary portion of the index must still be large enough to justify the creation of the storage index before the system will build one.

When it is determined that a storage index should be built, the SSP will try to build a storage index that will result in the fastest possible record scan (one entry per 100 sectors of index).

You can override the 8K maximum default size by specifying a value from 1 through 16 on the STORINDX parameter of the FILE OCL statement. The SSP will use this value in determining the size of the storage index. This value is the maximum storage index size.

Duplicate Keys

The system allows duplicate keys in indexed or alternative index files. For example, if you want to process employee records sequentially by department number, you can build an indexed file that uses the department number as the key. Because there would be more than one employee record for each department, the file must allow duplicate keys.

Specifying Duplicate Keys: You can specify whether a file is to allow duplicate keys when the file is created. If the BLDFILE procedure is used to create the file, allowing duplicate keys is specified as a parameter. If the file is created by a program, allowing duplicate keys is specified by the DUPKEY parameter of the FILE OCL statement. You can specify whether an alternative index file is to contain duplicate keys by using the DUPKEY parameter on the BLDINDEX procedure.

If an indexed file is created with records containing duplicate keys but the records are not supposed to contain duplicate keys, error message SYS-1365 is issued when the keys are sorted at the end of the job, and the duplicate keys are displayed. At that point, the user can choose a response that deletes the file or one that changes the file attribute so that duplicate keys are allowed for the file.

Checking for Duplicate Keys: If a file does not allow duplicate keys, the system checks for duplicate keys before an index entry is added to the file either during add operations or during update operations that change a key field. If the operation would cause a duplicate key in any index that does not allow duplicate keys, the operation is not allowed.

Sequence of Duplicate Keys: If an indexed file has duplicate keys, the sequence of the duplicate keys is maintained by the relative record number of the records in the file. That is, the first record entered or added to the file is represented by the first entry in the index. Update operations do not change the position of records in the file. If an update operation changes a key, the new key is sequenced by the relative record number of the original record.

Processing a File with Duplicate Keys: When a file with duplicate keys is accessed randomly by key, only one of the records having duplicate keys is available for processing. That record is the one whose index entry is the first in the set of duplicate entries. Random access by key is discussed under "Random Access by Key" on page 8-40.

If the generalized access method is used, the other records that have duplicate keys can be retrieved by requesting operations that read the next record in the file. The generalized access method is discussed under "Generalized Access Method" on page 8-41.

Records with duplicate keys can be accessed sequentially by key either for the entire file or for records within limits. Sequential access by key is discussed under "Sequential Access by Key" on page 8-37. When limits are specified for keys that have duplicates, the lower limit is set to the first duplicate key in the index, and the upper limit is set to the last duplicate key. The upper limit is determined as records are read from the file, and it includes any records added by other jobs.

Bypassing Duplicate-Key Checking: The BYPASS parameter on the FILE OCL statement allows the system to bypass the checking for duplicate keys for a particular file. The BYPASS parameter can be specified for indexed files used by programs other than BASIC programs. When this option is selected, disk data management does not check for duplicate keys when records are added to the file. This option should be used only when the program is designed so that records having duplicate keys cannot be added to the file. The BYPASS parameter eliminates the duplicate-key checking only for the index being used by that particular program. Disk data management still checks for duplicate keys in the indexes of other files that do not allow duplicate keys.

Using the KEYSORT Procedure: If you want to see the duplicate keys in a file, run the KEYSORT procedure with the CHKDUP parameter specified in the fourth position. This parameter causes the KEYSORT procedure to check for duplicate keys. When duplicate-key checking is specified, the KEYSORT procedure halts at each duplicate key found in the file and displays the keys. Specifying the CHKDUP parameter for the KEYSORT procedure may increase the amount of time it takes to sort the keys in the index.

Performance Considerations for Indexed Files

Many factors can affect system performance in the processing of indexed files. These factors include:

- Number of alternative indexes
- Number of indexes used at the same time
- Storage index
- Size of the overflow portion of the index
- Duplicate keys
- Key lengths
- Keysorting

Number of Alternative Indexes: If you update, delete, or add records to a multiple-index file, the system must update all the indexes. Therefore, the number of alternative indexes directly affects the performance of the program.

Because of these performance considerations, it may be more efficient to build alternative indexes for a file only when needed. For example, if a program requires a certain index in order to produce a monthly report, that index could be built just before running the job that produces the monthly report.

The decision about whether to use multiple-index files should be based on the functions of the application that is being designed. Using alternative indexes is only one of many ways to process files, and it should be considered with other options, such as whether to use indexed files or direct files.

Number of Indexes Used at the Same Time: When an index is being used, it is maintained while the program is running, and the entries in the overflow portion of the index are kept in order. When an index is not being used, entries are added to the overflow portion in an unordered sequence.

Storage Index: Building the storage index takes time when the file is opened. However, this time is made up if the program accesses the file several times. More information about storage indexes is under "Storage Index" on page 8-25.

Size of the Overflow Portion of the Index: When the overflow portion of an index becomes large, the time that it takes to update the index can become significant, because the system maintains the sequence of index entries in the overflow area whenever an entry is added. The larger the overflow area, the more time this sequencing takes. The size of the overflow area depends on both the key length and the number of records that were added or updated since the last keysort. For more information, see "Keysorting for Multiple-Index Files" on page 8-31.

Duplicate Keys: If a file is defined as not allowing duplicate keys, the system must check for a duplicate key each time an index entry is added or changed. This error-checking requires additional time that must be considered when you decide whether to allow duplicate keys.

This additional checking can be avoided by allowing duplicate keys in the file or, if you know that the keys are not duplicated, by specifying the BYPASS parameter on the FILE OCL statement. For more information, see "Duplicate Keys" on page 8-27.

The time required to update and/or delete records in a file that does not allow duplicate keys can be reduced if a storage index is built.

When you change the key fields in a file that allows duplicate keys, the system does extra processing to keep the duplicate keys in order by relative record number. If you create a key that has many duplicates, this extra processing can cause performance problems. You should design applications to avoid this problem. For example, if duplicate key ordering is not important, it may be better to delete the old record first and then add the updated record, instead of doing a single update operation (but you must make the file large enough to hold the deleted record slots).

Key Length: As the length of the key increases, the time required to process the file also increases. This relationship is a result of factors such as storage indexes, size of the index area, duplicate-key checking, and maintenance of the index.

Keysorting for Multiple-Index Files: Because of the considerations just mentioned, it is important to keep the overflow areas of the indexes relatively small (10K to 20K bytes). The length of an entry in the index is equal to the key length plus 3 bytes (for the binary address). For example, if you add 1000 entries of a 5-byte key, the overflow area could increase 8K bytes:

```
256 bytes per sector / (5 + 3) bytes per entry =
   32 entries per sector

1000 entries / 32 entries per sector =
   32 sectors for the overflow area = 8K bytes
```

One way to keep the overflow area small is to schedule regular keysorts of the indexes. Keysorting occurs when a STOP SYSTEM command, a KEYSORT command, or an IPL file-rebuild occurs. The KEYSORT procedure can be run concurrently with any programs accessing the file, as long as the file is defined as being shared.

Note: Keysorting when available disk space is limited will take longer if the index is large compared to the available disk space.

For more information about key sorting files, see "Keysorting" on page 8-20.

Accessing Files

Accessing files involves the following concepts:

- Current record pointer
- Nonkeyed and keyed processing
- File access methods

Current Record Pointer

When you open a file, the system establishes a current record pointer. The current record pointer points to a particular record position within the file. It positions a record for reading and maintains that position for updating or deleting the record.

The current record pointer is updated by the SSP when an input operation is completed successfully or when an end-of-file completion code is returned.

The current record pointer is always at one of the following positions:

- Beginning of file. In this position, the pointer is before the first record in the file. If a request to read the next record occurs, the first record in the file is read. After a file is opened, the pointer is set at the beginning of the file.
- End of file. In this position, the pointer is beyond the last record in the file. If a request to read the previous record occurs, the last record in the file is read.
- Record position. In this position, the pointer points to a record position in the file. The record at that position may be an active record or a deleted record.

Nonkeyed and Keyed Processing

Files can be processed either without using a key (nonkeyed processing) or according to the value of the key (keyed processing).

Nonkeyed Processing

In nonkeyed processing, the records are sequenced in the order in which they are stored in the file. This sequence allows records to be processed either randomly or consecutively. The current record position is based on the relative position of the record in the file. When the file is opened, the current record pointer is set at the beginning of the file. The current record position changes as read operations occur for the file. Update, delete, and release operations are performed on the record at the current record position. Add operations do not change the position of the current record pointer.

Nonkeyed processing allows the following operations for sequential, direct, and indexed files (except where noted):

- Read the first record in the file
- Read the last record in the file
- Read the next record in the file
- Read the previous record in the file
- Read the record at the current record position + N
- Read the record at the current record position - N
- Read the record at the relative record number
- Add a record at the end of data (except for direct files)
- Add a record at the relative record number
- Update the current record
- Delete the current record
- Release the current record

Note: Not all of the high-level languages allow every operation in the preceding list. Refer to the appropriate language manual for information about the operations allowed in a particular language.

Keyed Processing

In keyed processing, the records are in sequence by their key values. This sequence allows records to be processed either randomly by key or sequentially by key. During file processing, a current record pointer is maintained. When the file is opened, the pointer is set at the beginning of the first key in the index. The current record position changes as read operations occur for the file. Update, delete, and release operations are performed on the record whose index entry is at the current record pointer. Add operations do not change the position of the current record pointer.

Keyed processing allows the following operations for indexed files:

- Read the record that has a specific key value
- Read the first record in the file
- Read the last record in the file
- Read the next record in the file
- Read the previous record in the file
- Read the record that has an equal or greater key value
- Read the record that has a greater key value
- Read the record if it has a key equal to a specified value
- Add a record at the end of data
- Update the current record
- Delete the current record
- Release the current record

Note: *Not all of the high-level languages allow every operation in the preceding list. Refer to the appropriate language manual for information about the operations allowed in a particular language.*

File Access Methods

Before the file is created, you must select an access method. **Access method** is the general term used to describe the way a program retrieves disk records for processing. The access method defines a set of functions that the high-level language program uses to retrieve records from the file. The access methods are:

- Consecutive
- Sequential by key
- Random by relative record number
- Random by key
- Generalized access

Do not confuse these access methods with file organizations. However, the file organization plays a significant role in determining which access method can be used in a given program. The following table indicates which access method can be used for each file organization.

Processing Method	**Sequential Organization**	**Direct Organization**	**Indexed Organization**
Consecutive	Yes	Yes	Yes
Sequential by key	No	No	Yes
Random by relative record number	Yes	Yes	Yes
Random by key	No	No	Yes
Generalized access method (nonkeyed processing)	Yes	Yes	Yes
Generalized access method (keyed processing)	No	No	Yes

Consecutive Access Method

The consecutive access method reads records in the order in which they appear in the file, one after another from first to last (see Figure 8-6).

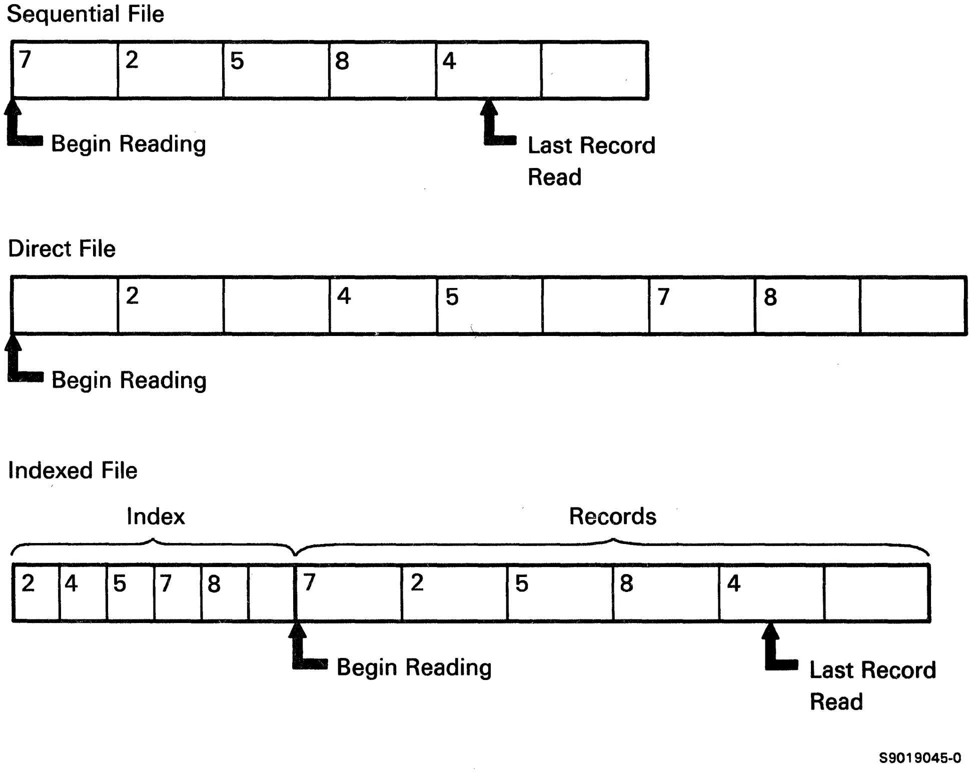


Figure 8-6. Consecutive Access Method

The consecutive access method can be used for all three file organizations.

Sequential Files: When a sequential file is accessed consecutively, the space at the end of the file that is reserved for new records is **not** read. Therefore, the last record to be read in the sequential file in Figure 8-6 is record 4.

Direct Files: When a direct file that is not delete-capable is accessed consecutively, the gaps that were left for new records **are** read by the program. Therefore, the program must test for a blank record each time it reads a record. When a program accesses a direct file that is delete-capable, the deleted records are bypassed.

Indexed Files: When an indexed file is accessed consecutively, the program ignores the index portion of the file as it reads the records. However, if keys are changed by update operations or if records are either added or deleted, the system automatically updates all indexes for the file. The space at the end of the file that is reserved for new records is **not** read. Therefore, the last record to be read in the indexed file in Figure 8-6 is record 4.

Sequential Access by Key

When an indexed file is accessed sequentially by key, the program processes the records according to the sequence of the keys in the index (see Figure 8-7).

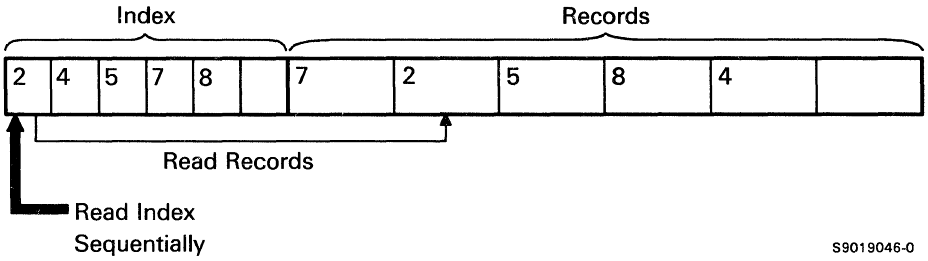


Figure 8-7. Sequential Access by Key

Although the keys are accessed consecutively, the records are accessed randomly, because the index entries are sorted, but the records are not. The records are processed in ascending order of key value.

If there are duplicate keys in the index, the records with duplicate keys are processed in the order in which they were placed into the file. For more information about duplicate keys, see "Duplicate Keys" on page 8-27.

In a delete-capable file, the deleted records are bypassed.

Sequential Access by Key within Limits: Normally, the sequential-by-key access method is used to process all records in a file. However, you can specify the upper and lower limits of the key values of the records to be accessed sequentially by key in ascending order (see Figure 8-8). You can also specify the limit of the key values for records to be accessed sequentially by key in descending order (see Figure 8-9 on page 8-38).

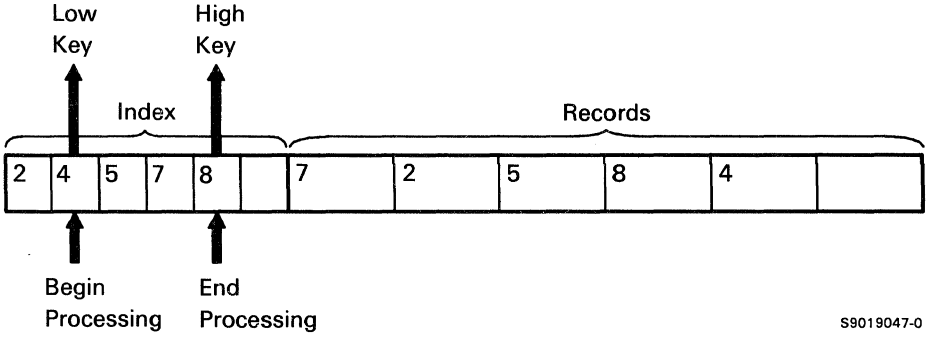


Figure 8-8. Sequential Access by Key in Ascending Order within Limits

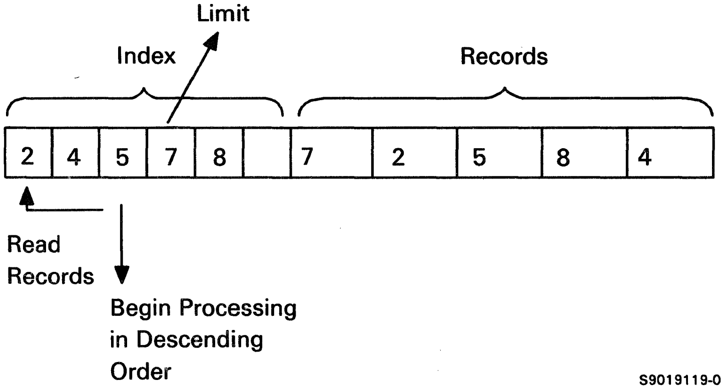


Figure 8-9. Sequential Access in Descending Order by Key within Limits

Accessing an indexed file sequentially by key within limits allows you to process a specific group of records in the file. You can access the file in ascending order or descending order.

The following describes the limits when accessing files in ascending or descending order:

- *Ascending order:* The lower limit is the key value at which processing begins, and the upper limit is the key value at which processing ends. If there are duplicate keys in the index, the first duplicate of the appropriate key is used as the lower limit, and the last duplicate of the appropriate key is used as the higher limit.

- *Descending order:* Processing begins at the key value just below the limit and continues in descending order to the first record in the file. If there are duplicate keys in the index, the first key returned will be the last key in the group of duplicates.

You specify the limits for accessing the file, and the limits you specify remain in effect until one of the following occurs:

- New limits are set.

- A random read operation occurs.

- The file is closed.

Random Access by Relative Record Number

The random by relative record number access method is used to read only the record the program needs, and all the other records in the file are ignored. This allows disk records to be processed in an order you determine during the processing of the program. The relative record numbers indicate the positions of the records in the file relative to the beginning of the file. Relative record numbers are positive whole numbers that the system converts into the disk addresses of the records.

All three file organizations can be accessed randomly by relative record number. When a delete-capable file is accessed randomly by relative record number, the deleted records cannot be read by the program. However, the deleted records do take up space in the file, so they affect the relative record number of other records.

Sequential Files: Normally, sequential files are accessed randomly by relative record number when only a few of the records in the file are to be processed and you know the relative record numbers of the records.

Direct Files: Figure 8-10 shows an example of a direct file accessed randomly by relative record number. In this example, the direct file contains customer records that are stored at record positions based on the customer number. The program receives a request to read the record for customer 4.

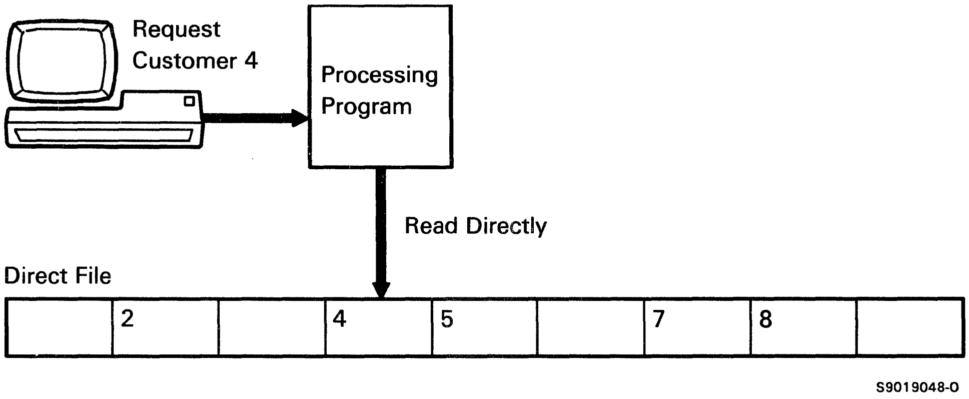


Figure 8-10. Random Access by Relative Record Number for a Direct File

The customer number is used as the relative record number. The program retrieves the requested record (the fourth record relative to the beginning of the file) without reading any other record. Thus, blank records are not a problem when retrieving records from a direct file, because only the desired records are read.

Indexed Files: When an indexed file is accessed randomly by relative record number, the records are processed by their relative record number values. If any of the keys in the records are changed by update operations, or if any records are added to or deleted from the file, the system updates all indexes for the file.

Random Access by Key

Only indexed files can be accessed randomly by key. Figure 8-11 on page 8-40 shows an example of this access method. In this example, the program receives a request to read the record for customer 4.

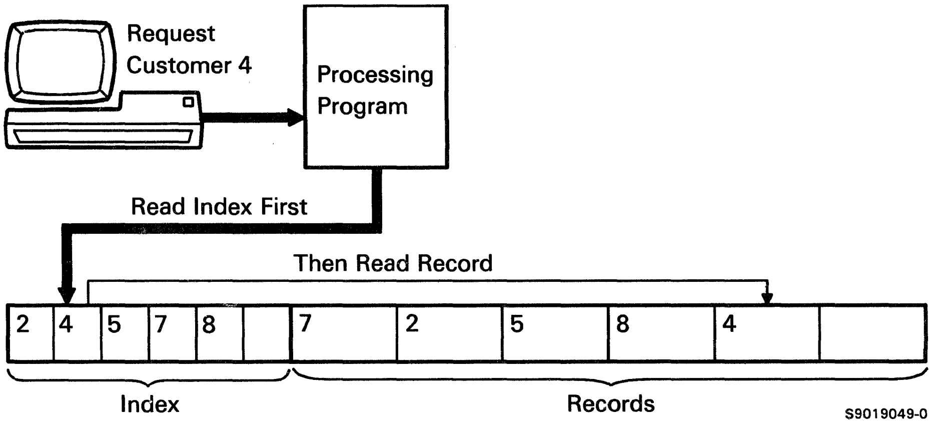


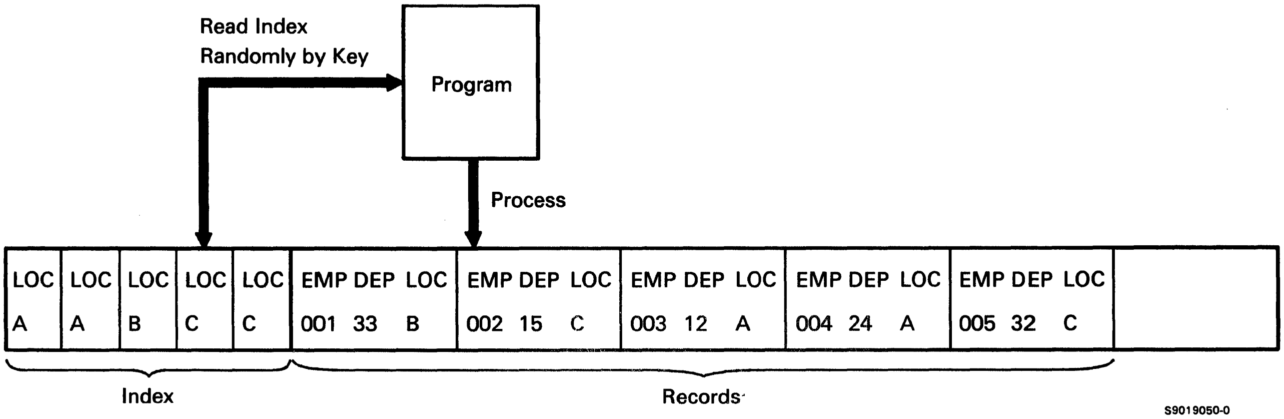
Figure 8-11. Random Access by Key for an Indexed File

The customer number is used as the key. The record for customer 4 is retrieved in two steps. First, the SSP searches the index for a value that matches the requested key. The index entry with the matching key value also contains the relative record number of the record for that key. Second, the SSP reads the record at that record position from the data portion of the file.

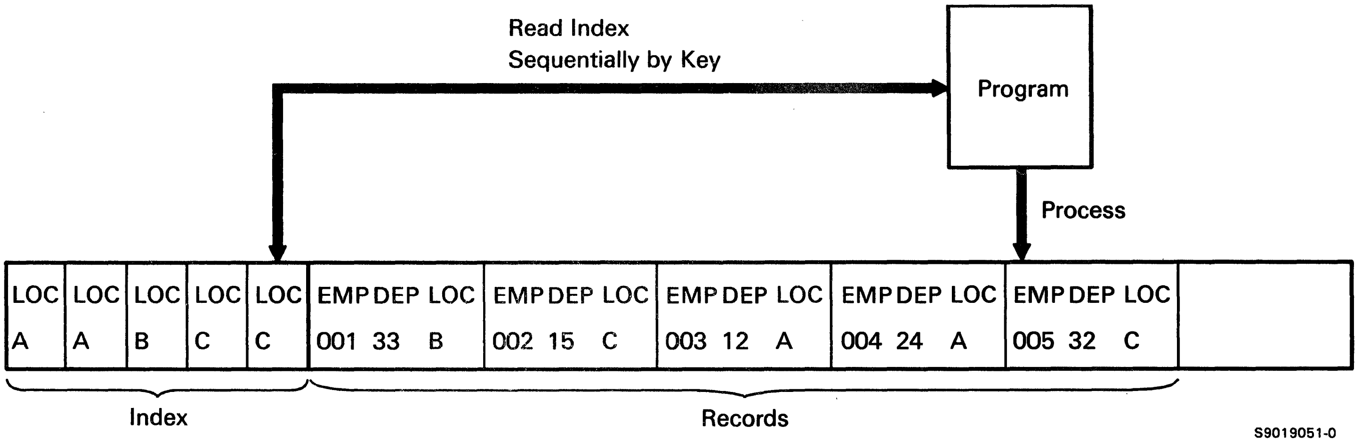
Generalized Access Method

The generalized access method is used to process a disk file randomly, consecutively, or sequentially while updating, deleting, or adding records. When using the generalized access method, you can do either nonkeyed or keyed processing.

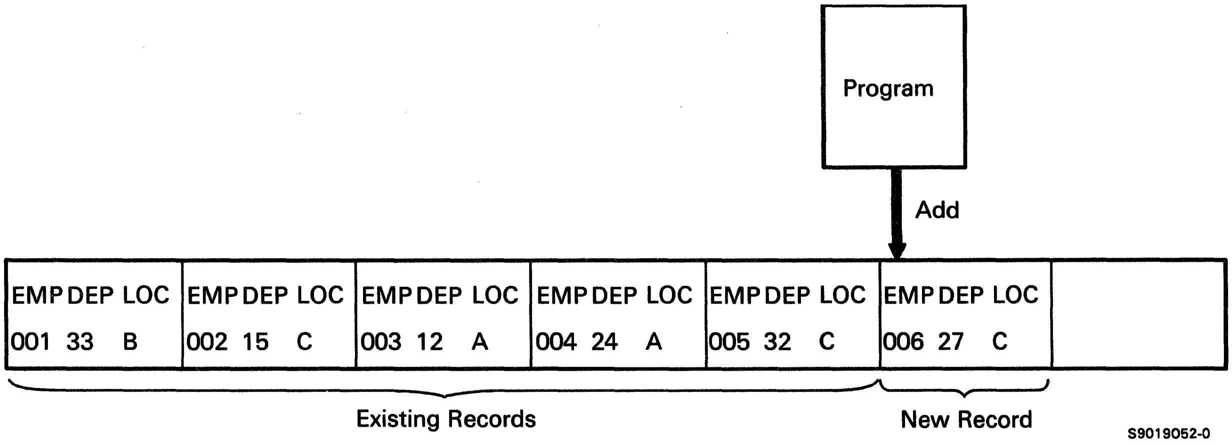
For example, assume that you have a file of employee records containing employee number, department number, and location code. Also, assume that the file is an indexed file with location code as the key. Using the generalized access method with keyed processing, your program can process the records for all the employees at a particular location. First, the program can access the file **randomly by key** to find the first record that has the particular location code:



After the program finds that record, it can access the file **sequentially by key** to process the records for all the other employees at that location:



While processing the file, the program could also add records to the file:



EMP DEP LOC	EMP DEP LOC	EMP DEP LOC	EMP DEP LOC	EMP DEP LOC	EMP DEP LOC	
001 33 B	002 15 C	003 12 A	004 24 A	005 32 C	006 27 C	

Sequential or Direct Files: The nonkeyed generalized access method allows you to process sequential or direct files either consecutively or randomly by relative record number.

Indexed Files: The nonkeyed generalized access method allows you to process indexed files by using:

- Consecutive access
- Random access by relative record number

The keyed generalized access method allows you to process indexed files by using:

- Sequential access by key
- Random access by key

Specifying the Generalized Access Method in High-Level Languages: The following table lists the ways in which various programming languages use the generalized access method on the system:

Programming Language	**Generalized Access Method**
Assembler	Disk file macroinstructions
BASIC	Used internally
COBOL	DYNAMIC access mode
FORTRAN IV	READ and DEFINE FILE statements
RPG II	Full-procedural files

You can also use logical files to process the same file by different access methods. Refer to "Using One File As Two Or More Logical Files" on page 8-89 for more information about multiple logical files.

Choosing a File Organization

When you create a file, you have to decide which file organization to use (sequential, direct, or indexed). You should choose the file organization that allows the file to be used most efficiently in the application. Among the factors that you should consider are:

- File usage
- Volatility of the file
- Activity of the file
- Disk space
- Processing speed

File Usage

Usage is the overriding factor in the selection of a file organization. Usage involves the type of data stored in the file, the access method used to process the data, and the type of application. Therefore, the design factors include the following considerations:

- What kind of data is stored in this file? Is it permanent data (such as a master file) or temporary data (such as a transaction file)?
- What access method is required?
- What does the type of application require?

Master File

A master file is relatively permanent and is often used in several jobs with several other files. When you choose an organization for a master file, consider the following processing requirements:

- How are the other files that are processed against the master file organized? If the other files are in an ordered sequence (that is, if they are sorted in the same sequence as the master file), the master file could be processed consecutively. Therefore, a sequential or indexed organization for the master file would be most efficient.

  If the other files that you process against the master file are in an unordered sequence, the master file needs to have a direct or indexed file organization, and the access method should be random. Direct files offer the advantage of faster processing than indexed files, because direct files require fewer accesses to the disk. Processing an indexed file randomly requires two disk accesses: one to read the index and another to read the record. Processing a direct file randomly requires only one disk access per record (unless the record has synonyms).

- How do other jobs process the file? If the master file is used in several jobs and is accessed both consecutively and randomly, either a direct organization or an indexed organization is better than a sequential organization.

- Does the master file require sorting to be processed by a subsequent job? If so, consider that when your master file is direct or indexed, the file that is produced by the sort is a sequential file. You would keep the original unsorted direct or indexed file as the master file, and use the sorted sequential file for that one job.

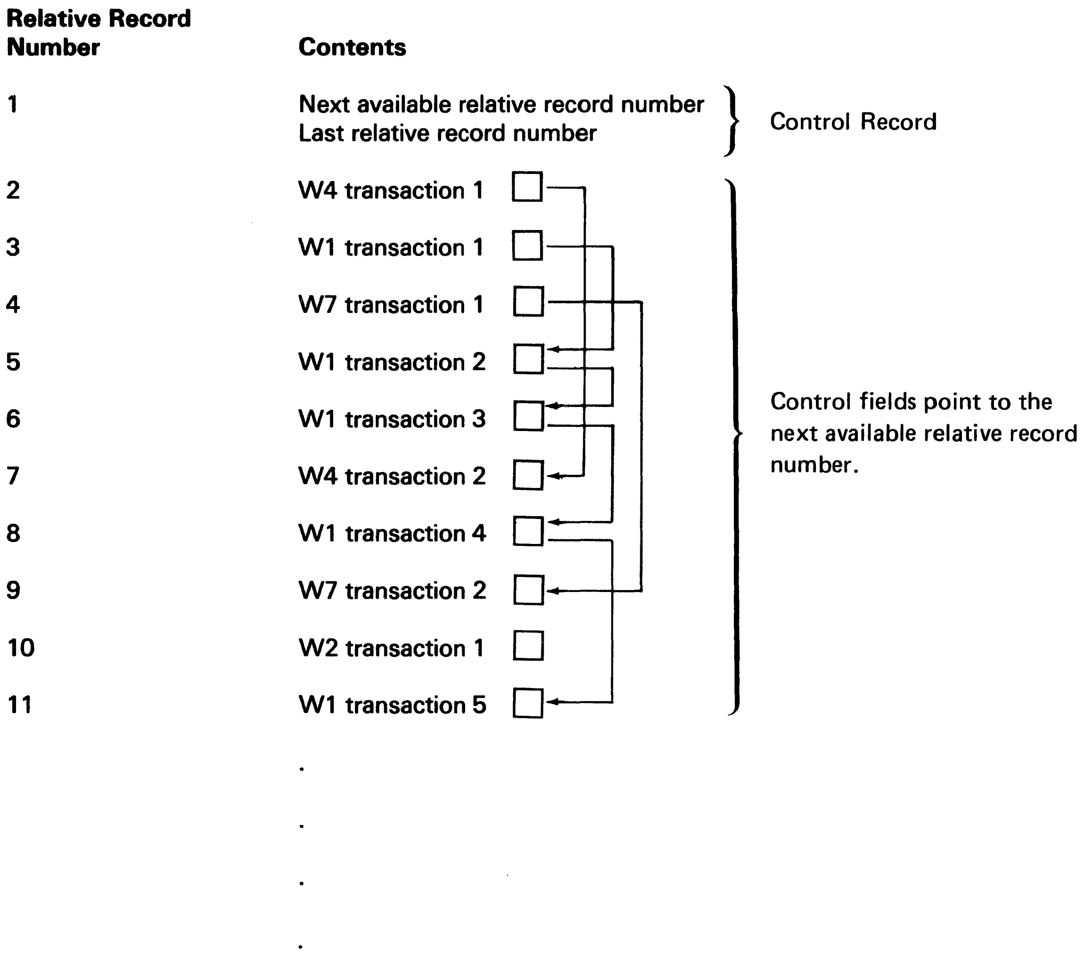
- Do you want display station operators to inquire into the master file? If so, how important is response time? To ensure the shortest possible response time, the file should be direct because reading a record from a direct file requires only one disk access if the record has no synonyms. If the record has numerous synonyms, reading a record can require several disk accesses. In that case, a direct file might not provide shorter response time than an indexed file that is accessed randomly by key.

Transaction File

Transaction files are less permanent than master files. Typically, they are used to update master files. Transaction file records are frequently logged in history files to keep records of business activities. An example of a transaction file is a cash receipts file for an accounts receivable application. An example of a more permanent transaction file is an open-item accounts receivable file. These transactions are usually maintained to show detail on reports such as statements sent to customers and aged trial balances.

Typically, transaction files entered from display stations in an interactive environment are direct files. The reason for using direct files is that the operator can usually expect a short response time when paging through a file, adding records, deleting records, or reviewing all or part of a file.

For example, the transaction file created by a WSU program is a direct file that has records separated **logically** by display station. Several display station operators can enter transactions concurrently, and these transactions become mixed **physically** in the file. But each transaction record contains a control field that identifies the next available relative record number and the relative record number of the last record entered at that display station (see Figure 8-12). This control field allows the transactions from each display station to be chained so that each operator can access the records entered from his display station.



Relative Record Number	Contents
1	Next available relative record number Last relative record number
2	W4 transaction 1 □
3	W1 transaction 1 □
4	W7 transaction 1 □
5	W1 transaction 2 □
6	W1 transaction 3 □
7	W4 transaction 2 □
8	W1 transaction 4 □
9	W7 transaction 2 □
10	W2 transaction 1 □
11	W1 transaction 5 □
	.
	.
	.
	.

Control Record

Control fields point to the next available relative record number.

Figure 8-12. Direct Transaction File Using Control Fields

A direct transaction file can also be organized so that each display station has its own work area (see Figure 8-13).

Relative Record Number	Contents	
1	Next available relative record number First relative record number Last relative record number	W1 work area
2	Next available relative record number First relative record number Last relative record number	W2 work area
.	.	
.	.	
.	.	
10	W1 transaction 1	
11	W1 transaction 2	
12	W1 transaction 3	
.	.	
.	.	
.	.	
110	W2 transaction 1	
111	W2 transaction 2	
.	.	
.	.	
.	.	

Figure 8-13. Direct Transaction File Using Separate Work Areas

Notice that a control record is required for the records entered from each display station. This direct file reduces the possibility that a display station operator may try to access a record within a sector assigned to another program. However, the number of records that can be entered from a display station is limited, and gaps can exist between the end of one section of the file and the beginning of the next section.

Access Method

The file organization is determined when the file is created, but the access method used to process that file can vary from one program to the next. If you create a file for an existing application in which a particular access method is used, your choice of file organization may depend on that access method.

If random access is used, the program can access specific records in a file, either by relative record number or by key, without having to process the entire file. For example, when display station operators are processing telephone orders, they want to access specific data in an indexed file by using the customer number as a key.

If consecutive access is used, the program processes all the records in the file. For example, to produce invoices for all customers, the program would access the file consecutively. In that case, the file organization could be sequential.

Consecutive access takes advantage of the disk cache because each read miss causes a whole page of consecutive disk sectors to be read into the cache.

Application

The type of application can also affect your choice of file organization. For processing all the records in a transaction file arranged in customer-number sequence and used as input for a report, sequential file organization might be best. For processing a master file of 10,000 records that has few additions or deletions and that is used for high-speed inquiry, direct organization would be appropriate. For processing a master file of employee addresses to print addresses on 15 percent of the payroll checks, indexed organization could be used.

Batch Processing and Interactive Processing: An important distinction between types of applications is whether the application uses batch processing or interactive processing.

In batch processing, groups of data are accumulated and processed at specific times, such as daily, weekly, or monthly. Certain applications, such as a payroll application that is processed once a week, are perfectly suited for batch processing.

In interactive processing, individual records or transactions are processed at the time the transaction occurs. For example, in an interactive order entry application, as soon as a sale is made, the quantity of merchandise sold is subtracted from the quantity on hand in the appropriate master file.

The type of processing might require a particular file organization. Batch processing, which does not require random access to the data, might require sequential files. Interactive processing requires immediate access to the file, so direct or indexed files is more appropriate.

Volatility of the File

The term **volatility** refers to the number of additions to and deletions from the file. Highly volatile files might require direct organization because a record can usually be added or deleted with fewer disk accesses than for other disk organizations, and fewer disk accesses should help shorten response time.

For example, adding a record to an indexed file requires:

1. Scanning the index, including added index entries, to make sure that the record does not already exist

2. Reading the data area where the new record will reside

3. Writing the record

4. Writing the new index entry

Adding a record to a direct file might require:

1. Reading a control record to find the next available location.

2. Writing the data.

3. Updating the control record. Updating the control record after each record is added makes it easier to program for recovery but requires an additional disk access.

The direct file might require disk space to allow for synonym records, and multiple disk accesses could be required for those records. Processing the direct file should be faster than processing a sequential file or an indexed file.

Note: The system can rebuild an index automatically, but it cannot rebuild control records and chain fields. Therefore, if your file requires control records or chain fields, direct organization may not be a good choice.

Activity of the File

The term **activity** refers to the frequency of accesses to the file. Activity is measured as a fraction in which the number of transactions to the file is divided by the number of records in the file. This fraction is usually expressed as a percentage. For example, if a file has 600 records and 1200 transactions are processed against the file each day, the activity of the file is 200 percent per day.

A relatively inactive file might be accessed randomly and, therefore, have a direct or an indexed organization. However, as activity increases, consecutive access becomes more advantageous because the likelihood increases that the record to be processed is available in a buffer and can, therefore, be accessed without reading from or writing to disk. Therefore, very active files could be either sequential (accessed consecutively) or indexed (accessed sequentially by key).

The total activity of an indexed master file might be reduced by sorting a transaction file so that only one retrieval of a master file record is needed for a group of transactions that have the same key. Activity might also be reduced by sorting the data in the master file to match the sequence of the index.

Disk Space

A sequential or direct file takes less space than an indexed file because an indexed file requires additional space for the index. Therefore, if disk space is a prime concern, you should consider using a sequential or direct file organization.

Processing Speed

If processing speed is the most critical factor, you would probably not want an indexed file organization. A sequential file organization is faster for consecutive access, and a direct file organization is faster for random access.

Sequential file organization coupled with a large cache page size gives faster processing speed when reading or reading/writing records. When only writing records, it is faster to not use the cache.

File Attributes

File attributes include the following:

- Scratch files
- Job files
- Resident files
- Extendable files
- Delete-capable files

Scratch Files

Scratch files are files that have RETAIN-S specified on the FILE OCL statement. Scratch files are usually used as temporary work files for a single job step. At the end of the job step in which the scratch files are created, the disk space used by the scratch files is released (see Figure 8-14). Thus, a scratch file can be used in only one job step. Scratch files cannot be shared by other programs.

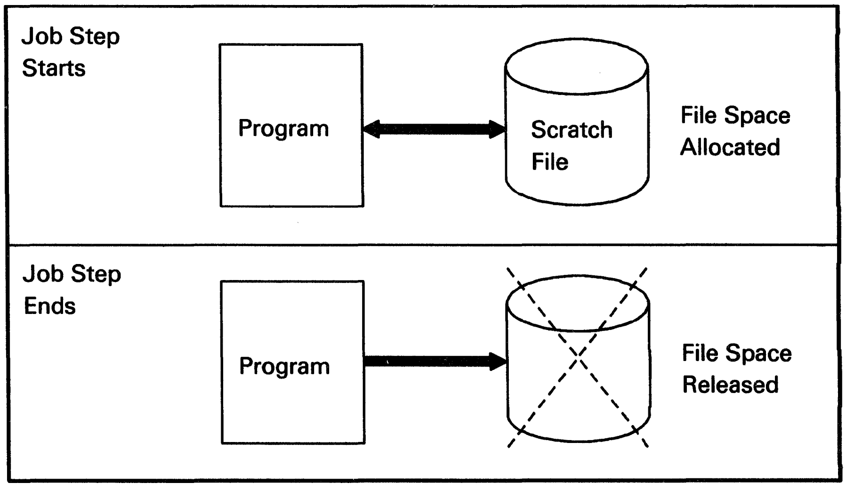


Figure 8-14. Scratch Files

Job Files

Job files are files that have RETAIN-J specified on the FILE OCL statement. Job files exist from the time they are created until the job ends or until they are deleted. The disk space used by the job file is released when the last job step in the job ends (see Figure 8-15).

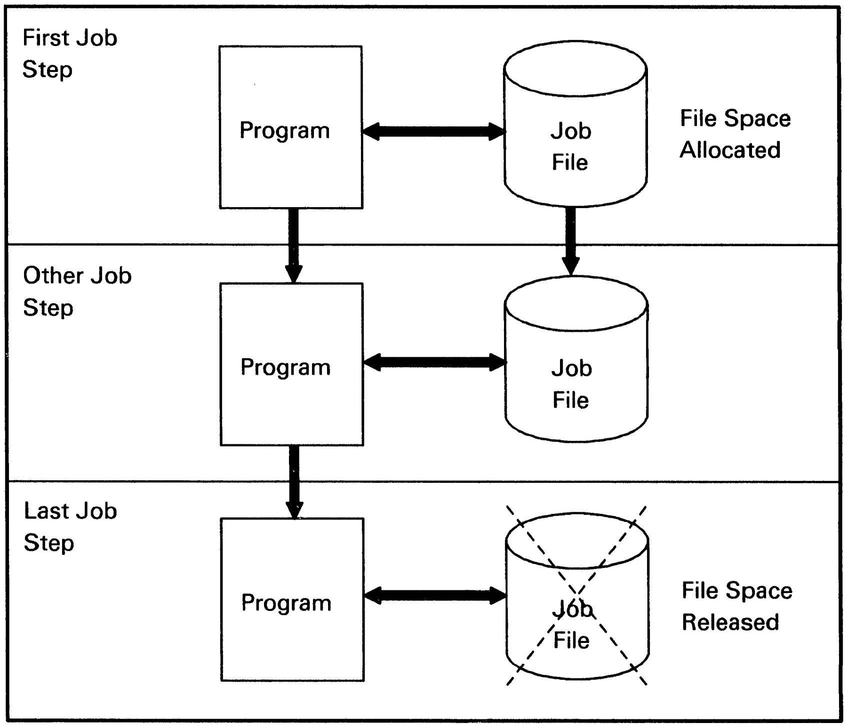


Figure 8-15. Releasing Disk Space Used by a Job File S9019036-0

The disk space used by a job file can also be released when the RETAIN-S parameter of the FILE OCL statement is specified in a particular job step for the file.

Job files usually contain a limited number of records from a particular file, and these records are used by various programs within the same job. For example, portions of a master file can be placed into a job file, and this job file can be used by many programs within the same job. A job file cannot be shared by programs in different jobs.

If you use the same job file name for a sort output file in another FILE OCL statement within the same job (different job step), sort does not allocate a new file but uses the existing job file. If you specify a size for the sort file different from the original FILE OCL statement, the system will ignore the new size specified and use the size of the original job file in the FILE OCL statement. This may result in an error message telling you that the output file is too small. To avoid this error, you can specify EXTEND on the original FILE OCL statement so that the sort output file will extend if more space is required.

Reserving Space for Scratch and Job Files

If you are running a job and want to ensure that you have enough disk space to create new scratch and new job files, you can use the RESERVE OCL statement to ensure that a specific amount of disk space is reserved for these files. The system reserves the number of blocks specified on the RESERVE OCL statement for new scratch files and job files.

The *System Reference* manual describes the RESERVE OCL statement.

Resident Files

Resident files are files that have RETAIN-T specified on the FILE OCL statement. Resident files can be considered permanent files. For example, a master file is a resident file. Resident files allow you to share data among various jobs. You can save and restore resident files by using diskettes. Resident files remain on disk until one of the following occurs:

- The DELETE procedure is run.
- The file retention parameter on the FILE OCL statement is changed to S (scratch) in a particular job step, and the file is allocated by the job. Note that the file cannot be shared with another job.
- The FREE command is issued when using BASIC.

Using Resident Files from One Job Step to Another

If you specify JOB-YES on the FILE OCL statement for a resident file, the file is kept for other job steps until the end of the job. The other parameters on the FILE statement remain in effect until the end of the job or until they are overridden by a FILE statement in another job step that has the same NAME parameter. The LOCATION and size (RECORDS or BLOCKS) parameters cannot be overridden after the file is created. New parameters can be added by a FILE statement in another job step that has the same NAME parameter.

If you specify JOB-YES on a FILE statement in another job step later in the job, the parameters specified on that FILE statement remain in effect until the end of of the job or until they are overridden in a later job step in the job. The parameters specified with JOB-YES on the FILE statement in a previous job step in the job no longer apply.

The JOB-YES parameter can be specified only for FILE statements that are outside the LOAD and RUN statements. Placing a FILE statement outside a LOAD and RUN pair causes the system to try immediately to acquire ownership of the specified disk file for use by the job. If the FILE statement is within the LOAD and RUN pair, the system waits until it encounters the RUN statement before it acquires ownership of the file.

If you specify JOB-NO on a FILE statement in a job step, the program gives up ownership of the file at the end of the job step.

The following example shows how the JOB-YES parameter affects the FILE OCL statement for a new file during each of three job steps.

```
* Job step 1:
// FILE NAME-A,JOB-YES,RECORDS-10,EXTEND-50,DBLOCK-200,
//      DISP-NEW
// LOAD PROG1
// RUN
```

In job step 1, the JOB-YES parameter means that the FILE OCL statement parameters specified for file A remain in effect until the end of the job or until they are overridden in a later job step. Those parameters are:

- The size of the file is 10 records.

- The file will be extended by 50 records whenever additional space is needed.

- 200 records will be moved between main storage and disk for each input/output operation.

Suppose that the program does not use file A during job step 1. In that case, file A is not created. Now suppose that job step 2 includes the following FILE statement for file A:

```
* Job step 2:
// LOAD PROG2
// FILE NAME-A,BLOCKS-20,EXTEND-60,DISP-NEW
// RUN
```

Because file A was not created in job step 1, the BLOCKS parameter in job step 2 is used instead of the RECORDS parameter in job step 1. Therefore, if file A is created in job step 2, its size will be 20 blocks instead of 10 records. The EXTEND parameter in job step 2 also overrides the EXTEND parameter in job step 1, but only for a single job step because JOB-YES is not specified on the FILE statement for job step 2. Therefore, if file A is created in job step 2, it will be extended by 60 blocks instead of 50 records whenever it requires additional space.

If file A is not created in job step 2, the EXTEND parameter is reset to 50 at the end of the job step. The DBLOCK parameter specified in job step 1 remains in effect in job step 2 because it was not overridden.

Suppose that file A is not created in job step 2 but is created in job step 3. Also suppose that job step 3 contains no FILE statement:

```
* Job step 3:
// LOAD PROG3
// RUN
```

In this case, the BLOCKS parameter stays at 20 because the size parameter (RECORDS or BLOCKS) and the LOCATION parameter are the only exceptions to the rule that parameters are reset to the value specified on the FILE statement with JOB-YES specified. (Incidentally, if the size and location parameters are specified with JOB-YES on a FILE statement for a file that already exists, the system ignores the size and location parameters.) The EXTEND parameter is reset to 50 because that is the value specified on the FILE statement with JOB-YES in job step 1. The DBLOCK parameter specified in job step 1 remains in effect in job step 3.

Resident files with the JOB-YES parameter can cause a file lock when the file is shared. If one program within a job acquires the file as a shared file, another program in another job cannot acquire the same file as a nonshared file until:

- The program that acquired the file as a shared file goes to end of job.

- JOB-NO is specified for a particular job step in the job that was sharing the file, and that job step ends.

Following is an example of OCL statements used with two jobs sharing a file for which JOB-YES is specified. In the example, program C wants exclusive use of file A; therefore, program C must wait until program B in job Y ends.

```
     Job X                               Job Y

  * Job step 1 for job X
  // FILE NAME-A,JOB-YES,DISP-SHR     // FILE NAME-A,JOB-YES,DISP-SHR
  // LOAD PROGA                       // LOAD PROGB
  // RUN                              // RUN

  * Job step 2 for job X
  // LOAD PROGC
  // FILE NAME-A,DISP-OLD
  // RUN
```

A file deadlock can also occur if two or more jobs are using two or more resident files with JOB-YES specified. If the jobs try to use files that do not permit sharing and have the JOB-YES parameter specified, both jobs may have to wait.

If you are running a job that contains a MRT procedure and you want the file to be used by the other job steps, the MRT procedure must contain the FILE OCL statement on which the JOB-YES parameter is specified.

The *System Reference* manual has more information about using the FILE OCL statement.

Extendable Files

An extendable file is a disk file for which the system automatically attempts to allocate more space each time the file becomes full. Specifying an extendable file prevents your program from ending abnormally when there is no room in the file to add additional records.

Specifying an Extendable File

You can specify a file as an extendable file by either of the following methods:

- FILE OCL statement. The EXTEND parameter specifies the number of blocks or records to extend the file.

- BLDFILE procedure. The number of blocks or records to extend the file is a parameter.

The extension value must be a numeric value that indicates the amount of additional space needed for the extension. If the file size was specified in blocks when the file was created, the extension value is in blocks. This value must be large enough to contain at least one record. If the file size was specified in records when the file was created, the extension value is in records. The amount of the file extension is the number of records or blocks specified, rounded up to a block boundary.

If you specify an extension value when the file is created or when an existing file is updated with new information, the extension value becomes an attribute of the file. In that case, the file is extended, if required, by any program using the file. Note that when an existing file is updated with new information, the old extension value is not saved for the file. The value must be specified again, as if the file were a new file.

You can also specify an extension value when your program uses an existing file. This allows you to override any existing extension values or to specify a new extension value for a file. For example, if a file was not specified as extendable when it was created, you can use the FILE OCL statement to make the file extendable while your program is using the file.

You can prevent your program from extending a file by specifying 0 on the EXTEND parameter of the FILE OCL statement.

If the file is being shared and the various programs use different EXTEND values on their FILE OCL statements, the system uses the extension value of the program that caused the file to become full. When the file is extended, all programs sharing the file take advantage of the extra space, even if the EXTEND parameter was not specified on the FILE OCL statement for each program.

A file can be extended any number of times. A file is not extended if there is not enough disk space or if a disk input/output error occurs. Once initiated, the extension cannot be canceled.

Automatic File Extension

The following conditions can cause the system to automatically extend a file:

- The addition of records to a sequential or indexed file.

- Reading a record from a direct file for updating when the specified relative record number is beyond the end of the file. If the relative record number is greater than the file size plus the extend value, the system does not attempt to extend the file. In that case, an invalid-record-number completion code is returned to the program. (An end-of-file condition is returned to BASIC programs.)

- Writing a record to a delete-capable file when the specified relative record number is beyond the end of the file. If the relative record number is greater than the file size plus the extend value, the system does not attempt to extend the file. In that case, an invalid-record-number completion code is returned to the program. (An end-of-file condition is returned to BASIC programs.) For a discussion of delete-capable files, see "Delete-Capable Files" on page 8-57.

What Happens When Extendable Files Become Full

The following list describes how the system handles various types of extendable files when they become full:

- Scratch and job files in the reserve area. The reserve area is an area on disk set aside for scratch and job files used by a job. The system displays a message to the system operator, and processing stops. The system operator can choose an option to extend the file. In this case, the system copies the file to a larger area on disk outside the reserve area, and the program continues processing. If the operator does not want the file to be extended, the operator can cancel the program or choose an option that returns an end-of-extent completion code to the program. The RESERVE OCL statement in the *System Reference* manual has more information about the reserve area.

- Sequential, direct, and alternative index files. The system attempts to allocate the additional space immediately following the file. If that is not possible, the system copies the file to a larger area on disk and frees the original space occupied by the file.

- Indexed files. The system copies the indexed file to a larger area on disk and then frees the original space occupied by the file. If the file has alternative indexes, they are also extended.

When a file is extended, a message is placed in the history file, stating that the file was extended.

If the file cannot be extended because of a lack of disk space or because of a disk error, an end-of-extent completion code is returned to the program.

Delete-Capable Files

A delete-capable file is a file in which programs can delete records. The reason for specifying a file as delete-capable is to allow your programs to delete unwanted records when processing the file. If, for some reason, you need the data that was in a deleted record, do not use a delete-capable file. Instead, have your program place a delete code in the record. Then, when the file is processed, your program can check for this code. For information about using a delete code, see "Providing for Deletion of Records" on page 7-9.

Creating a Delete-Capable File

To create a delete-capable file, you can do any of the following:

- Specify the DFILE parameter on the BLDFILE procedure.
- Specify DFILE-YES on the FILE OCL statement.
- In a BASIC program, specify the OUTIN parameter on the OPEN statement if no FILE OCL statement is used.

When a delete-capable direct file is created, all bytes in the file are set to hex FF. That is, all the bits for every character in every record are set on.

Deleting Records from a Delete-Capable File

When you delete records from a delete-capable sequential or indexed file, the records are not physically removed from the file (unless you use the COPYDATA procedure to remove them). Instead, the records are filled with hex FF. Therefore, the data that was in the record before it was deleted is no longer available to the program.

When a record is deleted from a multiple-index file, the system deletes the key for that record from all indexes.

In RPG II, if you use an address output (addrout) file to access records in a file and you delete a record, the record is deleted from the file you access but not from the addrout file. To delete the record from the addrout file, you must delete the entry for the record in the addrout file or re-create the addrout file.

The following table lists the statements used by various programming languages to delete records from a delete-capable file:

Programming Language	**Statements Used to Delete Records**
Assembler	$PUTD macro with OP-DELETE parameter
BASIC	DELETE statement
COBOL	DELETE statement
FORTRAN IV	RTNCD subprogram
RPG II	U in column 15 of file description specifications DEL in columns 16-18 of output specifications

Processing a File Containing Deleted Records

When a file containing deleted records is processed consecutively or sequentially by key, each deleted record is bypassed and the next record in the file is read.

When a file containing deleted records is processed randomly by key or randomly by relative record number, a record-not-found completion code is returned to the program when a deleted record is accessed.

Adding Records to a Delete-Capable File

You can add records to delete-capable sequential, direct, and indexed files by using relative record numbers.

The system does not allow a record having hex FF as its first byte to be written to a delete-capable file during an add or update operation. If the first byte of the record contains hex FF, an invalid update/add/output completion code is returned to the program.

Using RPG II to Add Records to Delete-Capable Files: For sequential and direct files, you must first place the relative record number of the record to be added to the file into the RECNO field. The RECNO field is defined on the continuation line of the file description specifications. The relative record number must be the record number of a deleted record. Then, to add a record to the file, you code output specifications that contain ADD in columns 16 through 18. RPG II uses the relative record number from the RECNO field to locate where the record is to be added to the file. If the relative record number is not the number of a deleted record, a halt occurs and the system displays a message that a duplicate record exists in the file.

For indexed files, you add records randomly by key using chaining. **Chaining** means comparing the key field of the record to be added with the key fields already in the index. The reason for this comparison is to make sure that the record to be added is not a duplicate of a record already in the file. Chaining allows you to design your program so that, if a duplicate key field is found, your program can handle it appropriately without requiring the person using the display station to decide how to respond to an error message. The *Programming with RPG II* manual has more information about adding a record to an indexed file using chaining.

Using COBOL to Add Records to Delete-Capable Files: If you specify relative organization for the file, you can add records to delete-capable files. When ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, new records are inserted into the file. The RELATIVE KEY specified for the file must contain the desired relative record number for this record before a WRITE operation is performed. When the WRITE operation is performed, the record is placed at the specified relative-record-number position in the file. If the relative record number is not the number of a deleted record, a halt occurs and a file-status return code indicating that a duplicate record exists is returned to the program.

Using DFU with Delete-Capable Files

DFU can update, list, or inquire into delete-capable files. DFU cannot be used to create delete-capable files or to delete records from a delete-capable file.

Using WSU with Delete-Capable Files

WSU does not delete records from a transaction file in the same way as the other programming languages. Therefore, a WSU program ends abnormally if it tries to use a delete-capable transaction file. However, a WSU program can use a delete-capable **master** file.

Blocking Records and Index Entries

This section describes blocking of records and index entries, and the differences between physical and logical input/output operations.

Blocking Records

A record block is the number of records transferred as a unit of information between a disk file and a buffer in main storage. Although only one record at a time is available for processing by your program, one or more records may be transferred into the data buffer at a time. The block length specifies the amount of main storage used for a data buffer in your program.

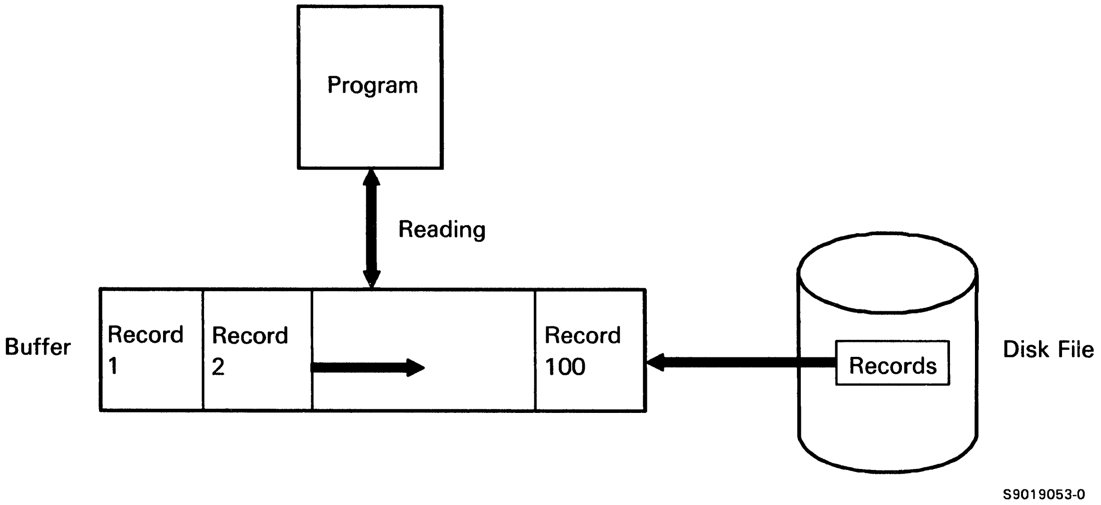
The size of the buffer is determined by the block length specified in your program. You can change the buffer size by using the DBLOCK parameter of the FILE OCL statement. The block length does not affect the way that records are stored on disk.

Considerations for Efficient Record Blocking

Block length is a multiple of record length. For example, if the record length is 64 characters and the blocking factor is four, the block length is 256 characters. In that case, four records are transferred at one time.

The system always transfers data between disk and main storage in sectors (256 bytes). Because of this, the system may round your block size up to a multiple of 256. For efficient blocking, you should choose a record length that is either a multiple or submultiple of 256. For example, 512 is a multiple of 256, and 64 is a submultiple of 256 because it divides into 256 an even number of times. This choice is best because data is always transferred in sectors, and for records with a record length less than or equal to 256, you eliminate the chance of having records stored in more than one sector. The system attempts to get the storage space as requested. However, it may allocate less space, depending on the amount of main storage available.

Blocking is useful if you are likely to process multiple records in a data buffer. By specifying a large blocking factor, you reduce the number of times the system must read from and write to disk. For example, assume that your program reads a file consecutively when records are blocked 100 per data buffer. To read the first record, the system must transfer 100 records from disk to the data buffer, which takes a relatively long time.



However, the next 99 times the program reads a record, the record is already in main storage, so no time is required to read the disk file.

The system takes advantage of a large block size if you are sharing files. However, the records may be reread if other programs sharing the file change records that are in the buffer. For example, if a program reads 100 records into the buffer and then another program updated the 50th record in the block, the system would have to read the file again to get the current version of the 50th record.

By increasing the size of your data buffers, the number of disk reads and writes required by your program is reduced. However, increasing the size of your data buffers increases the amount of main storage required to run your programs. This can affect the performance of both your program and system.

For files processed randomly, you should not specify a large block length unless you are sure that more than one record will be processed in a block before another block is transferred.

If the record length is less than or equal to 256 and blocking is not specified for the file, the data buffer holds either a single sector of data if the record length is a submultiple of 256, or two sectors of data if the record length is not a multiple or submultiple of 256. If the record length is greater than 256 and blocking is not specified for the file, the data buffer size is the record length rounded up to the next multiple of 256.

To take advantage of the disk cache, it is better not to specify a blocking factor.

Blocking Index Entries

When a program processes an indexed file, the index entries for the file are read into a buffer area of main storage. The number of index entries placed in this buffer is determined by the program and can be changed by the IBLOCK parameter on the FILE OCL statement.

The IBLOCK parameter tells the system how many index entries to read from the disk to main storage on each disk read. Blocking index entries saves processing time if consecutive index entries are used, because the system does not have to read the disk file again until all the entries in the buffer are processed. When the program is finished processing the index entries in the buffer area, more index entries are read from disk to main storage.

The size of the index buffer area that is specified by the IBLOCK parameter is based on the length of the key. The number of index entries that will fit into one sector equals 256 divided by the sum of the key length plus 3. This number is rounded down to the next lower whole number. The number of sectors in the index buffer equals the IBLOCK value specified on the FILE OCL statement, divided by the number of index entries per sector. This value is then rounded up to the next highest whole number. The system attempts to get the storage space as requested. However, it may allocate less space, depending on the amount of main storage space available.

For example, suppose a file contains records that have a 9-digit key, and 100 is specified as the IBLOCK parameter on the FILE OCL statement. The following calculations show how the system determines the number of sectors in the index buffer for this example:

$256 / (9+3) = 21$ index entries per sector

$100 / 21 = 5$ sectors in the index buffer

When using the IBLOCK parameter, you should be aware that increasing the size of your index buffers can reduce the number of disk reads required by your program, but it also increases the amount of main storage required to run your program.

Considerations When Specifying the IBLOCK Parameter

- If you are accessing an indexed file sequentially by key, and the file records are not organized to match the keys, specify DBLOCK = 1 and specify a larger value for IBLOCK.

- The benefit of a large IBLOCK can be realized by issuing a sequential read first. If you specify a large IBLOCK and you are doing only random reads by key, the SSP will only read one sector of the index into the large index buffer. This is due to a disk hardware scan performed rather than a hardware read.

- For better system performance, be sure that the program processing size (the program code plus the index buffer plus the data buffer for all the files used by the program) is less than 64K bytes. If the processing size is greater than 64K bytes, the system puts the buffer in the task work space, requiring more time to access the files. For more information, see "Program Processing Size" later in this chapter.

- If a program uses several files, you may want to specify the largest IBLOCK size for the file that is accessed the most.

- If you specify the IBLOCK parameter for a file that is not an indexed file, the system ignores the IBLOCK parameter.

- If you do not specify the IBLOCK parameter for an indexed file, the system uses the blocking factor that was specified in the program. High-level language programs set the index blocking factor equal to the blocking factor for the data buffer.

Using Record Blocking and Index Blocking

On System/36, buffers are not included in your program when it is compiled. Instead, the system acquires the buffers when your program opens files.

This approach has several advantages:

- Programs (logic) can be larger. On System/36, a program can be a maximum of 64K bytes; this size does not include space required for the buffers.

- Buffer sizes can be changed without recompiling the program. The DBLOCK and IBLOCK parameters of the FILE OCL statement are provided for this purpose.

- Less library space is required to store your program.

- Storage for buffers is allocated only for opened files. If your program can control opening and closing files, buffer space for closed files is reused.

Storage Space Required for a Program

To improve performance, you may want to vary the amount of storage required to run your program.

One way to determine the amount of storage space required to run your program is to use the STATUS control command while your program is running. The STATUS USERS control command shows the storage space used by the program.

Another way to determine the storage space required for your program involves two steps:

1. Calculate the amount of storage required for the file space used by your program. **File space** means the space required for record blocking, index blocking, and control blocks.

2. Add the file space amounts to the size of your compiled program. The sum is the total amount of storage required to run the program.

Calculating Storage Space Required for Disk Buffers and Control Blocks

In general, the amount of storage space required for buffers is based on the following:

- Number of files being used.

- Length of the records in each file.

- Record blocking factor specified for each file.

- Index blocking factor for each file.

- Number of alternative indexes for each file.

- Key length.

- Whether batch add processing is done (see "Delaying Maintenance for Batch Adds" on page 8-72).

In addition to buffer space, 200 bytes are required for internal control blocks. The following formula shows how to determine the amount of main storage required for buffers and control blocks for each file opened by your program.

```
Main storage space  = 200 + data buffer size +
(in bytes)
                      index buffer size + (6 x key length)
```

Note: The `(6 x key length)` *expression is required only if the file is processed by key.*

The **data buffer size** is calculated from the record length and DBLOCK value as follows:

1. Multiply the DBLOCK value by the record length to determine the record area.

   ```
   (DBLOCK value) x (record length) = (record area)
   ```

2. Divide the record area by 256 to determine how many sectors are required by the buffer.

   ```
   (record area) / 256 = (number with remainder)
   ```

3. If the remainder is 0, the data buffer size equals the record area.

   If the remainder is a submultiple of 256, the data buffer size equals the record area plus 255, rounded down to the next multiple of 256.

   Submultiples of 256 are: 1, 2, 4, 8, 16, 32, 64, 128, and 256.

   ```
   (record area) + 255 = (data buffer)
   ```

   Round the data buffer size down to the next multiple of 256.

   If the remainder is not a submultiple of 256, the data buffer size equals the record area plus 510, rounded down to the next multiple of 256.

   ```
   (record area) + 510 = (data buffer)
   ```

   Round the data buffer size down to the next multiple of 256.

The **index buffer size** is calculated from the key length and IBLOCK value as follows:

1. Divide 256 by the key length plus 3 to determine the number of index entries per sector.

   ```
   256 / (key length + 3) = (index entries per sector)
   ```

   Round the result down to the next whole number.

2. Divide the IBLOCK value by the number of index entries per sector to determine the number of sectors in the index buffer.

   ```
   (IBLOCK value) / (index entries per sector)
                  = (sectors in index buffer)
   ```

   Round the result up to the next whole number.

3. Multiply the number of sectors value times 256 to determine the index buffer size.

   ```
   (sectors in index buffer) x 256 = index buffer size
   ```

The main storage space can be made larger if:

- Batch add processing is done (see "Delaying Maintenance for Batch Adds" on page 8-72).

- Alternative indexes are defined for the file.

- The file is an indexed file not processed by key.

For batch-add processing of an indexed file, an index buffer and a 24-byte control block are allocated for each index that is not the index on the access path. For example, if nonkeyed batch-adds are done to an indexed file that has one alternative index, two additional index buffers and two 24-byte control blocks are allocated. (One index buffer and one control block are for the primary index because it is not used to access the file; the second index buffer and control block are for the alternative index.) The batch-add index buffers are the same size as the index buffers on the access path. If no index buffer exists on the path, the length of the index buffer is 256 bytes.

The maximum amount of storage space that the system allows for each file is 44K bytes. If more storage is requested, the system automatically eliminates or reduces the size of some of the buffers so that the total storage required for the file is 44K bytes or less.

Allocating Storage for Buffers and Control Blocks

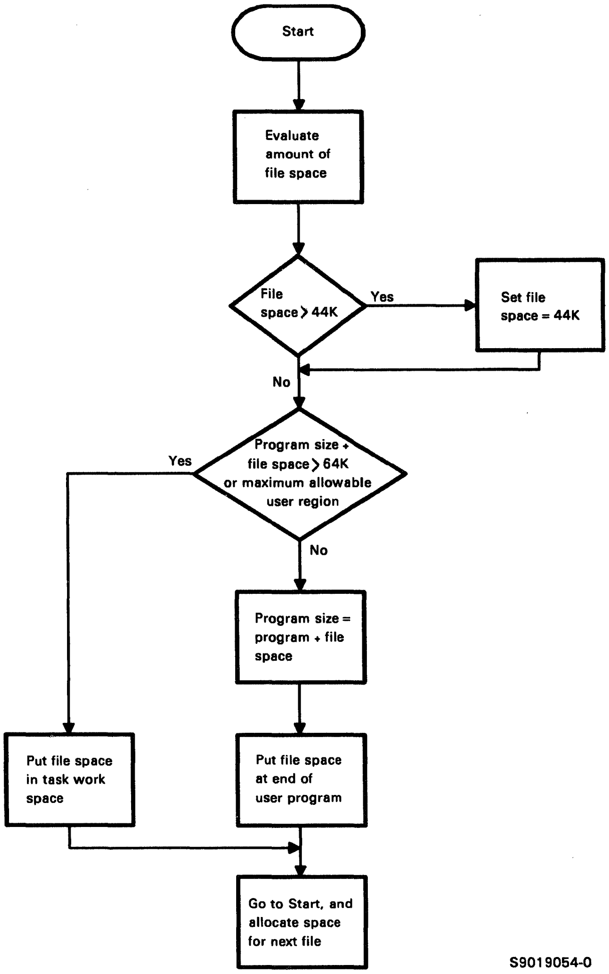
The storage space required for each file is allocated to the program when the file is opened. This required storage space called **file space**, includes data blocking, index blocking, and control blocks.

The system allocates file space to a program in one of two ways:

- Adding the file space to the user program area and increasing the program's processing size by the amount of the file space. This occurs if adding the file space to the user program area does not make the program's processing size greater than 64K bytes or greater than the available user storage. This is called **appending** the file space to the user program.

- Placing the file space in the task work space, which is an area separate from the region size of the program. This occurs if adding the file space to the user program area makes the program's processing size greater than 64K bytes or greater than the available user storage.

The system uses the following logic when allocating file space:


Start
Evaluate amount of file space
File space > 44K
Yes
Set file space = 44K
No
Program size + file space > 64K or maximum allowable user region
Yes
No
Program size = program + file space
Put file space in task work space
Put file space at end of user program
Go to Start, and allocate space for next file
S9019054-0

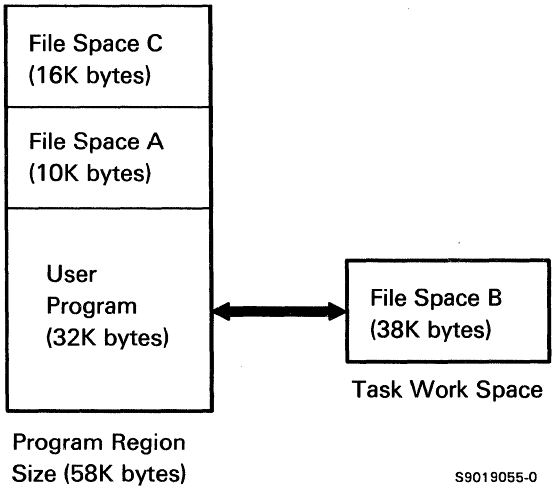
Program Processing Size

The program processing size is defined by the program code (load member) plus appended file spaces. The actual method of file space allocation is determined separately for each file. The optimum situation occurs when the program plus all file spaces are less than 64K bytes.

If the program's processing size is greater than 64K bytes, the system assigns the file space that caused the size to exceed 64K bytes to the task work space. Assigning the file space to the task work space may decrease system performance because of the time needed to address the task work space (which is outside the program region size), and to swap the task work space into and out of main storage separately from the program.

The following diagram shows what happens when a program opens three files: file A, file B, then file C. Note that even though file B was opened before file C, the buffer space for file B was put in the task work space. File B could not be appended to the program because its buffer space (38K bytes) would have caused the total program size to exceed 64K bytes:

```
32K bytes (program) + 10K bytes (file A)
+ 38K bytes (file B) = 80K bytes
```



It is important to understand that data buffers are allocated to the program as files are opened. Therefore, data buffers most commonly accessed by the program should always be opened first, so that they are allocated to the program's region instead of being placed into the task work space.

RPG II opens all files at program initiation. The primary file is opened first, and then the other files are opened in the sequence in which they are coded on the file description specifications.

A COBOL program can define the sequence for opening files by using the OPEN statement. It may be advantageous to open a file only when required and to close the files that are no longer in use, rather than opening all files at program initiation and closing them at program termination. This approach allows the system to reallocate the buffer space used by closed files.

Physical and Logical Input/Output Operations

Physical input/output operations perform disk read and write operations. These operations take time because they usually require positioning and moving the disk arm. Therefore, during application design, you should plan to minimize physical input/output operations in order to improve response time and system performance.

Logical input/output operations access records. The number of physical input/output operations that result from logical input/output operations is affected by the following factors:

- Record blocking
- Access method
- Deferring file operations
- File sharing

Record Blocking

For information about how record blocking is related to physical and logical read operations, see "Blocking Records" on page 8-60.

Access Method

When an indexed file is accessed, each logical input/output operation results in two physical read operations for each record. The index entry and the data record are read in two separate operations.

Direct file organization allows faster random access to records in a file than any other file organization. However, there the program must compute the relative record number location of the record within the file.
Appendix A, "Access Algorithms for Direct Files" describes several access algorithms.

When records need to be processed consecutively, a program accessing the records can process faster if the records are ordered sequentially in the file. For example, assume that a file has 50 records per block and that consecutive processing is used. A physical input/output may be required only once for each set of 50 logical requests.

Delayed File Operations

In certain cases, the system delays the following file operations:

- Reading records from a file
- Writing records to a file
- Maintaining indexes

Delayed Input Operations

For input operations, the system reads a disk file when the required record is not in the buffer or when the buffer contains records that no longer reflect the current file status. This ensures that any record retrieved reflects the latest information in the file.

If an indexed file is used, the system reads the index from disk under the same conditions.

The amount of data read is equal to the buffer sizes specified for either the data or the index entries.

Delayed Output Operations

For output operations, the record is written to the file on each output operation unless the system determines that the output operation can be delayed until the buffer is full. File output is delayed if the program accessing the file meets the following conditions:

- The file is not shared.
- Multiple logical files are not defined in the program.

Output of index entries may be delayed under the same conditions listed above.

When the output is delayed, the amount of data written is equal to the size of the buffer specified. The buffer sizes depend on the blocking factors specified in the program or on the FILE OCL statement.

When the system does not delay output, only the sector(s) in the buffer containing changed or added records is written to disk.

Delaying file output can improve performance. However, it may result in lost data if a program ends abnormally. The maximum amount of data that can be lost depends on the size of the data buffer specified in the program or on the DBLOCK parameter of the FILE OCL statement. Also, because of the structure of the buffer, data records may be in more than one sector. Therefore, after an abnormal termination, the last record in the file buffer may be only a partial record.

Delaying Maintenance of Indexes

Normally, the overflow portion of an index is arranged in ascending key sequence. Maintaining this sequence requires that entries added to the index be inserted into the overflow area of the index in sequence. However, at certain times, the system delays arranging the index in sequence. Instead, it places the new index entry at the end of the overflow area of the index, and later it sorts the overflow portion. The maintenance of indexes is delayed in the following two cases:

- Unused alternative indexes
- Batch add processing

Delaying Maintenance for Unused Alternative Indexes

Maintenance of indexes is delayed for all alternative indexes to a file that are not accessed by a program. When an alternative index is being used, the system always maintains the index. The maintenance of the index is delayed until the alternative index file is opened. When it is opened, the overflow portion of the index is sorted. From that time on, the index is maintained while the file is used. The maintenance of the primary index is not delayed, even if no programs are using the file.

Delaying Maintenance for Batch Adds

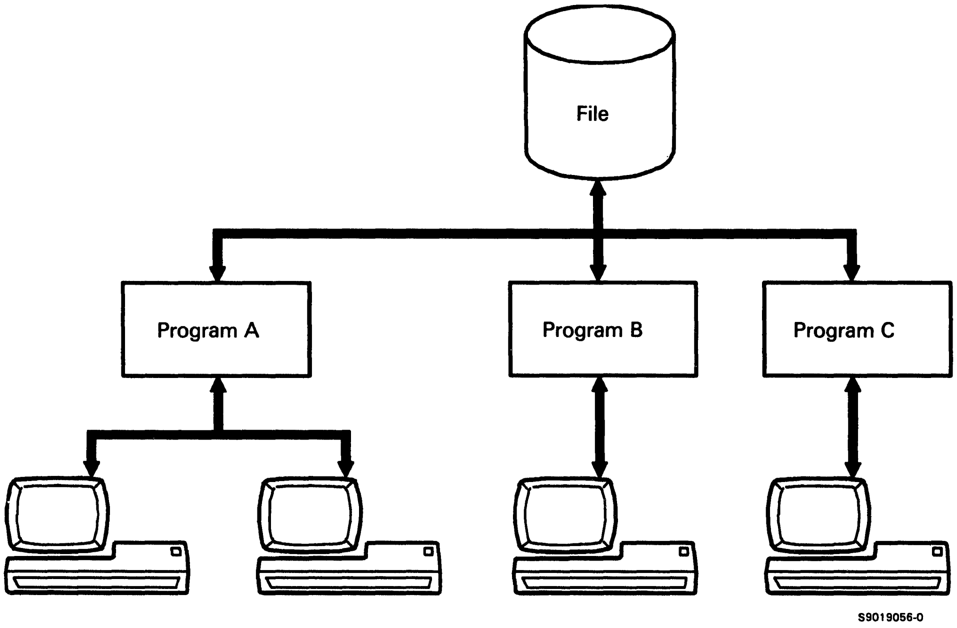
Maintenance is delayed if the file access meets the following conditions:

- The file is not shared
- No other logical files have been specified for the file
- Only add operations are done to the file (or, if other operations are being done, the add operations must be sequential)

When the system adds records in batches, the maintenance of the index is delayed until the job step ends. At that time, the overflow portion of the index is sorted if the entries are not in sequence. The maintenance of all other indexes is deferred until the particular file is opened as described in the previous topic.

Sharing Files

File sharing means allowing two or more programs to access the same file at the same time.



File Sharing Considerations

If you want to allow more than one program to share a file, consider that:

- Only resident files can be shared.
- Files that are being created cannot be shared.
- If you change a resident file to a scratch file by specifying a RETAIN-S parameter on the FILE OCL statement, the file cannot be shared.
- The system protects records read for update by one program from being changed by another program using the same file. For more information, see "Record Protection" on page 8-78.
- If programs share more than one file, all programs should access the files in the same sequence to reduce the chances of a file deadlock occurring. For more information, see "File Deadlock Conditions" on page 8-84.

Levels of File Sharing

The system allows several levels of file sharing. The level of file sharing is determined by the DISP parameter of the FILE OCL statement, except in BASIC programs, where the share level is specified on the OPEN statement.

When one or more programs are using a file, the programs that own the file determine which other programs can share the file. The share level tells the system how the program uses the file and what types of processing other programs can do while sharing the file. For example, if DISP-SHRMR is specified, the program can modify the file while sharing it with other programs that can only read the file. Once the system lets the program use the file, other programs that want to modify the file are not allowed to use the file.

The following table shows:

- The levels of file sharing you can specify
- The type of processing you can do when you own the file
- The type of processing other programs can do while your program owns the file

Share Level	The Program That Owns the File Can	Other Programs Can
SHR	Read and modify	Read and modify
SHRMM	Read and modify	Read and modify
SHRMR	Read and modify	Read only
SHRRM	Read only	Read and modify
SHRRR	Read only	Read only
OLD	Read and modify	Not allowed to access file
NEW	Read and modify	Not allowed to access file
Not specified	Read and modify	Not allowed to access file

Note: Modifying includes update, delete, and add operations to a file.

Waiting for Files to Become Available

If a file is used by another program and the file is at a share level that does not permit your program to use the file, the system either displays a message or waits for the file.

A message is displayed if the file is owned by a never-ending program with a share level of noshare with the requesting program or if the file was acquired by a FILE OCL statement with JOB-YES specified. The message allows you to either cancel your job or have the system try again to get ownership of the file.

If the file is not available and is not used by a never-ending program, the system automatically waits for the file until the program using the file goes to the end of the job step. While the program is waiting, all other programs that request the file also have to wait. When the file becomes available, the program gains ownership of the file and begins running. If other programs are also waiting to use the file, the system checks if they can use the file at their requesting share level.

The following table shows whether another program can share a file at a requested level when one program owns the file:

Share Level Requested by Another	**Share Level for the Program That Owns the File**				
Program	**SHRRM**	**SHRMM[1]**	**SHRMR**	**SHRRR**	**No share[2]**
SHRRM	Yes	Yes	Yes	Yes	No
SHRMM[1]	Yes	Yes	No	No	No
SHRMR	Yes	No	No	No	No
SHRRR	Yes	No	No	Yes	No
No share[2]	No	No	No	No	No

[1] DISP-SHRMM is the same as DISP-SHR.

[2] No share is specified by DISP-OLD, DISP-NEW, or by no DISP parameter being specified.

For example, if your program requests to share a file by using DISP-SHRMM on the FILE OCL statement, the system would allow your program to share the file only if the programs that own the file specified either DISP-SHRRM, DISP-SHRMM, or DISP-SHR on the FILE OCL statement.

When several programs are sharing a file, other programs are allowed to share the file only when their requesting share levels are compatible with all other share levels.

Using the WAIT Parameter

Another way to determine (within your jobs) whether a file is available is to use the WAIT parameter on a FILE OCL statement that is not between LOAD and RUN statements. You can specify WAIT-YES or WAIT-NO.

If WAIT-NO is specified, the system tries to acquire the file for the program at the desired shared level. If the file is unavailable to the program, completion code 2030 or 2031 is returned to the procedure. By using the ?CD? substitution expression and an IF conditional expression within your procedure, you can choose the processing steps done within a job if the file is unavailable for you by your program. For example, if a file you need as input to the first program of a job is unavailable, you may decide not to run the rest of the job.

If WAIT-YES or if no WAIT parameter is specified, the program waits until the file becomes available. This wait condition lasts until the program can use the file. For example, you may decide to submit a program using a particular file and not have this program use the file until all other programs using the file are finished.

If a file is owned by a suspended program or by a never-ending program, the system ignores the WAIT-YES parameter. The system issues a halt message to the display station operator and stops processing the program.

The following example uses the WAIT parameter to check whether a file is unavailable (busy) for more than 30 minutes. If the file is unavailable for more than 30 minutes, the system sends the system operator a message to run the job later.

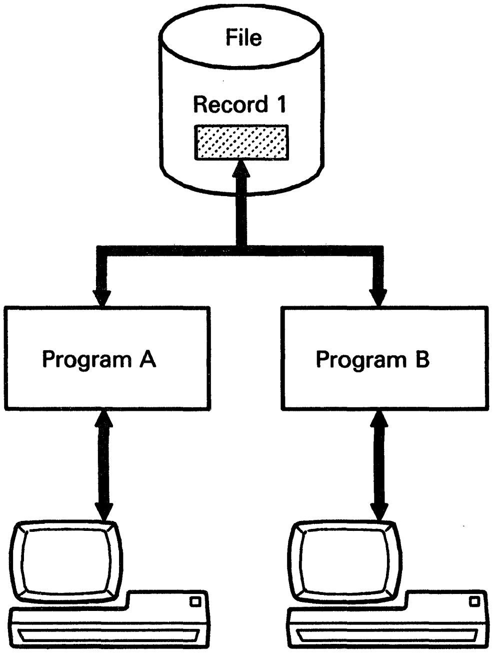
```
* Parameters used:
* Parameter 1  = Number of minutes between tries.
*                Range of 1 to 59 minutes.
*                Default 5 minutes.
* Parameter 2  = Maximum number of tries.
*                Default is 6 tries.
* Parameter 64 = Number of times the program tried
*                to get the file
*********************************************************
// IFF ?1?=' ' EVALUATE P1='?1?00'
// EVALUATE ?64F'1'?  P1,6=?1'000500'?
// TAG LOOP
// FILE NAME-TEST,WAIT-NO
// IF ?CD?=0000 GOTO GOTFILE
// IF ?64?>?2'6'? GOTO NOFILE
// EVALUATE P64=?64?+1
// WAIT INTERVAL-?1?
// GOTO LOOP
*
// TAG NOFILE
// ** 'FILE "TEST" IS BEING USED; RUN JOB LATER'
// RETURN
*
// TAG GOTFILE
// LOAD PROG1
// RUN
```

Note: *If you place the FILE statement before the LOAD statement, the system attempts to acquire the file for the job immediately. For example:*

```
// FILE NAME-INPUT,UNIT-F1,LABEL-MASTER,RETAIN-T,
//      DISP-OLD,WAIT-YES
// ATTR CANCEL-NO,MRTMAX-20,NEP-NO,PRIORITY-HIGH,
//      RELEASE-YES
// LOAD ORDPRG,ORDERLIB
// PRINTER NAME-REPORT,ALIGN-YES,SPOOL-YES
// RUN
```

Record Protection

Record protection is the means of preventing two or more programs from updating a record in a shared file at the same time. Record protection applies to programs that access the file with SHR or SHRMM share levels.



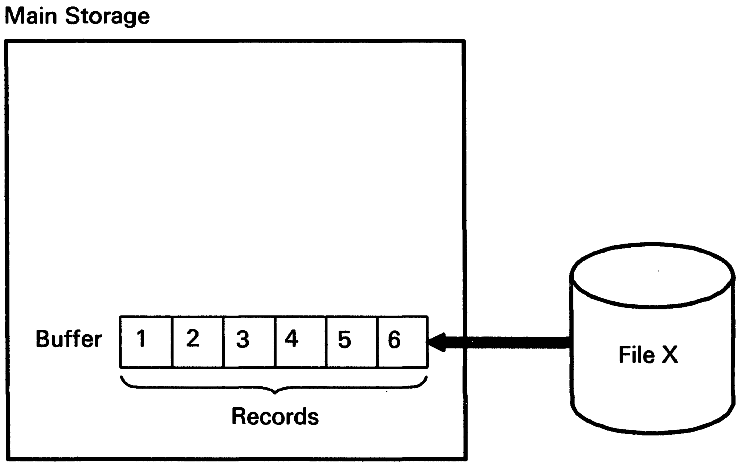
S9019057-0

To understand how a record is protected, you should first understand how a record is updated when only one program is involved and then what happens when more than one program is involved.

When there is only one program, updating a record consists of the following steps:

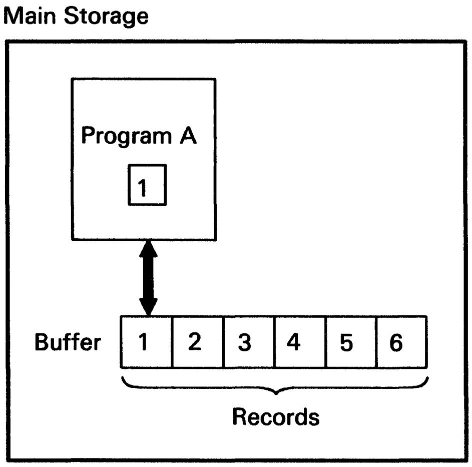
1. The program requests data management to read a record for update.

2. Disk data management places one or more sectors containing the record into a buffer area in main storage. The number of sectors placed into the buffer depends on the blocking factor specified either in your program or by the DBLOCK parameter of the FILE OCL statement. For example, if program A is to update record 1 in file X, the buffer assigned to the program might look like this:



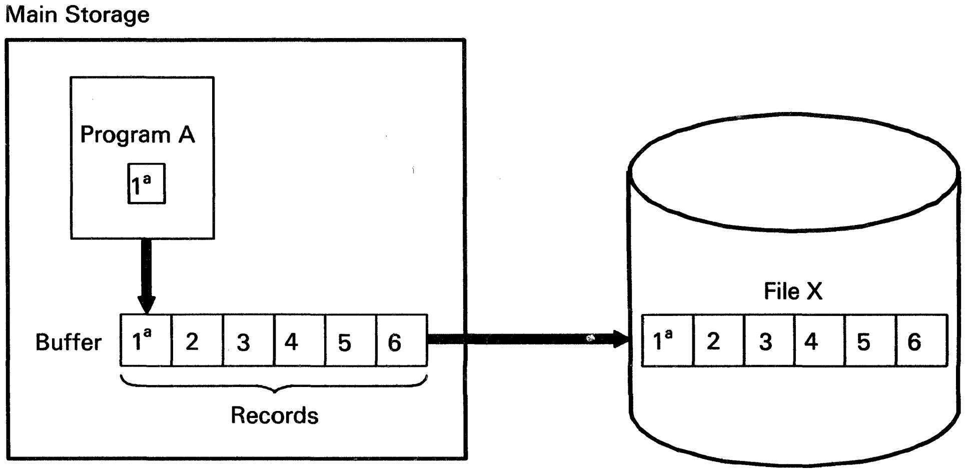
S9019058-0

3. Disk data management retrieves record 1 from the buffer, locks it so that other programs do not have access to the record, and gives the record to program A to update:



S9019059-0

4. The program updates the record and calls data management to write the updated record back to the file.



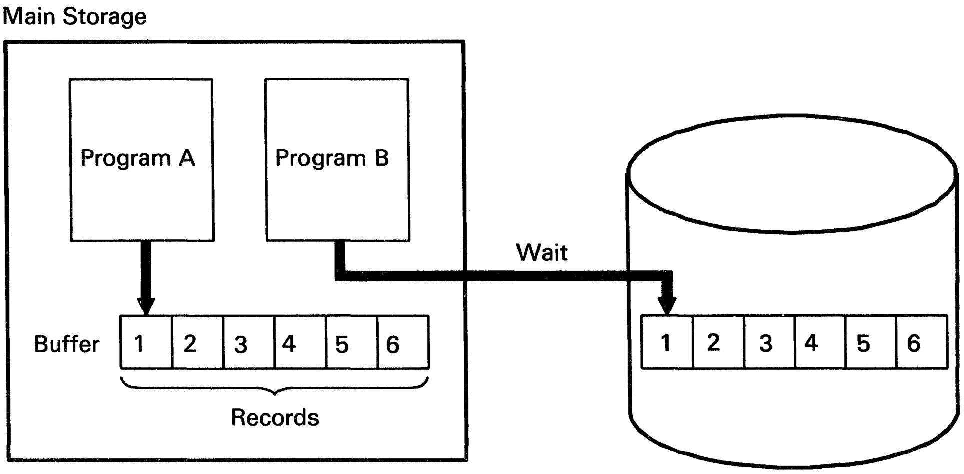
[a] Record 1 locked.

S9019060-1

5. Disk data management moves the updated record into the buffer and writes the sector(s) containing the record back to the file.

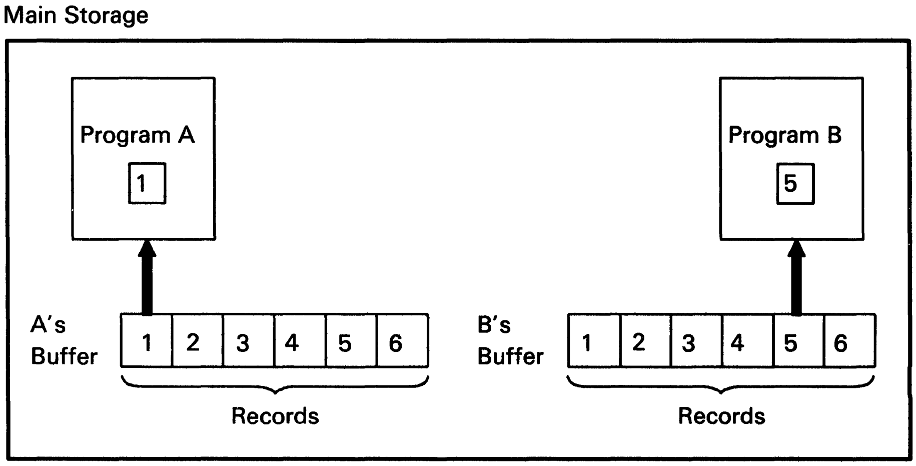
6. Disk data management unlocks the record so other programs can access the record for possible updating.

The following section describes how a record is updated when there is more than one program. Suppose that program B wants to update record 1 in file X. Because program A is using that record, it is not available to program B. Program B must wait until the record is released.



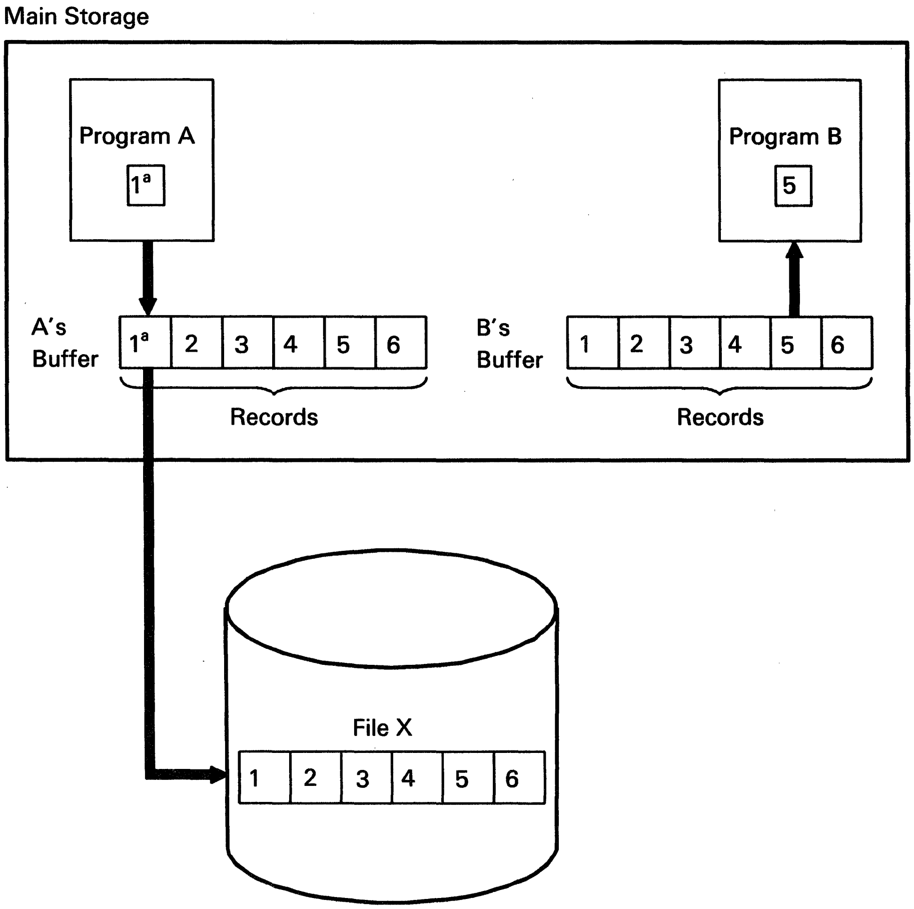
S9019061-0

Now suppose that program B wants to update record 5, which is in the same sector(s) as record 1. The system places the sector(s) containing records 1 and 5 in a buffer, locks record 5, and gives record 5 to program B.



S9019062-0

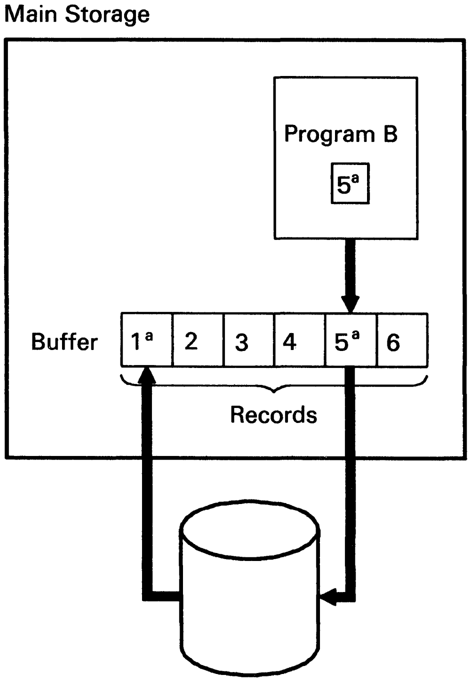
If program A finishes updating record 1 before program B finishes updating record 5, then when disk data management writes the sector(s) containing records 1 and 5 back to disk, it notes that program B's buffer does not contain the updated copy of record 1.



[a]Record 1 locked.

S9019063-1

When program B finishes updating record 5 and calls data management to write the record, the sector(s) it has in its buffer no longer contains the data that is in the file, because the buffer does not contain the updated record 1. Therefore, if data management wrote the sector(s) from program B's buffer back to the file, the changes made to record 1 by program A would be lost. To ensure that the changes made to record 1 are not lost, data management again reads the sector(s) containing records 1 and 5 from the file to a buffer in main storage. The updated record 5 is then moved into the buffer, and the sector(s) containing both updated records is written back to the file.

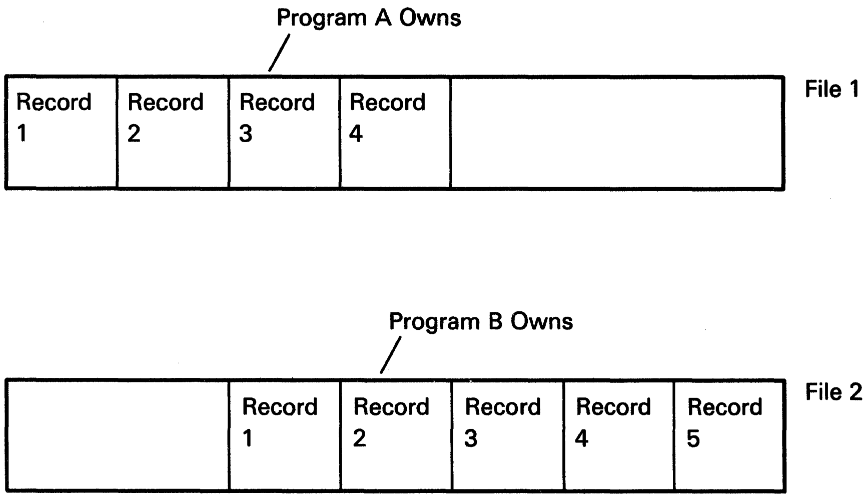


[a]Records 1 and 5 locked.

S9019064-1

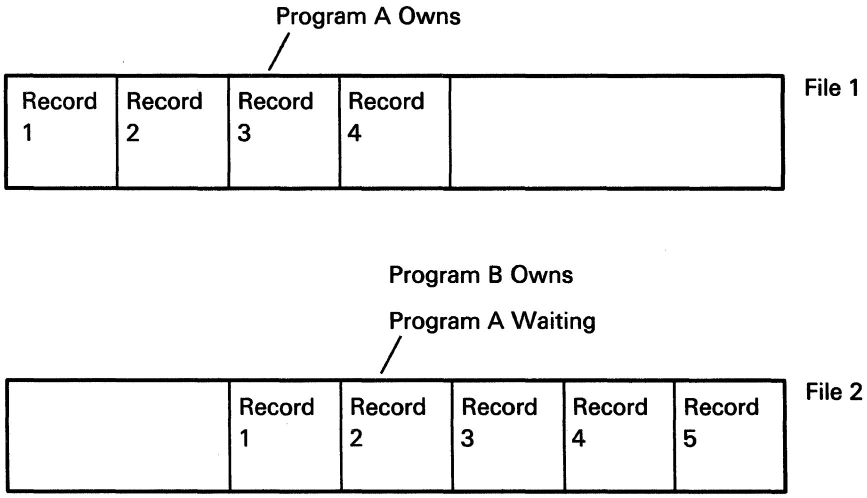
File Deadlock Conditions

A file deadlock condition can occur when two or more update files are shared. For example, assume that program A and program B are updating two shared files, file 1 and file 2. Program A reads record 3 for updating from file 1, and program B reads record 2 for updating from file 2.



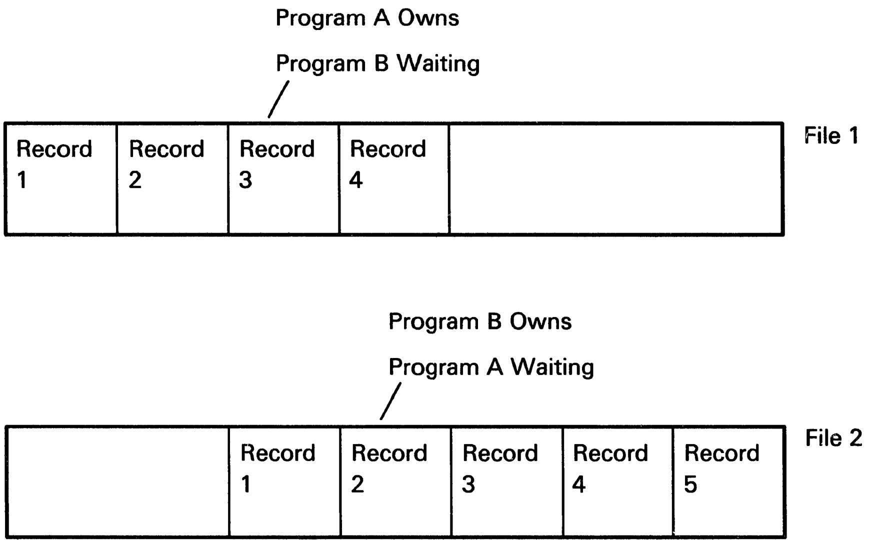
S9019065-0

Suppose that program A tries to read record 2 from file 2. Program A must wait because program B is using the record.



S9019066-0

Then, if program B tries to read record 3 from file 1, program B must wait because program A is using that record.



S9019067-0

This condition of programs waiting for each other is called a **file deadlock.** To ensure that file deadlocks do not occur, you should always release a record before reading a record from another shared update file.

If you press the Attn key to interrupt a program when a record lock exists on one of the files opened by the program, option 1 (request Command display) of the Inquiry Options display will have an asterisk (*) before the 1. This means option 1 can be chosen but will be delayed until all record locks are released. This prevents another job from being started at the display station while the suspended program is holding a locked record.

You must release the locked records before the Command display will appear. See "Releasing Locked Records" later in this chapter.

Releasing Locked Records

A record is released when any of the following conditions occurs:

- The program reads another record from the file.
- The program does a data management operation that causes an error to occur (see the note that follows).
- The program updates the record (see the note that follows).
- The program deletes the record from the file.
- The program adds a new record to the file (see the note that follows).
- The program does a release operation. How your program releases records depends on the high-level language you are using. For example, in RPG II, you can release a locked record by writing a record with no output fields.
- The program closes the file.
- The program ends.

Note: *In COBOL, these operations cause the record to be unlocked. However, if the program is using random access, updates to the record can be done. Because the record is unlocked, other programs can update the record between the time the record is unlocked and the time the COBOL program does its update. In this case, the other program's updates will be overlaid by the update from the COBOL program.*

File Update Programming Considerations

Possible Errors

If a single, logical file is processed by two or more display stations within the same program and if the program reads a record for updating but then reads a second record from the same file before updating the first record, the following errors can occur:

- An update or part of an update can be lost. For example, suppose a record is read from file X and displayed at display station 1. Then suppose the same record is read from file X and displayed at display station 2. If file X is not shared and the program does not reread the record, then the last update might overlay any previous updates. If file X is shared, an error message is displayed, and the second update is not performed.

- The wrong record can be updated. For example, suppose a record is read from file X and displayed at display station 1. Then suppose a different record is read from file X and displayed at display station 2. If display station 1 tries to update the first record but does not read that record again, disk data management tries to update the last record read from file X, which is displayed at display station 2. If this condition occurs during an attempt to update an indexed file, an error message might be displayed if the primary key field is changed, and the requested update is not performed. Otherwise, the wrong record is updated.

- An update performed by another program sharing the file can be lost. For example, suppose a record is read by program A from file X and is displayed at display station 1. Then suppose another record is read by program A from file X and is displayed at display station 2. The second read operation from file X causes the SSP to free the first record. Therefore, a second program sharing file X can update the first record. Then, if display station 1 reads the record again and updates the record by using the original field values, the updates made by the second program might be lost.

Note: For the above errors, the program can be an SRT with acquired work stations, or an MRT. The important thing to note is that when the program supports more than one work station at the same time, these errors can occur.

Avoiding These Errors

You can avoid the preceding error conditions by using one of the following techniques:

- Before doing an update, the program should read the record again and check that none of the fields being updated have been changed since the record was displayed for updating. If any of the fields were changed, the program should display the field again for updating or, if possible, use the field values currently in the record to do the update.

- Protect records being updated by establishing a field in the record to be used as a busy indicator that indicates the record is being updated. For example, a busy indicator might be the display station ID and the program name. Subsequent attempts to access the same record should test for the busy indicator and, depending on the value of the indicator, not allow the record to be updated. The busy indicator should be removed from the record when the update is performed by the requesting program or if no update is performed. If records in a file can be updated at the same time by two different programs, both programs should test and use the same busy indicator.

  If the program ends abnormally and you are not going to restart the program, you should run another program that turns off the busy indicators in records that were being updated by the program when it ended, so that programs that check the busy indicator can handle the record properly.

- Consider defining a separate logical file for each display station. Separate logical files protect against updates by other programs, but they do not protect against multiple updates within a single program. For more information, see the note under "Using One File As Two Or More Logical Files" on page 8-89.

Using One File As Two Or More Logical Files

Each file defined within a program is called a **logical file**. A program can use one disk file as two or more logical files. For example, a program can be written to access two files called FILEA and FILEB, which are the same physical file, by using the following OCL statements:

```
// FILE NAME-FILEA,LABEL-MASTER,DISP-SHRMM
// FILE NAME-FILEB,LABEL-MASTER,DISP-SHRMM
```

Defining a disk file as two or more logical files allows you to process one file by two separate access methods in one program. For example, one part of the program can access a master file randomly by key, and the other part of the program can access the same file randomly by relative record number.

Using the generalized access method is another way to process a file by two separate access methods (randomly and consecutively). For information about the generalized access method, see "Generalized Access Method" on page 8-41.

A single physical file is also used as multiple logical files in a program when more than one index is used by the program to process the file. For example, if FILEA has an alternative index labeled ALTINDXA, a program can use both files by using the following OCL statements:

```
// FILE NAME-FILEA,DISP-SHRMM
// FILE NAME-ALTINDXA,DISP-SHRMM
```

Records can be added to or updated in more than one of the logical files.

Note: *The system does not prevent a COBOL or RPG II program from updating the same record in two logical files at the same time. If a program updates the same record in two logical files at the same time, only the update that was made last appears in the physical file. However, if the file is shared, the system protects each record in the file from being updated at the same time by other programs (see "Record Protection" on page 8-78). Assembler users can specify LOCKCK-Y on the $DTFD macro to prevent the same record from being read for update through two logical files. BASIC prevents the same record from being read for update through two logical files.*

Chapter 9. Libraries

The purpose of this chapter is to:

- Describe what a library is.
- Suggest how you can use libraries for your applications.

Library Concepts and Uses

A **library** is a named area on disk that contains **library members**. These library members contain your programs, procedures, display formats, and message members that are used by your jobs.

Types of Libraries on the System

The system can contain the following types of libraries:

- The **system library (#LIBRARY)**. The system library contains most of the IBM-supplied programming support for the system. In most cases, you should create your own libraries to store other programming information, instead of using the system library. Any user information placed into the system library is erased when a new release is installed on the system.
- Other **program product libraries**. These libraries contain the IBM-supplied programming support for the programming languages, Development Support Utility, the Utilities Program Product, and the OFFICE/36 Program Products.
- **Your application libraries**. These are the libraries you create to store **your** programming information.

Library Naming Conventions

A library name can be up to 8 characters long and must begin with an alphabetic character (A through Z, #, $, or @). The remaining characters can be any combination of characters (numeric, alphabetic, and special) except #LIBRARY, F1, READER, DISK, PRINT, ALL, and TAPE.

You should avoid using the following characters because these have special meanings in procedures: commas (,), apostrophes ('), question marks (?), slashes (/), greater than signs (>), equal signs (=), plus (+), and hyphens (-).

You can create meaningful library names by abbreviating the name of the application that uses the library. For example:

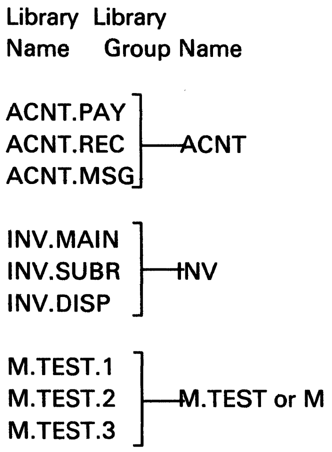
Library Name	Application
ACCTLIBR	Accounts receivable
PAYLIB	Payroll
PROGLIBR	Miscellaneous programs
MSGLIBR	Messages library

Group Libraries

Group libraries are a set of libraries collectively identified by a name that contains identifiers separated by one or more periods. The characters preceding a period identify the library group.

One advantage of using group libraries is that they can be secured easily. See the *System Security Guide* for more information about securing group libraries.

Here are some examples of library names that identify library groups:



S9019125-0

The limit of 8 characters for a library name also applies to names for library groups. The period counts as one of the 8 characters.

Libraries in a Group Resource

Folders, files, and libraries can be part of a group resource. The group resource must be named and each file, library, or folder that is part of the group must be named.

For example, a group resource could be PAY: a file that is part of this group could be PAY.FILE, a library could be PAY.LIB, and a folder could be PAY.FOLD. As you can see, the group resource name and the name of the file, library, or folder are separated by a period (.). PAY is called the group identifier.

One advantage of using groups is that they can be secured easily. See the *System Security Guide* for more information about securing groups.

The limit of 8 characters for a library name also applies to names of groups. A period counts as one of the 8 characters.

Types of Library Members

A library contains four types of library members:

- **Source members** (SOURCE or S members). Source members contain statements such as program statements or source specifications that are used as input to a compiler. Examples of source members are:
  - Source statements for programs
  - S-, H-, and D-specifications for display formats
  - Source statements for menus
  - Source statements for message members
- **Procedure members** (PROC or P members). Procedure members (also called **procedures**) contain the statements necessary to run a program or a group of programs.
- **Load members** (LOAD or O members). Load members contain information in a form that the system can use directly. Examples of load members are:
  - Compiled and link-edited programs
  - Compiled display formats
  - Compiled menus
  - Compiled message members
- **Subroutine members** (SUBR or R members). Subroutine members are usually members that have been compiled. BASIC programs are normally stored as subroutine members. WSU, DFU, and Query/36 also store members as subroutine members.

Library Member Subtypes

A **subtype** is a further classification of a library member. For example, a library source member might contain RPG II program statements, COBOL statements, BASIC statements. You specify the library member subtype for source and procedure members when you create or change the member. When you compile, link-edit, or generate a load member, the subtype is automatically assigned by the system based upon the subtype of the source member from which the load member is derived.

The system allows you to assign subtypes to your library members. These subtypes allow you to better define the library member. You can use them to help you identify the members when you list them. Also, some of the utilities allow you to select the subtypes you want to work with. For example, the screen design aid utility (SDA) allows you to select subtypes; thus, you can choose to view only display formats if you want.

Subtype	**Meaning**
ARP	RPG II auto report member
ARS	Automatic response member
ASM	Assembler member
BAP	BASIC procedure (source member)
BAS	BASIC member
BGC	Business graphic chart
BGD	Business graphics data
BGF	Business graphics format
COB	COBOL member
CSM	Alert source member
CSP	Cross-system product
DFU	Data file utility member
DLS	Document library services
DTA	Data member
FMT	Display format member
FOR	FORTRAN IV member
ICF	CNFIGICF procedure Interactive Communications Feature member
KEY	KEYS procedure
MNU	Menu member
MSG	Message member
PHL	Phone list member
QDE	Query data entry
QRY	Query/36 member
RPG	RPG II member
SRT	Sort member
SSP	CNFIGSSP procedure system configuration member
TXT	Text member
UNS	Member subtype not specified
WSU	Work station utility member
X25	CNFIGX25 procedure line configuration member

Library Member Naming Conventions

A library member name can be up to 8 characters long and must begin with an alphabetic character (A through Z, #, $, or @). The remaining characters can be any combination of characters (numeric, alphabetic, and special).

You should avoid using the following characters because these have special meanings in procedures: commas (,), apostrophes ('), question marks (?), slashes (/), greater than signs (>), equal signs (=), plus (+), period (.), and hyphens (-). Do not use DIR, SYSTEM, NEW, or ALL as a member name.

Uses of Libraries

Libraries allow you to group programs, display formats, and message members for a specific application. For example, you could place all your order entry jobs into a library named ORDERLIB.

Having several libraries allows you to group programs, displays, and message members in separate libraries. If you decide to use multiple libraries for your applications, you can group the members contained in those libraries in several ways:

- By application or job. For example, you could place the order entry application members into a library named ORDERLIB and the payroll application members into a library named PAYLIB.

- By user. You can assign each operator his own library for creating his own menus or procedures.

- By shifts. For example, you can assign the first shift a library which contains all the programs they can use, but restrict the second shift to a different library. The second shift library would contain only those programs that can be run without your supervision.

- By business cycle. You could create one library containing only programs that are run daily, a second library that contains programs to be run weekly, and a third library containing programs to be run monthly. In this way, you can save disk space by loading the second and third libraries only when you need them.

- By production and testing. As you develop your applications, you may find it easier if you have a development library (to contain the programs you are working on) and a production library (to contain the programs that have been developed and tested). When you finish testing the applications, you can move them into the production library. The members in each library are kept separate to ensure that no untested programs are run by mistake.

- By member type. For example, you could place the displays and messages in one library and the programs in another library.

Libraries can be secured. This allows you to specify the operators that can run the application programs and who can change the programs in the library.

Sign-On, Current, Session, and Job Library

Libraries can be assigned to display stations, operators, or jobs in the following ways:

- **Sign-on library**. You can have a particular library designated as the library to be used by a particular display station or operator. You use the SET procedure to define the library to be used by a particular display station; the specified library is then shown on the Sign-On display at that display station.

  Another way to have an automatic sign-on library is to have a security officer use the SECEDIT procedure and the user ID file to define a sign-on library for a particular operator. This library name will not appear on the Sign-On display, but the system will automatically search the user ID file for the sign-on library. See the *System Security Guide* for more information.

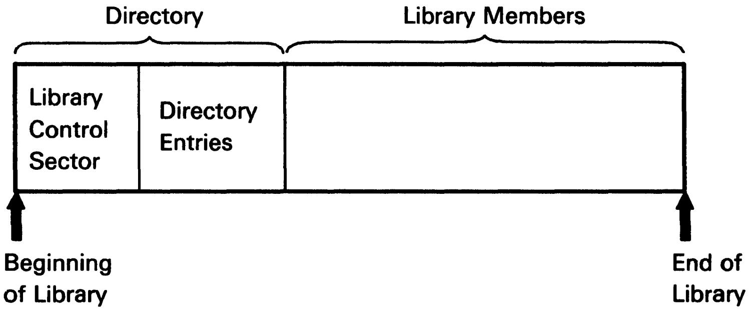
- **Current library**. Each display station that is signed on has a current library associated with it. Normally, the current library is the sign-on library, but you can specify either a session library or a job library as the current library.

  - **Session library**. A library can be designated for a session. The system first searches the session library for the members needed for a job. If the job information is not found in the session library, the system then searches the system library (#LIBRARY). You can use the SLIB procedure or the MENU control command to change the session library.

  - **Job library**. A library can be designated for each job or job step by using the LIBRARY OCL statement.

Library Format

A library has the following format:



S9019069-0

The **library directory** contains a library control sector and an entry for each member in the library. The library control sector is used to keep track of the library space that is being used or is available for use. Each time you create or remove a member, the library control sector is updated to indicate the change in space used and available.

The directory entries contain the following information for each member in the library:

- Name of the member.
- Type of the member; for example, a procedure or source member.
- Subtype of the member; for example, RPG (RPG II) or COB (COBOL).
- Date and time the member was created or last changed.
- Reference number of the member. The reference number for source or procedure members is usually increased by 1 each time the member is edited. The reference number for load or subroutine members is set to the reference number of the source member that was compiled or link-edited.
- Attributes of the member; for example:
  - Display format member
  - IBM-supplied member
  - Multiple requester program
  - Never-ending program
  - Whether a program temporary fix (PTF) was applied
- Release level of the system programs when the member was created or last changed.
- Maximum number of requesters that can be attached to a multiple requester terminal program.
- Amount of storage required for the program (for load members).
- Length of the statements in source or procedure members.
- Number of statements in source or procedure members.
- Size of the member (in sectors).

The listing produced by the LISTLIBR procedure provides this detailed information about libraries and library members. The *System Reference* manual has information about running the LISTLIBR procedure.

Library Size

When you create a library, you specify the total size for the library in blocks. The minimum size for a library is 2 blocks; the maximum is 15,000 blocks. Typically, a good starting size for a library is 100 blocks. You create a library by using the BLDLIBR procedure. The *System Reference* manual has information about running the BLDLIBR procedure.

You can specify a separate size for the directory; the directory size is specified in sectors (1 block equals 10 sectors). The minimum size for a directory is 2 sectors; the maximum is 2500 sectors.

To determine the number of available directory entries for a given number of sectors, multiply the number of sectors by 5 and subtract 7 from that number. For example, for 10 directory sectors, you could have 43 entries:

5 x 10 = 50
50 - 7 = 43

If you do not want to specify a directory size when you create the library, the system automatically assigns the size for you. The directory size will be 1/100 of the total library size. For example, if you specify a library to be 100 blocks, the system automatically assigns a directory size of 10 sectors.

1/100 x 100 blocks = 1 block

1 block = 10 sectors

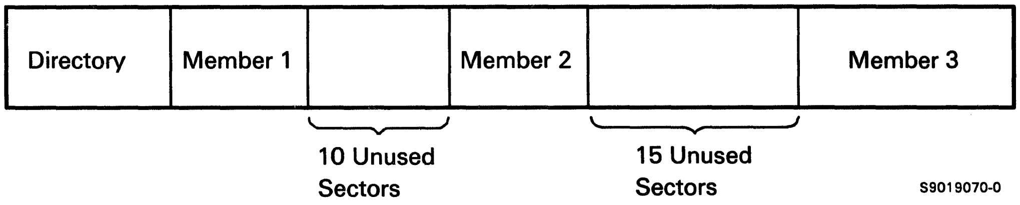
If you find that the library is either too small or too large for your use, you can use the ALOCLIBR procedure to increase or decrease the size. The *System Reference* manual has information about running the ALOCLIBR procedure.

Reorganizing Library Space

When you are developing applications, you will change, create, or remove members in your library. This can create unusable gaps in the library. Library space reorganization is the collecting of the unused space in a library in order to make a continuous unused area for the creation of new members.

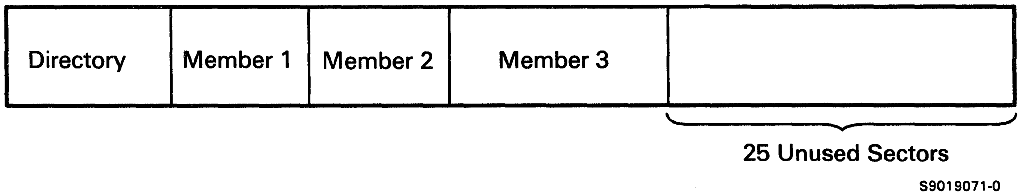
The CONDENSE procedure collects the unused space in a library into one area and, therefore, makes the space usable for new library members. For example, you want to create a member that takes 20 sectors of space and your library looks like the following:

Library Before CONDENSE



The library contains two areas of unused space: one is 10 sectors, the other is 15 sectors. Neither area is large enough to contain a 20-sector member. After you condense the library, it will look like this:

Library After CONDENSE



Now you have enough continuous space for the 20-sector member to fit in the library. The *System Reference* manual has more information about the CONDENSE procedure.

Library Extension

When you are copying a member to a library and the member is too large to fit in the library, the system automatically creates an additional area on disk into which this member can be placed. The additional area is called a **library extent**. The system also displays a message informing the user when an extent is about to be created.

A library can have only one extent. If the extent becomes full and you attempt to copy a member into the library, the system displays a message, and you must reallocate your library and then copy the member into the library.

This extent is normally 50 blocks (provided 50 blocks of disk space are available). The size of the extent may also depend on the following:

- If the library member being copied is larger than 50 blocks, the size of the extent will be the same size as the member.
- If less than 50 blocks of disk space are available, the system attempts to provide as much disk space as possible to contain the member.
- If not enough space is available to create an extent, the system displays a message that indicates you must reallocate or condense your library. You use the ALOCLIBR or CONDENSE procedure to reallocate or condense a library. The *System Reference* manual has information about how to run the ALOCLIBR and CONDENSE procedures.

Note: The system library (#LIBRARY) can be extended in sector mode, only. For more information see "Library Sector-Mode and Record-Mode Files" on page 9-15.

Library Sharing Restrictions

Usually two or more programs can read from the same library at the same time. However, the following library functions require that no other programs use the same library at the same time:

- Condensing a library using the CONDENSE procedure.
- Restoring a library using the RESTLIBR procedure.
- Removing all library members or all library members of one type using the REMOVE procedure.
- Renaming a library using the RENAME procedure.
- Deleting a library using the DELETE procedure.
- Copying IBM-supplied library members into the system library.
- Reallocating a library using the ALOCLIBR procedure.

Changing Libraries in a Job

If you decide to use multiple libraries, you often need to change libraries in the middle of a job. The LIBRARY OCL statement specifies the library to be used for a job or a specific job step. The *System Reference* manual has information about the LIBRARY OCL statement.

Securing Libraries

You can secure both user libraries and the system library by using resource security. The access levels you can specify for resource security are shown below in descending order, with the highest access levels having the greatest authority. The authority given by a higher access level includes all the authority of the access levels below it (except, of course, for an access level of None). For example, the user with change access to a library also has update and read access to it.

These access levels allow you to decide who is the owner of the library, who can change the library or members in the library, and whether operators can only run programs stored in the library.

Access Level **Description**

Owner The owner of a library can:

- Indicate the users of the library and their access levels.
- Create, rename, or delete the library.
- Create, change, run, list, remove or copy any member of the library.

Change A user with change access to a library can:

- Create, rename, or delete the library.
- Create, change, run, list, remove or copy any member of the library.

Update A user with update access to a library can create, change, run, list, remove or copy any member of the library.

Read A user with read access to a library can run, list, or copy any member of the library.

Run A user with run access to a library can run any member of the library.

None A user with this access level is not allowed to use any member of the library.

The *System Security Guide* has more information about how to secure libraries.

Backup and Recovery Considerations

If a member is being created or changed and if the program changing the member terminates abnormally, the changes may or may not have been added to the member. The source entry utility (SEU) and Development Support Utility (DSU) automatically create a work file to help with recovery from abnormal terminations. The *SEU Guide* or *DSU Guide* has more information about SEU or DSU.

If you were copying a member to a library (for example, by using the TOLIBR procedure) when an abnormal termination occurs, you should check the member to ensure it is correct. If the member is not correct, do the copy again. Also, if you were copying several members to a library, some of the members may not have been copied at all.

Some general objectives for library backup and recovery are:

- To ensure your data is correct
- To minimize recovery time for each program

If you make any changes or additions to a library or library member, you should create a backup copy of the library. You will then have a current copy of the libraries and members on diskette or tape in case a machine or program error occurs, and you won't have to reenter all the changes.

You must be able to reconstruct your libraries if a failure occurs so that your applications can continue to operate. For more information about backup and recovery, see Chapter 19, "Error Prevention, Detection, and Recovery."

To help you back up and recover libraries, you can use the SAVELIBR and RESTLIBR procedures. SAVELIBR saves a library on diskette or tape; RESTLIBR restores a library from diskette or tape.

Library Sector-Mode and Record-Mode Files

Library members can be copied to disk, diskette, or tape files in either of two formats: sector mode or record mode.

Sector-Mode Files

Sector-mode files contain library members that are stored in the internal format used by System/36. These files are only created by the FROMLIBR and SAVELIBR procedures, or by the $MAINT utility program. The system reads data from these files one sector at a time.

Sector-mode files can be used only with another System/36.

The *System Reference* manual has more information about using sector-mode files.

Record-Mode Files

Records in a record-mode file have a special format. One statement from the member is contained in each record. All records have the same length, and the record can be from 40 to 120 characters long. System/36 either pads the record with blanks or truncates the statements within the library member to match the specified record length.

The first record in the file is a COPY statement that defines the library member. The last record in the file must be a CEND statement. The records in between contain the statements in the library member.

If you use the FROMLIBR procedure or the $MAINT utility program to create a record-mode file from either a source or procedure member, the COPY and CEND statements are placed in the file automatically. Otherwise, you must specify the COPY and CEND statements (for example, if a program you coded is to create a file that becomes a library member). Also, if the record-mode file is organized as a direct file on disk or tape, you must have an END statement following the CEND statement.

Record-mode files can be used on other systems if the files are written as basic data exchange files.

The *System Reference* manual has more information about using record-mode files.

Programming Guidelines for Libraries

The following describes the functions you can do with libraries and library members, and indicates the procedures you can use to do the functions. The *System Reference* manual has more information about these procedures.

Creating Libraries

Use the BLDLIBR procedure to create libraries on disk.

Creating and Changing Members

To create or change a library source or procedure member, use the Development Support Utility (the DSU procedure) or the source entry utility (the SEU procedure). The *SEU Guide* and *DSU Guide* have information about using SEU and DSU. You can also create library members by using the $MAINT utility program (if you don't have SEU or DSU).

The screen design aid utility (SDA) allows you to create and change source members for display formats and menus.

Load and subroutine members are created by compilers.

Listing Members and Library Information

When your library is on disk, use the LISTLIBR procedure to:

- List the library's directory.
- List members from the library.
- List information about a library, such as, its size, its location on disk, or the amount of space used in the library.

When you have a library saved on diskette or tape, or you have one or more members in a diskette or tape file, use the LISTFILE procedure to list information about the members.

DSU also lets you list members or the contents of a library (only members that can be edited).

Saving and Restoring Libraries

Use the SAVELIBR procedure to save an entire library on diskette or tape. Use the RESTLIBR procedure to restore an entire library from diskette or tape to disk.

Copying Libraries and Library Members

Use the LIBRLIBR procedure to copy library members from one library to another. Use the FROMLIBR procedure to copy library members from a library to a disk, diskette, or tape file. Use the TOLIBR procedure to copy library members from a file (either disk, diskette, or tape) to a library.

Changing Library or Directory Size

You use the ALOCLIBR procedure to change the size of a library, a library's directory, or both. You can either increase or decrease the sizes. (The directory size of the system library cannot be changed.)

Condensing a Library

Use the CONDENSE procedure to condense a library; that is, to gather all the unused spaces into a single area. This allows you to add more members to a library.

Securing a Library

Library resource security is described under "Securing Libraries" on page 9-13. Use the SECEDIT RESOURCE procedure to secure a library. See the *System Security Guide* for information about using the SECEDIT procedure.

Renaming a Library or Library Member

To rename a library, use the RENAME procedure. To rename a library member, use the CHNGEMEM procedure.

Removing a Library or Library Members

To remove an entire library, use the DELETE procedure. To remove library members, use the REMOVE procedure. DSU also allows you to remove source or procedure members.

Chapter 10. Folders

The purpose of this chapter is to:

- Describe what a folder is.
- Describe the procedures you use to work with folders.

Folder Concepts and Uses

A **folder** is a named area on disk that contains folder members created and used by DW/36 (DisplayWrite/36), Personal Services/36, IDDU (the interactive data definition utility), and PC Support/36.

Types of Folders and Folder Members

Several different types of folders can be created for storing specific kinds of information. Each folder can contain many members. The table that follows shows some of the different kinds of folders and folder members.

Folder Type	**Created by**	**Contents (Member Type)**
Document	DW/36, DW3, DW4, or PC Support/36	Text ot PC files
Mail	Personal Services/36	Mail documents for the entire system
Mail Log	Personal Services/36	Memo status information for tracking mail
Data Dictionary	IDDU	Field, format, and file definitions

Some folder types can contain only one type of member. For example, data dictionary folders can contain only field, format, and file definitions.

Uses of Folders

Data dictionaries created by IDDU are stored in folders. Personal Services/36 stores the mail that it sends and receives in a central mail folder; each user of Personal Services/36 keeps information about the mail in a personal mail log folder. DW/36 lets you create and edit documents; these documents are also stored in folders. You can group information in DW/36 document folders for a specific purpose. For example, a folder named MGRS could contain all letters and memos sent to managers of a company.

Folder Naming Conventions

A folder name can be up to 8 characters long and must begin with an alphabetic character (A through Z). The #, $, and @ characters should not be used because IBM often uses these characters for names on the system. The remaining characters can be any combination of characters (numeric, alphabetic, and special).

Avoid using the following characters because these have special meanings in procedures: commas (,), apostrophes ('), question marks (?), slashes (/), greater than signs (>), equal signs (=), plus (+), hyphens (-), and ALL.

You can create meaningful folder names by using an abbreviated description of the folder. For example:

Folder Name	Description
TXTEMPNU	Document describing office procedure for training new employees
MYMAIL	Your personal mail log
COMDIC1	Data dictionary for a communications file

Folders in a Group Resource

Folders, files, and libraries can be part of a group resource. The group resource must be named, and each file, library, or folder that is part of the group must be named.

For example, a group resource could be PAY: a file that is part of this group could be PAY.FILE, a library could be PAY.LIB, and a folder could be PAY.FOLD. As you can see, the group resource name and the name of the file, library, or folder are separated by a period (.). PAY is called the group identifier.

One advantage of using groups is that they can be secured easily. The *System Security Guide* has more information about securing groups.

The limit of 8 characters for a folder name also applies to names of groups. A period counts as one of the 8 characters.

Folder Layout

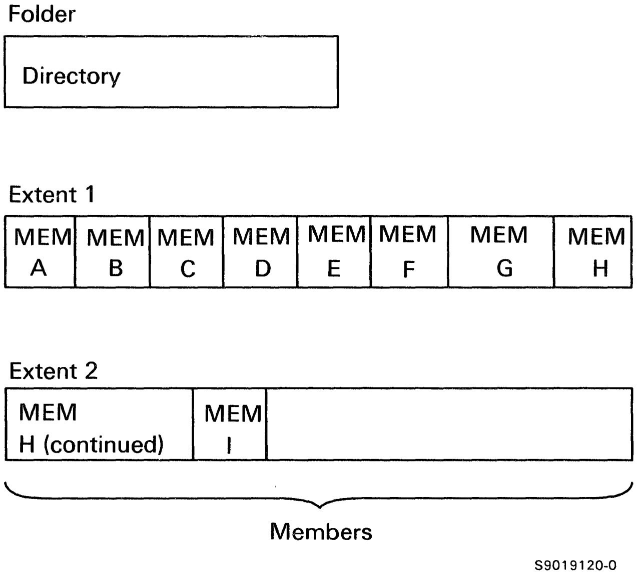
A folder is composed of a *directory*, which contains information about the members and subdirectories in the folder, and a number of *folder extents*, which contain the contents of the members.

Note: Subdirectories are a part of the shared folder facility.

The directory contains an entry with the name, description and member type of each member and subdirectory in the folder. For more information about subdirectories, see Chapter 11, "Subdirectories."

Folder members are stored in disk areas called folder extents. The first part of each folder extent is used to keep track of the folder extent space. Each time you create, change, or move a member, this area is updated to indicate the change in space used and available.

When the system attempts to place a member into a folder extent that cannot contain the member, the system automatically creates an additional extent into which this member can be placed. A folder can have up to 99 extents.



If you attempt to add a member to a folder when all its extents are full or you have reached your limit, the system displays a message, and you must reorganize the folder using the ALOCFLDR procedure with the MIN parameter specified or the CONDENSE procedure with the folder parameter specified.

Folder Size

In general, some of the programs that use folders create the folders for you. For example, Personal Services/36 creates mail log folders. When folders are created, there is no need for you to specify the folder size.

When using IDDU and DW/36, you must have the authority to create folders. Authority for creating folders is specified in your user profile. If you cannot create folders but wish to do so, contact your master security officer or security officer.

If you have the authority to create folders, you can use the TEXTFLDR procedure to create a folder and optionally specify the size. For more information about DW/36 folder size and extension, see "Folder Considerations for DisplayWrite/36" on page 10-6.

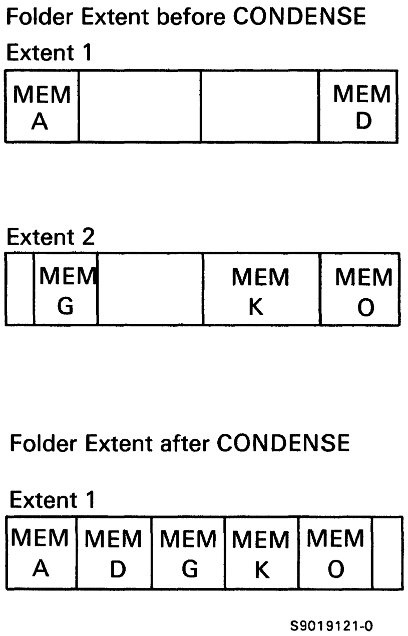
Using this procedure, you provide an approximate number of documents that the folder will contain and the average number of pages for each document. DW/36 translates this information into the number of blocks needed for your folder or you can specify the size in the number of PC files and file size in kilobytes.

You can use the ALOCFLDR procedure to increase or decrease the size of a folder. The *System Reference* manual has information about running the ALOCFLDR procedure.

Reorganizing Folder Space

When you are working with folders, you change, create, or remove members. This can create unusable gaps in the folder extents between the members. As a result, you may want to reorganize your folder, moving all folder members so that they are next to each other. Reorganization may increase the speed with which the system works with the folder; it also makes the unused space free for other uses.

The CONDENSE procedure is used to reorganize a folder and to free all unused space in the folder by placing the members next to each other in the smallest possible number of folder extents. After you run the CONDENSE procedure, the folder is at its minimum size. The following figure shows a folder before and after the CONDENSE procedure has been run.



The reorganized folder area is determined by calculating the amount of space currently being used plus a 10 block system overhead. At best, all of the extents will be combined into one extent; the maximum extent area is 6,553 blocks.

The ALOCFLDR procedure is used to reorganize the folder and, optionally, to increase or decrease the size of the folder. ALOCFLDR with MIN specified works the same as CONDENSE.

The *System Reference* manual has more information about the ALOCFLDR and CONDENSE procedures.

Folder Considerations for DisplayWrite/36

The security officer can specify in your user profile the maximum size of a folder you can create. If you exceed the size limitations specified in your user profile, the system sends you a message telling you that the block size is too large. If a larger maximum size is required, contact your security officer.

When the maximum folder size is exceeded, the folder must be reorganized using the ALOCFLDR procedure with the MIN parameter specified. However, the authorized size limit may prevent reorganization of the folder. If this occurs, you must remove data from the folder or you must ask the security officer to increase the maximum size of the folders you can create.

You can specify the maximum number of extents that are added automatically to the folder. Folders can have from 1 to 98 extents. The default value is 6. If you specify a value close to 98, you will not have much warning before you run out of space and you could lose part of your document. When the limit you specified is reached, you must reorganize the folder to reduce the number of extents or you must enlarge the folder.

Extension continues to occur up to the maximum limit you specified. When you have reached that limit, the system will issue a warning telling you that you have reached the maximum number of extents. The folder or a folder member cannot be opened for any purpose that would cause further extension. However, if you are currently updating a member when the maximum is exceeded, you are permitted to complete your operations even though it may result in several more extensions. The maximum number of extents specified when the folder was created cannot be changed without creating a new folder and copying the original folder members into it.

Folder Sharing Restrictions

Usually two or more programs can read from the same folder at the same time. However, the following folder functions require that no other programs use the same folder at the same time:

- Reorganizing a folder using the ALOCFLDR or CONDENSE procedure
- Restoring a folder using the RESTFLDR procedure
- Renaming a folder using the RENAME procedure
- Deleting a folder using the DELETE procedure

Securing Folders

The security officer can specify in your user profile whether you can create folders when using IDDU and DW/36.

You can secure entire folders, subdirectories, or individual folder members by using resource security. The access levels that you specify for resource security allow you to decide who can read or change a folder, subdirectory, or members in a folder or subdirectory.

For folders, subdirectories, or folder members to which you have owner access, you can specify the access levels for other users on an authorization list. An authorization list is a list of user IDs and their access levels. You assign a name to each authorization list. Then, when you secure a folder, subdirectory, or folder member, you specify an authorization list name for it. You can use the same authorization list for many folders, subdirectory, and some folder members.

Only the owner of an authorization list can change or remove it.

Access levels for folders and folder members are listed in Figure 10-1 on page 10-8 and Figure 10-2 on page 10-10 with the highest access level (Owner) having the greatest authority. The authority that is given by a higher access level includes all the authority of the access levels below it (except, of course, for an access level of None). For example, the user with change access to a folder also has update and read access to that folder.

For information about access levels for subdirectories, see Chapter 11, "Subdirectories."

The *System Security Guide* has more information about how to secure folders, subdirectories, and folder members.

Access Levels for Folders

Description	Owner Access Level	Change Access Level	Update Access Level	Read Access Level	None Access Level
Read, revise, or delete security information for a folder or subdirectory.	X				
Rename a folder	X				
Add, revise, or delete security information for any member in the folder or subdirectory (whether or not the user has access to the member)	X				
Read, revise, or copy any information in any members of a folder or subdirectory	X				
Create or delete a folder or subdirectory at the folder level	X	X			
Save the folder on disk, diskette, or tape	X	X			
Restore the folder from disk, diskette, or tape	X	X			
Read, revise, or copy information in folder members for which a user has at least update access	X	X			
Create and secure folder members	X	X	X		
Delete folder members for which a user has at least update access	X	X			
Read, revise, or delete security information for folder members that a user owns	X	X	X		
Reorganize the folder and optionally change the size	X	X	X		

Figure 10-1 (Part 1 of 2). Access Levels for Folders

Description	Owner Access Level	Change Access Level	Update Access Level	Read Access Level	None Access Level
Restore the folder on disk, diskette, or tape for which a user has update access to all the folder members	X	X	X		
Save the folder on disk, diskette, or tape for which a user has read access to all the folder members	X	X	X		
Read and copy information in folder members for which a user has update access	X	X	X		
Read or copy information in a folder member for which a user has at least read access	X	X	X	X	

Figure 10-1 (Part 2 of 2). Access Levels for Folders

Access Levels for Folder Members

Access Description	Owner Access Level	Update Access Level	Read Access Level	Run Access Level	None Level
Revise or delete security information for a folder member if the user also has at least update access to the folder or subdirectory	X				
Secure folder members	X				
Rename folder member	X				
Read, revise, or delete a folder member if the user has at least update access to the folder or subdirectory	X	X			
Retrieve folder members from disk, diskette, or tape if the user has at least read access to the folder or subdirectory	X	X			
Read or copy information in a folder member if the user has at least read access to the folder or subdirectory	X	X	X		
Run a PC file if the user has at least read access to the folder or subdirectory	X	X	X	X	
Archive folder members on disk, diskette, or tape if the user also has at least read access to the folder or subdirectory	X	X	X	X	X

Figure 10-2. Access Levels for Folder Members

Note: *If a member has its own security, a user must have the proper authorization to access the member. If a member does not have its own security, it assumes the same security as the folder or subdirectory.*

Backup and Recovery Considerations

Some general reasons for folder backup and recovery are:

- To protect against loss of data
- To ensure your data is correct
- To minimize recovery time for each program

If you make any changes or additions to a folder or folder member, you should create a backup copy of the folder. You will then have a current copy of the folders and folder members on diskette or tape in case a machine or program error occurs, and you will not have to reenter all the changes.

You can save a folder member or all folder members marked for archive from disk to diskette or tape by entering the ARCHIVE procedure command. You can restore a folder member from diskette or tape to disk by entering the RETRIEVE procedure command.

ARCHIVE and RETRIEVE can also be used to save or restore folder members to or from a file on disk as well as tape or diskette.

You can also use the SAVEFLDR and RESTFLDR procedures to back up and recover entire folders. SAVEFLDR saves a folder to disk, diskette, or tape, or all folders on the system to diskette or tape; RESTFLDR restores a folder from disk, diskette, or tape.

SAVEFLDR and RESTFLDR can also be used to save or restore a folder to or from a file on disk as well as tape or diskette.

If a folder member is being created or changed and if the program changing the folder member ends abnormally, the following occurs:

- Changes made to the directory, such as member name and member type, are retained. These changes are effective as soon as you enter them.
- Changes to the contents of the member are not made until you save the member. When the program is run again, you will have a copy of the folder member before you made any changes (old version) and a copy of the folder member with the changes (new version) up to the time the program ended abnormally. You can select either the old or the new version and continue your job. If you select the new version, verify that your previous changes are correct. Generally, if the program ended abnormally, changes made after you pressed the Enter key or a command key are lost.

For DW/36, if you were copying a member to a folder (for example, by using the TEXTDOC procedure) when the program ended abnormally, delete the new member and do the copy again. If you were copying several folder members to a folder, some of the folder members may not have been copied at all.

For more information about backup and recovery, see Chapter 19, "Error Prevention, Detection, and Recovery."

Working with Folders in General

The programs that create folders let you select menu options to work with folders. You can also work with folders by entering procedure names and parameters. The following is a list of some of the things you can do with your folders:

- Changing folder size
- Create folders
- Create folder members
- List information about folders
- List information about folder members
- Remove folders from disk
- Remove folder members from disk
- Rename folders
- Rename folder members
- Reorganize folders
- Restore folders from disk, diskette, or tape
- Restore folder members from disk, diskette, or tape
- Save folders to disk, diskette, or tape
- Save folder members to diskette or tape
- Secure folders
- Secure folder members

The following information describes functions and procedures you can while working with folders. For more information about these procedures, see the *System Reference* manual. The online information for DW/36, IDDU, and Personal Services/36 and the following manuals have information about working with folders through menus:

- *Getting Started with DisplayWrite/36*
- *Getting Started with Interactive Data Definition Utility*
- *Getting Started with Personal Services/36*

Changing Folder Size

You use the ALOCFLDR procedure to change the size of a folder. You can either increase or decrease the size. ALOCFLDR reorganizes the folder when it changes the size of the folder: the folder members are moved to the front of the folder to eliminate as many folder extents as possible.

Creating a Folder

For DW/36, you use the TEXTFLDR procedure to create a folder.

For IDDU, you use the IDDUDCT procedure to create a data dictionary (which is a folder).

Note: *For IDDU and DW/36 folders, you must have authority to create folders.*

For Personal Services/36, you use the OFCINSTL procedure to install Personal Services/36 and create the central mail folder; each user creates his personal mail log folder using the OFCMAIL procedure.

Displaying or Printing Folder Members

For DW/36, you use the TEXTDOC procedure to print or display the list of folder members.

Listing Folder Information

Use the CATALOG procedure to list information about a folder on disk, diskette, or tape.

Listing Archived Folder Member Information

Use the LISTFILE procedure to list information about folder members saved (using the ARCHIVE procedure) on diskette or tape.

Removing a Folder

To remove an entire folder, use the DELETE procedure.

Renaming a Folder

To rename a folder, use the RENAME procedure.

Reorganizing a Folder

You can use the CONDENSE or the ALOCFLDR procedure to reorganize a folder to its smallest possible size. You can also use the ALOCFLDR procedure to reorganize a folder and leave the folder size the same or increase or decrease the folder size by a specified number of blocks.

Reorganizing a folder places all the members next to each other at the front of the folder and when possible, decreases the number of extents the folder has.

Saving and Restoring Folders

Use the SAVEFLDR procedure to save an entire folder to disk, diskette, or tape, or all folders on the system to diskette or tape. Use the RESTFLDR procedure to restore an entire folder from disk, diskette, or tape to disk.

Saving and Restoring Folder Members

Use the ARCHIVE procedure to save a folder member or all folder members marked for ARCHIVE on disk, diskette, or tape. Use the RETRIEVE procedure to restore a folder member or all folder members marked for archive from disk, diskette, or tape to disk.

Securing a Folder

Resource security for folders is described under “Securing Folders” on page 10-7. Use the SECEDIT procedure to secure a folder. The *System Security Guide* has information about using the SECEDIT procedure.

Chapter 11. Subdirectories

This chapter discusses the concepts of subdirectories (directories) within shared folders. Subdirectories can be created by DW/36 or by PC Support/36 users.

Subdirectories are used when System/36 users and personal computer users are sharing folders or personal computer users are sharing PC files on System/36 with other personal computer users. Support for shared folders is part of the optional SSP.

The purpose of this chapter is to:

- Describe what is a subdirectory.
- Describe how to manage subdirectories.
- Describe the procedures you use to work with subdirectories.

Subdirectory Concepts and Uses

A subdirectory (directory) is the part of the folder that contains information, such as names, descriptions, member types and security information for folder members and other subdirectories (except security, subdirectory security information is contained in the resource security file).

This section discusses concepts, uses, and management of folders with subdirectories.

Shared Folder Facility

Shared folder facility allows the personal computer user to:

- Work with folders created by DW/36 users and take advantage of the functions of some of the System/36 office products.
- Access members directly in a folder on the system instead of copying members from virtual disk into a folder.
- Use all the capabilities of virtual disk.

A folder with subdirectories has a hierarchical structure beginning at the root and proceeding downward to dependent member types. This structure was added to folders so that personal computer users can share PC files or members with other personal computer users or System/36 users. The relationship between folders and subdirectories allows you to group and store folder members at different levels in the folder.

Using Subdirectories

Subdirectories allow you to better manage your folders and folder members. Similar members, such as spreadsheets, memos, or documents, can be grouped together in one subdirectory.

Managing Subdirectories

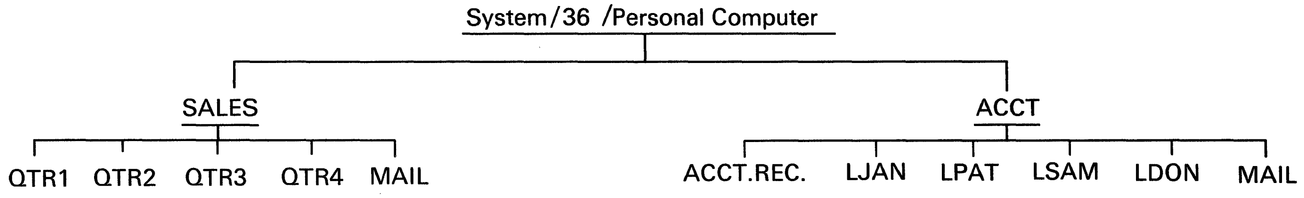
Subdirectories give you the ability to better organize your folders and disk space. Instead of placing a large number of files or folder members in two or three folders, which could be inefficient for both you and your system, you can group similar files or folder members in several subdirectories within a single folder.

Organization and Identification

If you have files or folder members that are spreadsheets, memos, or documents, you can place all of your spreadsheets in one subdirectory and do the same for memos and documents in another subdirectory.

For example, let us assume you have two departments that share the same folders and folder members. All the folders are stored on System/36. The two departments, Sales and Accounting, share different types of files such as spreadsheets (PC data), or folder members such as spreadsheets, documents, and mail (System/36 data). Both are contained within folders.

Assume the present organization of the folders looks like the following:



S9019136-0

By using subdirectories, the files and folder members could be organized as follows:

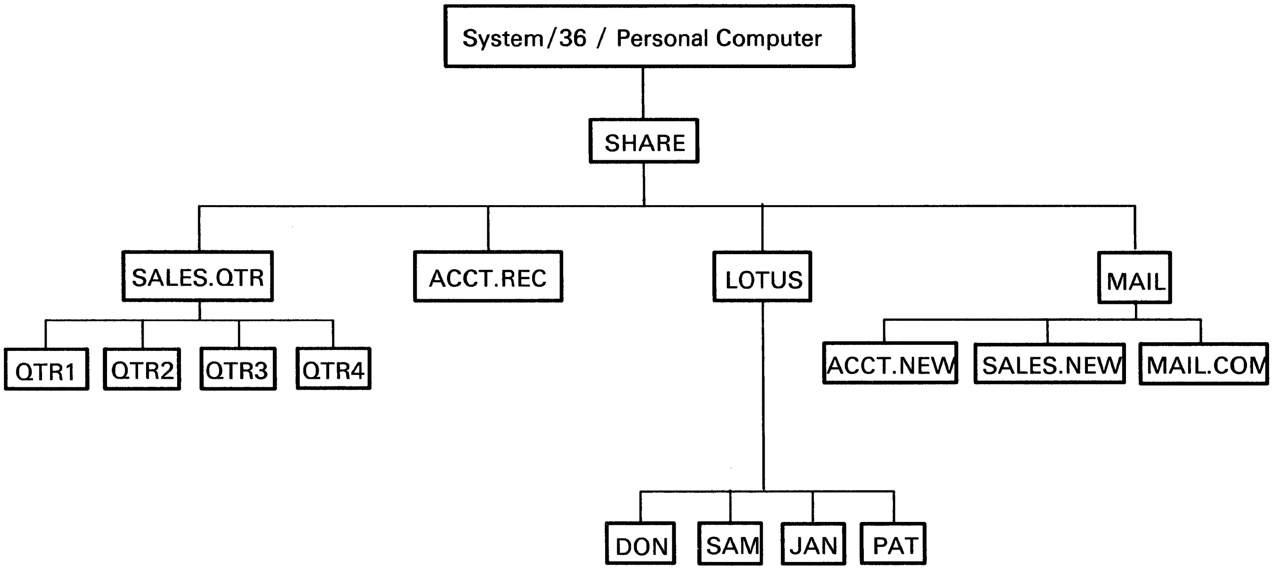


Figure 11-1. Subdirectory Structure

S9019137-0

Organized this way, all files and folder members relating to quarterly sales are in the subdirectory SALES.QTR

All the files and folder members relating to spreadsheets are in the subdirectory LOTUS. All files and folder members relating to mail are in the subdirectory MAIL, which contains files and folder members and a subdirectory called MAIL.COM, which contains answered mail.

Terms Used with Subdirectories

There are several different terms used when discussing subdirectories. They are:

- Root (system) directory
- Subdirectory
- Current directory

Root (System) Directory

The root (system) directory contains the names of all the files and directories on the system.

System/36: The root directory is called the VTOC on System/36. You can use the CATALOG procedure to list all the files, libraries, and folders in the VTOC. You will not see any of the folder members.

You can also get a list of folders or folder members from the Work with Documents display.

Personal Computer: The root directory is the parent of all other directories on that drive.

If you assign a drive to the system, the root directory will contain all the folders on the System/36.

If you assign a drive to a folder that is shared, you will see the members in the folder directory in your root directory.

When you issue a DIR command on the shared folder drive, you will see a list of all the folders on the System/36.

Subdirectory

Subdirectories contain the names of other directories (subdirectories) and folder members that are found within the folder.

Although subdirectories are limited only by the amount of space available in the folder, for better management, three levels are recommended.

The following figure shows an example of a folder with four levels of subdirectories.

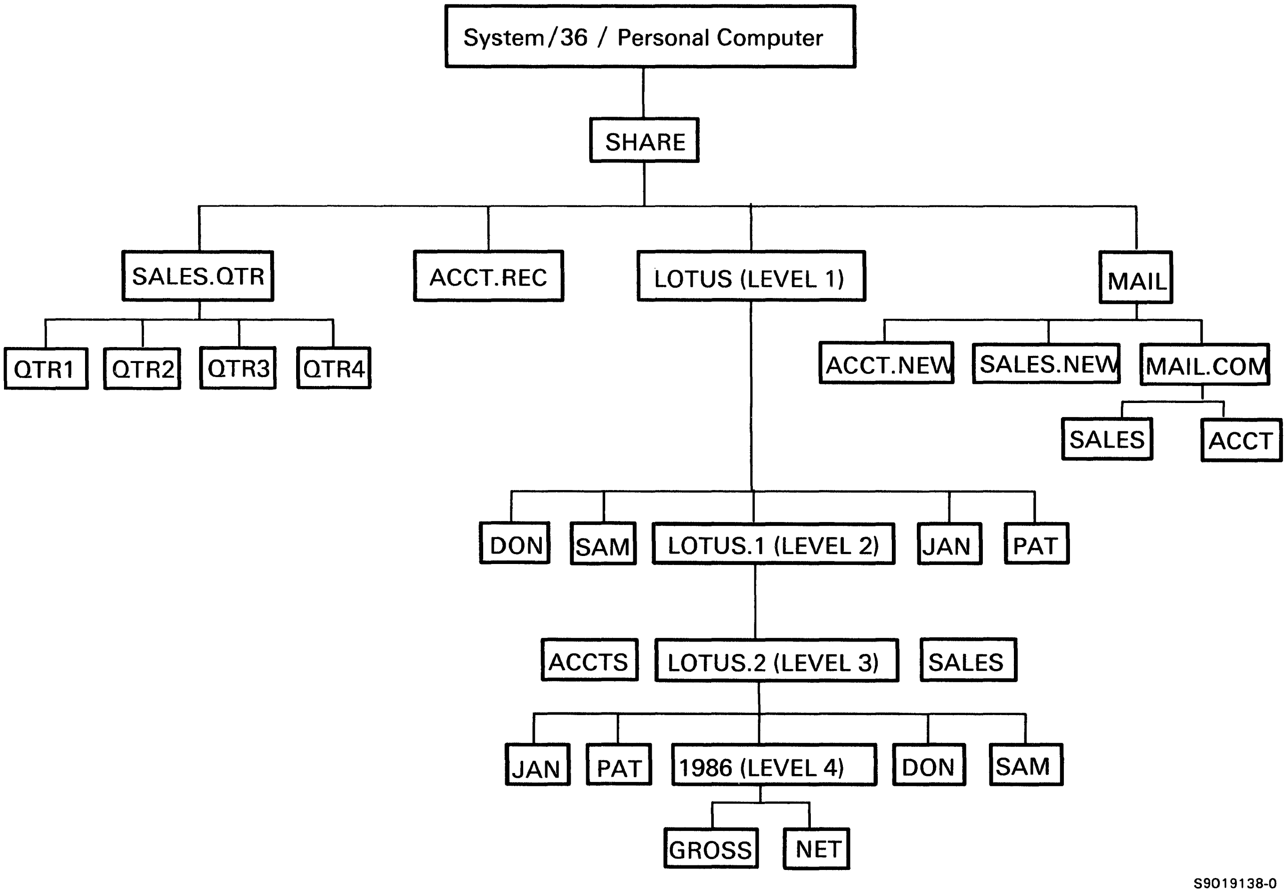


Figure 11-2. Subdirectory Structure

Current Directory

This directory is the default directory for each drive on the system or the directory (subdirectory) you are presently in. The current directory is searched first if no other directory is specified.

In Figure 11-2, the folder (SHARE) has four levels of subdirectories. If you are at the folder level, then SHARE is the current directory. To make 1986 (level 4) the current directory, you must specify the series of directory (subdirectory) names to 1986. For example:

```
/SHARE/LOTUS/LOTUS.1/LOTUS.2/1986
```

System/36: You can change your current directory from a display that lists all of the subdirectories you have access to. Each entry in the list is similar to the above example. You do not see the folder members on this display.

Personal Computer: You can change your current directory by using the change directory (CD) command. The change directory command is discussed later in this chapter.

When you specify a current directory and then use the DIR command, you will see all subdirectories and files or folder members that you have access to at the current directory level.

Naming Conventions

A subdirectory name can be up to 8 characters long followed by a period (.) and a 3-character extension and must begin with an alphabetic character (A through Z) or a numeric character (0 through 9). The #, $, and @ characters are valid but should not be used because IBM often uses these characters for names on the system. The remaining characters can be any combination of characters (numeric, alphabetic, and special).

The following characters are not allowed: greater than signs (>), double quotes ("), plus signs (+), equal signs (=), semicolons (;), commas (,), asterisk (*), question marks (?), and colons (:). If you try to create a subdirectory name with any of these characters, it will be rejected by System/36.

Do not use back slash (\), forward slash (/), less than sign (<), and double bars (|). These characters have special meaning on the personal computer.

Subdirectory names and member names must be different if they are at the same level in the folder.

You can create meaningful subdirectory names by using an abbreviated description of the contents of the subdirectory. For example:

Subdirectory Name	**Description**
TXTEMPNU.PAY	Subdirectory for documents describing office procedures for training new employees
MYMAIL.NEW	Subdirectory for your personal mail logs

It is possible to create a PC file name, System/36 document name, or System/36 folder name that will not be valid for the personal computer, System/36, or both. The following table shows some names that are valid or invalid for the System/36 or the personal computer.

Example Name	PC File Name	Document Name	Folder Name
/TEXT	No	No	No
>FRED	No	Yes	No
A"	No	Yes	Yes
GC010886	Yes	Yes	Yes

The following is an explanation of the items in the table.

- /TEXT is not a valid name for a PC file, a document, or a folder, and is rejected by both System/36 and the personal computer because the forward slash is used to define root directories.

- >FRED is a valid document name on the System/36 but is not valid for folders or PC files because the greater than sign (>) has special meaning on the personal computer.

- A" is a valid folder name and document name on the System/36 but is not valid for a PC file. Because folder A" is not valid for the personal computer, it will not show as a PC file and will not allow access to the folder or folder members.

  For example, if the following folders and documents are on the System/36:

Folder Name	Document Name	Document Name	Document Name
TXTFRED	XYZ	122285	010886
TXTJOHN	STUFF	JUNK	XYZ
SOMEFLDR	DOC1	DOC2	DOC3
A"	DOC1	XYZ	DOC2

  Then, when Fred is using the personal computer at the root directory level and uses the DIR command, he will see in his directory:

```
TXTFRED   <DIR>   mm-dd-yy  hh : mm
TXTJOHN   <DIR>   mm-dd-yy  hh : mm
SOMEFLDR  <DIR>   mm-dd-yy  hh : mm
A"        <DIR>   mm-dd-yy  hh : mm
```

  Even though A" is not a valid name for the personal computer, folder A" would appear as a directory on the personal computer. If Fred uses the CD command (change directory) to make folder A" his current directory, the system would issue an error message telling him that A" is not a valid directory. Therefore, Fred cannot access any document in folder A".

- GC010886 is valid for PC files, System/36 documents and folders because the name is valid for *both* System/36 and the personal computer.

Folder Layout with Subdirectories

A folder contains two major areas: the directory and folder extents.

The directory contains the names, descriptions, and member types of the subdirectories and folder members in a folder. It can also contain security information about the folder and subdirectories.

With folders, it is possible to have more than one member type. Each time a new subdirectory is created, a new member type is added to the folder by the system. For example, if the first member type in a folder is 10, and three new subdirectories (SALES.QTR, LOTUS, and MAIL) are created, they would also be assigned member type 10.

All members in SALES.QTR would be assigned member type 11. The subdirectory and members in LOTUS would be assigned member type 12. All subdirectories and members in MAIL would be assigned member type 13.

If another subdirectory (LOTUS.1) and members are created in LOTUS, the member type for LOTUS.1 and the members would be assigned member type 14. A subdirectory (MAIL.COM) and members created in MAIL would be assigned member type 15.

If a subdirectory (LOTUS.2) and members are created in LOTUS.1, the member type for LOTUS.2 and the members would be assigned member type 16. A subdirectory (1986) created in LOTUS.2 would be assigned member type 16 but the members in 1986 would be assigned member type 17.

The folder directory would be similar to the following:

Folder Directory

	Member	Member Type
Member Type 10	SALES.QTR	11
	ACCT.REC	
	LOTUS	12
	MAIL	13
Member Type 11 (SALES.QTR)	QTR1	
	QTR2	
	QTR3	
	QTR4	
Member Type 12 (LOTUS)	DON	
	SAM	
	LOTUS.1	14
	JAN	
	PAT	
Member Type 13 (MAIL)	ACCT.NEW	
	SALES.NOW	
	MAIL.COM	15
Member Type 14 (LOTUS.1)	ACCTS	
	LOTUS.2	16
	SALES	
Member Type 15 (MAIL.COM)	SALES	
	ACCTS.	
Member Type 16 (LOTUS.2)	JAN	
	PAT	
	1986	17
	DON	
	SAM	
Member Type 17 (1986)	GROSS	
	NET	

S9019139-0

Figure 11-3. Folder Layout with Subdirectories

For information about folder extents, see "Folder Layout" on page 10-3 in Chapter 10, "Folders."

Securing a Subdirectory

To restrict access to a specific subdirectory, you can secure it with the SECEDIT RESFLDR procedure. In this procedure, you must first specify the series of subdirectory names to the subdirectory you want to secure. The subdirectory names in the series are separated by a forward slashes (/). The last subdirectory in the series is the one you are securing. The sum of the forward slashes, characters in folder name, and subdirectory names cannot exceed 63 characters.

The access levels that you specify for resource security allow you to decide who can read or change a subdirectory and the members in that subdirectory. For subdirectories, to which you have owner access, you can specify the access levels for other users on an authorization list. An authorization list is a list of user IDs and their access levels. You assign a name to each authorization list. Then, when you secure a subdirectory, you specify an authorization list name for it. You can use the same authorization list for many subdirectories. Only the owner of an authorization list can change or remove it.

The access levels are listed below in descending order, with the highest level having the greatest authority.

The authority given by the higher access level includes all the authority of the access levels below it (except, of course, an access level of None). For example, a user with change access to a subdirectory also has update and read access.

Access Levels for Subdirectories

Access Level **Description**

Owner A user who has owner access to a subdirectory can:

- Create, secure, or delete a subdirectory.
- Create and secure folder members.
- Revise security information for a subdirectory or any folder member in that subdirectory.
- Revise information in a folder member or delete folder members. access.
- Read or copy information in a member.

Change A user with change access to a subdirectory can:

- Create, secure, or delete a subdirectory.
- Create or secure folder members.
- Revise information in the folder members or delete folder members for which he has update access.
- Read or copy information in a member.

Update — A user with update access to a subdirectory can:

- Create or secure folder members.
- Revise information in folder members or delete folder members for which he has update access.
- Read or copy information in a member for which he has read access.

Read — A user with read access to a subdirectory can read and copy information in the folder members for which he has read access.

None — A user with an access level of none cannot access any members in a subdirectory.

The *System Security Guide* has more information about securing subdirectories.

Backup and Recovery Considerations

Backup and recovery for subdirectories must be done at the folder level on System/36.

Some general reasons for backup and recovery are:

- To ensure your data is correct
- To minimize recovery time for each program

If you make any changes or additions to a folder, subdirectory, or folder member, you should create a backup copy of the folder. You will then have a current copy of the folders, subdirectories, and folder members on diskette, or tape and you will not have to reenter all the changes in case a machine or program error occurs.

You can use the SAVEFLDR and RESTFLDR procedures to back up and recover entire folders with all subdirectories and files or folder members.

You can also save a folder member in a subdirectory marked for archive to diskette or tape by entering the ARCHIVE procedure command. You can restore a folder member to a subdirectory from diskette or tape by entering the RETRIEVE procedure command.

For more information about backup and recovery, see "Backup and Recovery Considerations" in Chapter 10, "Folders."

Working with Subdirectories

You can work with folders and subdirectories by selecting menu options or by procedures and commands from the System/36 or by entering commands from the personal computer. The following is a list of the things you can do with subdirectories.

- Assign a drive to a folder
- Change the current directory
- Create a subdirectory
- Specify a path to a file or folder member
- Display a list of subdirectories
- Display a list of folder members in a subdirectory
- Save folder members in a subdirectory
- Restore a folder member to a subdirectory
- Delete a subdirectory
- Reorganize a subdirectory at the folder level
- Secure a subdirectory

The following describes these functions and indicates the commands or procedures you can use to do them.

Assigning a Drive to a Folder

System/36: You do not have to assign a drive to a folder. You can access folders with the TEXTDOC procedure.

Personal Computer: You must first assign a drive to System/36 before you can work with shared folder support.

You can assign a drive in two ways or a combination of both. The first way is to assign one drive per System/36. The second way is to assign one drive to a folder or a subdirectory in a folder. You can, however, assign one drive to the system and have other drives assigned to folders.

If you assign a drive to the system, you will see the folders that you have access to at the root directory level.

If you assign a drive to a folder, you will see the files or folder members and subdirectories in the folder that you have access to. Assigning one drive per folder is similar to using virtual disk.

Whichever way you choose, you must use the following command to assign a drive to System/36 or to a folder.

```
FSPC ASSIGN <d:> </Path>
```

Following are the two parameters that can be specified with the assign command:

Parameter	**Description**
d:	The drive that you want to access the System/36 with. If this parameter is not specified, the next available drive will be used.
Path	The series of directory (subdirectory) names to the System/36 directory you want to assign to the drive. `ASSIGN E:/DIRa/DIRb/DIRc` If you do not specify the path parameter or only specify a < >, the path defaults to the system directory.

Changing the Current Directory

System/36: You can change your current directory by selecting a different folder or subdirectory from a list on the Work with Documents display.

Personal Computer: If you are at the *root directory* level and are using a system drive, you must use the change directory command to change the current directory before you can create a subdirectory. The CD command tells the system which directory it should remember as the current directory. If you do not use the CD command, the system will default to the root directory of the personal computer.

For example, you have assigned drive E to System/36 and drive F to a folder. To change your current directory (drive E) to the directory of the folder (drive F), type the following:

```
F:
```

This command will take you to the drive you have assigned to the folder. The folder directory is now the current directory and you will see all the files or folder members at the folder level.

Creating a Subdirectory

System/36: You can use the DEFSUBD procedure to create a subdirectory. The *System Reference* manual has more information about the DEFSUBD procedure.

Personal Computer: If you have assigned a drive, you can use the make directory command (MD) to create a subdirectory. (You cannot create a shared folder with the MD command: you must use the TEXTFLDR procedure on the System/36.) With the MD command, you must specify certain parameters depending on which directory is current.

If your current directory is the root directory, you must use the CD command before you can access a folder in drive E. For example:

```
CD E:
```

To create a subdirectory, you must specify the series of subdirectories names preceding it. The subdirectory names in the series are separated by forward slashes (/) and can be up to 63 characters in length.

For example, you want to create a subdirectory in folder TXTEMPNU assigned to drive E on the system, and you are at the folder level. To create a subdirectory named TXTEMPNU.PAY, you would specify:

```
MD /TXTEMPNU/TXTEMPNU.PAY
```

You can create as many subdirectories as you want, however you should make sure that the longest series of subdirectory names you create from the forward slash and root directory to the last subdirectory is not more than 63 characters in length (including forward slashes). You cannot change the name of a subdirectory after you have created it.

If you have changed your current directory to the folder directory, you do not have to use the CD command because the system will default to that folder directory.

Specifying a Path to a File or Folder Member

Once you have created a subdirectory and files or folder members reside within the subdirectory, anytime you wish to access a file or folder member, you must specify the series of subdirectory names to that file or folder member. This series of subdirectory names is called a path.

System/36: You specify the document you want to work with from the Work with Documents display. From this display, a command key 14 will take you to the Select a Subdirectory display. On this display, a list of paths is shown. You select from this list the path or series of subdirectory names you want for the folder member you specified. The path or series is similar to the personal computer example showing the path to JANET except JANET is not shown. JANET is specified on the Work with Documents display.

Personal Computer: You must specify the series of directory (subdirectory) names to the file or folder member. For example, you have a file or folder member named JANET that resides in the folder TXTEMPNU, but also within the subdirectory TXTEMPNU.PAY. If you are at the folder level, you must specify the following to access JANET:

```
/TXTEMPNU/TXTEMPNU.PAY/JANET
```

If you are at the personal computer directory (root) level, you must also specify the drive ahead of the path by using the DIR command.

```
DIR E:/TXTEMPNU/TXTEMPNU.PAY/JANET
```

By specifying the series of directory (subdirectory) names to JANET, the system will search in sequence TXTEMPNU (root directory) first until it finds the subdirectory TXTEMPNU.PAY. After the system finds TXTEMPNU.PAY, the system will search until it finds JANET.

If you have the right security access level to the subdirectory and the member does not have its own security information, you can work with that folder member. For more information on access levels, see “Securing a Subdirectory” earlier in this chapter.

Listing Subdirectories

System/36: You can get a list of subdirectories from the Work with Documents display.

Personal Computer: If you are a personal computer user, use the DIR command to get a list of subdirectories. If you do not specify the subdirectory you want, it will default to the current directory.

Once you are at the specified subdirectory level, you can get a list of subdirectories within that subdirectory.

Listing Members in a Subdirectory

System/36: You can get a list of files or folder members from the Work with Documents display.

Personal Computer: To get a list of files in a subdirectory, use the DIR command with the subdirectory parameter. If you do not specify the subdirectory you want, it will default to the system directory.

Saving Folder Members in a Subdirectory

To save folder members in subdirectories to disk, diskette, or tape, use the ARCHIVE procedure.

The subdirectory parameter is optional when using the ARCHIVE procedure. However, if you specify a subdirectory, the system will use the subdirectory to find the correct member to archive. If you do not specify the subdirectory, the system will archive the member in the base folder (root directory). The *System Reference* manual has more information about the ARCHIVE procedure.

Restoring Members to a Subdirectory

To restore folder members from disk, diskette, or tape to a specific subdirectory, use the RETRIEVE procedure.

The subdirectory parameter is optional when using the RETRIEVE procedure. However, if you specify a subdirectory, the member will be restored to that subdirectory. If you do not specify a subdirectory, the system will restore the member to the base folder (root directory). The *System Reference* manual has more information about the RETRIEVE procedure.

Deleting a Subdirectory

Subdirectories can be deleted (removed) only if they do not have any members or subdirectories in them. The root directory or current directory cannot be deleted.

System/36: You can delete a subdirectory or a list of subdirectories from a display or you can use the DEFSUBD procedure. The *System Reference* manual has more information about the DEFSUBD procedure.

Personal Computer: Use the remove directory command (RD) to remove a subdirectory. (You cannot delete a shared folder with the RD command: you must use the TEXTFLDR procedure on the System/36.) Only one directory can be removed at a time.

If you want to remove a subdirectory named TXTEMPNU.PAY and it resides in folder TXTEMPNU , you can remove it using the RD command. For example:

```
RD /TXTEMPNU/TXTEMPNU.PAY
```

The RD command specifies to the system that you want the last subdirectory in the series of directory (subdirectory) names deleted, therefore, subdirectory TXTEMPNU.PAY will be deleted from the folder TXTEMPNU.

Reorganizing Subdirectory Space at the Folder Level

Reorganizing space in a subdirectory can only be done at the folder level. For more information, see "Reorganizing Folder Space" in Chapter 10, "Folders."

Securing Subdirectories

To secure a subdirectory, use the SECEDIT RESFLDR procedure. The *System Security Guide* has more information about the SECEDIT RESFLDR procedure and securing subdirectories.

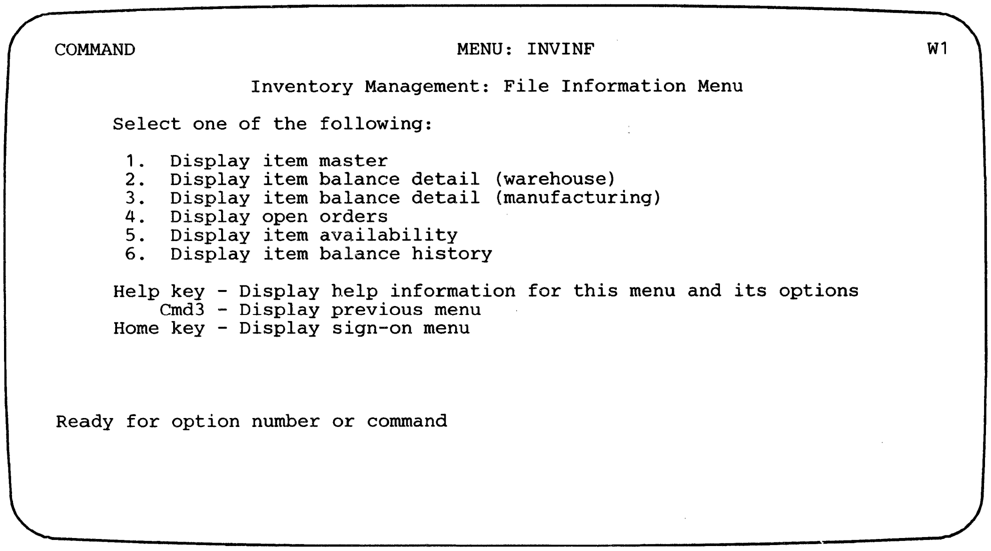
Chapter 12. Menus and Menu Design

Note: The information in this chapter is intentionally similar to the information in Chapter 1 of the manual ***Creating Displays: Screen Design Aid and System Support Program.***

The purpose of this chapter is to introduce menus and to describe how you can design and create them.

Application users can select the work they want to do from menus. A menu is a displayed list of options; each option has an option number and a brief description of the job. When the user enters the option number for a particular job description, the system runs the job associated with that option number.

The following is an example of a menu, INVINF, that a warehouse foreman might use. This menu is used to display information in the files used by the inventory management application.



```
COMMAND                                MENU: INVINF                              W1
                  Inventory Management: File Information Menu

     Select one of the following:

      1.  Display item master
      2.  Display item balance detail (warehouse)
      3.  Display item balance detail (manufacturing)
      4.  Display open orders
      5.  Display item availability
      6.  Display item balance history

     Help key - Display help information for this menu and its options
         Cmd3 - Display previous menu
     Home key - Display sign-on menu



Ready for option number or command
```

Notice that the menu shows:

- The name of the menu
- A descriptive title for the menu
- The option numbers
- A brief description of what each option does
- Prompts describing other functions that can be performed using this menu

Benefits of Menus

Menus can significantly simplify the duties of anyone who uses your application, and can reduce the chances that the user will make a mistake. Because the user simply selects an option number, he needs no knowledge of the operation control language (OCL) statements, procedures, or control commands needed to run a job. The amount of typing and the chance of error are reduced considerably.

Menus can be used to group jobs by application; for example, all accounts receivable jobs could be listed on one menu, all order entry jobs on another menu, and all inventory management jobs on a third menu. Such grouping keeps related jobs together, allowing users to run several related jobs consecutively.

Using Menus

The application user can display a menu by:

- Entering the name of a menu in the menu field on the Sign-On display. The requested menu appears when the user has signed on.
- Leaving the menu field blank during sign-on if the user is assigned a **default menu**. The default menu appears when the user has signed on. (If the menu field is left blank and the user is not assigned a default menu, the main system help menu appears.)
- Requesting a menu from the main system help menu.
- Selecting a menu from another menu.
- Running a procedure that displays a menu.
- Entering a MENU control command.

The user can respond to a menu by:

- Entering an option number.

- Entering a control command, a procedure, or OCL statements.

- Pressing the Help key to request help for the menu or its options.

- Pressing the Home key to return to the menu named during sign-on or to return to the default menu.

- Pressing command key 3 to return to the menu displayed immediately before the current menu.

- Entering the HELP command or pressing command key 6 to display the system help menu assigned to the user.

- Entering a 0 instead of an option number. This removes the current menu from the display and replaces it with a command display.

Using Menu Security

To limit the jobs that a user can run, you can use the SSP-provided **menu security**. If the user is assigned a default menu, that menu can also be defined as **mandatory**. If the default menu is mandatory, the user is restricted to making selections from that menu or from a menu displayed by an option on the default menu, and to using a few control commands. The user is limited to doing those jobs controlled by the mandatory menu.

The user can also be assigned a system help menu. That help menu is called a **beginning help menu**. When the user enters the HELP command or presses command key 6 when a menu is displayed, the beginning help menu is displayed. The beginning help menu is selected according to the user's responsibilities and level of experience. For example, a user responsible for the printing of output from a particular printer may be assigned the system help menu named SPOOLJOB, the menu used to control spool file entries.

For more information about menu security, default menus, mandatory menus, and beginning help menus, see the manual *System Security Guide*.

Menu Formats

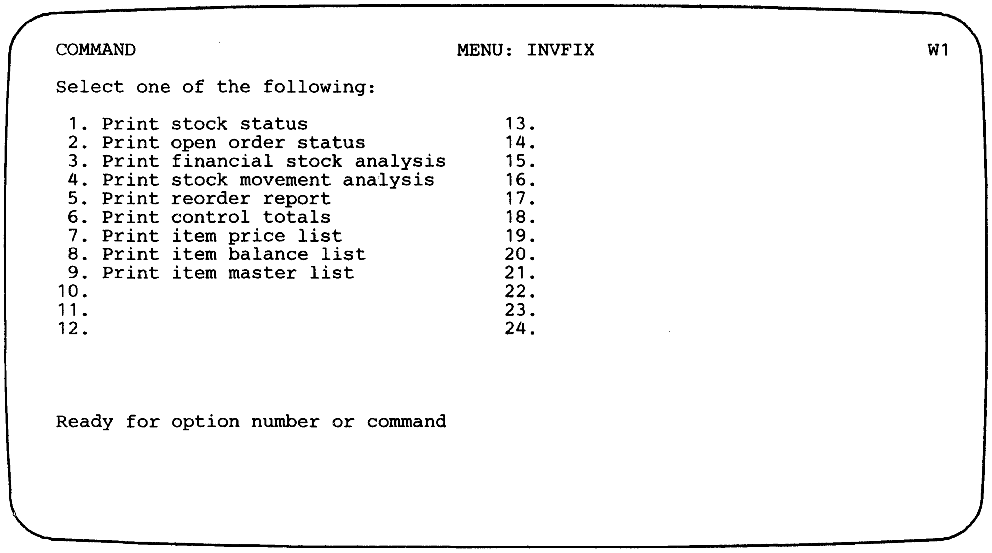
The system allows you to create two types of formats for menus:

- Fixed-format
- Free-format

Menus must be displayed *only* in 24 x 80 format (that is, 24 lines of 80 characters). If you specify 27 x 132 for a menu format (for a 3180 Display), the menu may not display correctly.

Fixed-Format Menus

A fixed-format menu always contains two columns of menu option numbers, 1 through 24, with 12 options in each column. When you create a fixed-format menu, you simply tell the system the name of the menu and which option numbers you will use, give the descriptive text for each option, and supply the procedure to be run for each option. The system formats the information as shown in the following example. (In this menu, the programmer provided the name of the menu, and the text and procedures for option numbers 1 through 9 only.)



```
COMMAND                            MENU: INVFIX                     W1

Select one of the following:

 1. Print stock status              13.
 2. Print open order status         14.
 3. Print financial stock analysis  15.
 4. Print stock movement analysis   16.
 5. Print reorder report            17.
 6. Print control totals            18.
 7. Print item price list           19.
 8. Print item balance list         20.
 9. Print item master list          21.
10.                                 22.
11.                                 23.
12.                                 24.



Ready for option number or command
```

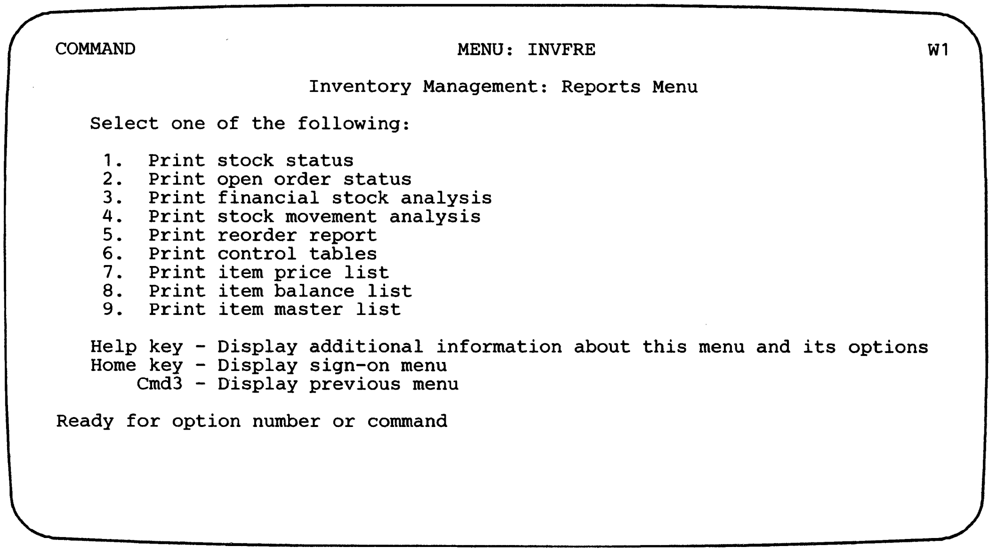
Although all 24 option numbers are displayed on a fixed-format menu, the user can select only the option numbers that have descriptive text and procedures.

Free-Format Menus

A free-format menu contains only the option numbers you want to use; you determine exactly how the menu will appear. A large portion of the display is available to provide descriptive text to your users.

When you create a free-format menu, you can define up to 24 option numbers and descriptions, and the procedure, control commands or OCL statements associated with each option number. You decide the exact placement of the option numbers and their associated descriptions, as well as any other text you might consider helpful to the user.

The following is an example of a free-format menu. Notice that the options on this menu are the same as the options on the previous example of a fixed-format menu, but the programmer decided the placement of the numbers and their descriptions, supplied a descriptive title, and provided information about help text and command key usage.



```
COMMAND                              MENU: INVFRE                                W1
                            Inventory Management: Reports Menu

   Select one of the following:

    1.  Print stock status
    2.  Print open order status
    3.  Print financial stock analysis
    4.  Print stock movement analysis
    5.  Print reorder report
    6.  Print control tables
    7.  Print item price list
    8.  Print item balance list
    9.  Print item master list

   Help key - Display additional information about this menu and its options
   Home key - Display sign-on menu
       Cmd3 - Display previous menu

Ready for option number or command
```

Designing Menus

When you design menus, consider how the menu will be used, the types of jobs you want the menu to control, and the level of experience or responsibility of the user. A well-organized and descriptive menu helps to increase user productivity.

Use the following guidelines when you design your menus:

- Use free-format menus as much as possible. Free-format menus do not have unused option numbers that can clutter the menu and confuse the user.

  Avoid mixing free-format and fixed-format menus within the same application. Consistency is one of the keys to building user confidence and improving productivity.

- Do *not:*

  - Specify the 27 x 132 character attribute for menus.
  - Put any fields with user-supplied data on a menu.
  - Move the input field on a menu.

  If you do any of these things, the results may be unpredictable.

- Write your menus in both uppercase and lowercase letters. TEXT WRITTEN ONLY IN UPPERCASE LETTERS, LIKE THIS SENTENCE, IS DIFFICULT TO READ.

- Number your options beginning with 1.

- Place the most frequently selected options near the top of the menu, or place them in the sequence they are to be selected. If order is not important, arrange the options alphabetically.

- Make the menu title and option descriptions meaningful and descriptive. Use words that describe simply and clearly what job is to be selected. For example, Release orders is a more meaningful option description than RELORD, the name of the program that releases orders.

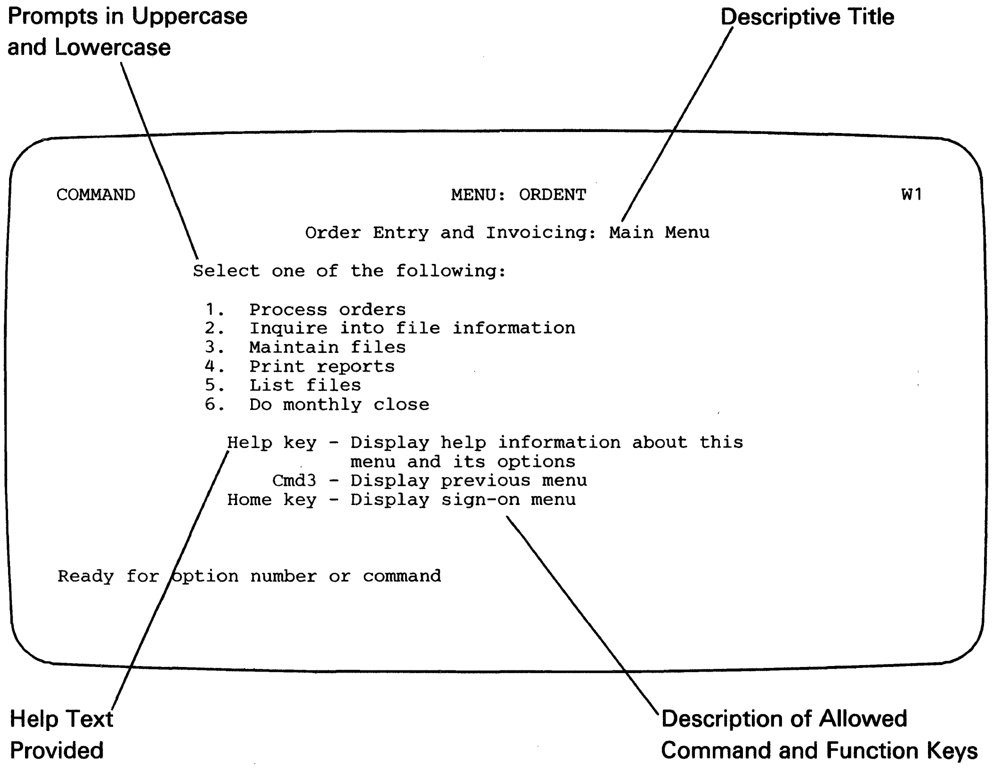
- Use a word that suggests action, such as list or print, for the first word in the option description.

- If the same task is done on several different menus (sign-off, for example), use the same option number for that task on each of the menus.

- Avoid using abbreviations.

- Provide menu help text.

The following sample menu, ORDENT, the main menu for the order entry and invoicing application, illustrates good design techniques:



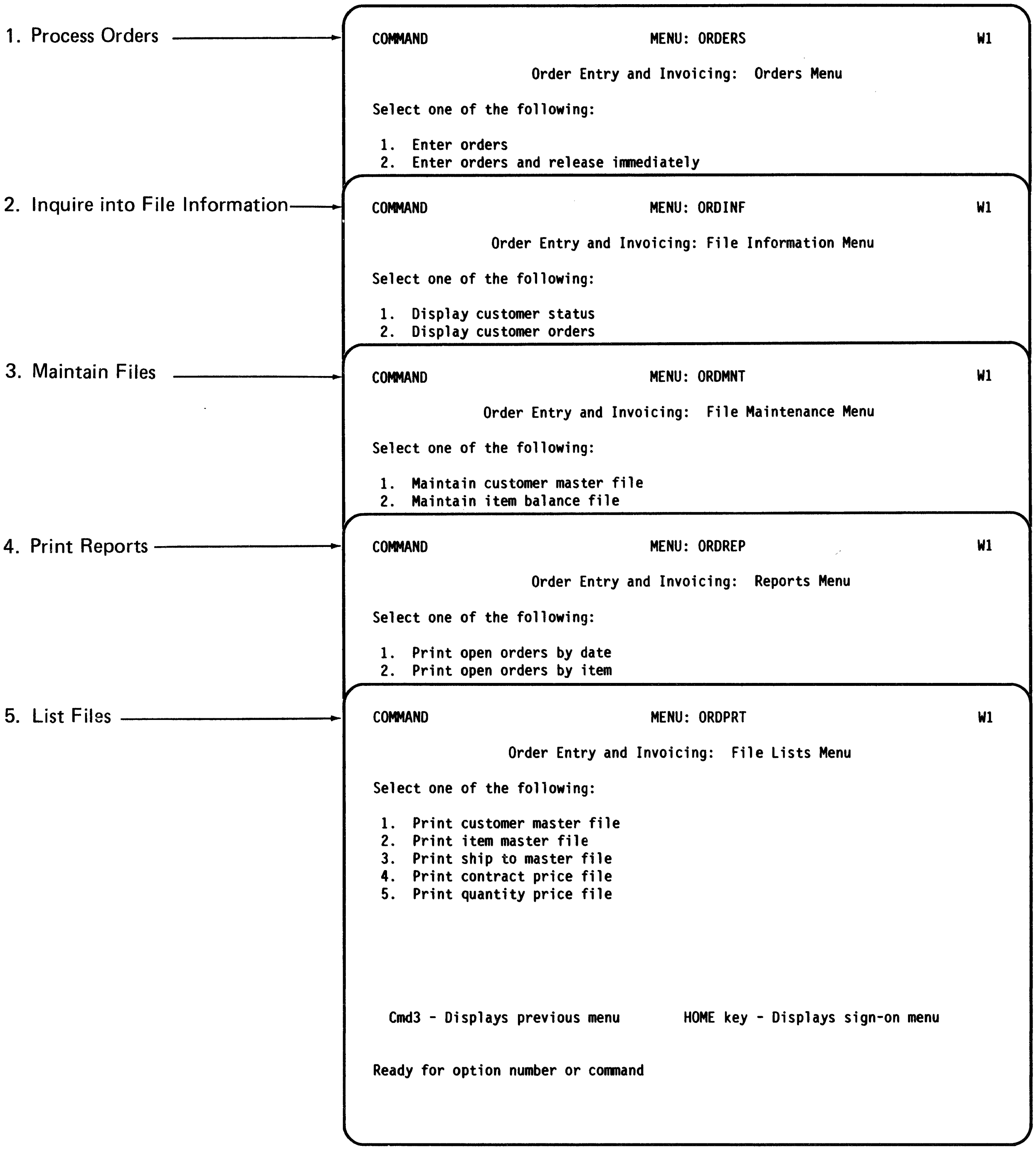
```
COMMAND                            MENU: ORDENT                          W1
                         Order Entry and Invoicing: Main Menu
             Select one of the following:

                  1.  Process orders
                  2.  Inquire into file information
                  3.  Maintain files
                  4.  Print reports
                  5.  List files
                  6.  Do monthly close

                Help key - Display help information about this
                           menu and its options
                    Cmd3 - Display previous menu
                Home key - Display sign-on menu


Ready for option number or command
```

Prompts in Uppercase and Lowercase
Descriptive Title
Help Text Provided
Description of Allowed Command and Function Keys

S9019127-0

Menu chaining is a good technique to use for applications because it helps organize a user's work by guiding the user to the displays needed to do a particular job. Menu chaining uses a main menu that lists other menus (generally more specialized) from which a user can select a job. For example, the previous menu (ORDENT) is the main menu for order entry and invoicing applications.

The following shows the options on ORDENT and the menus chained to ORDENT.



```
COMMAND                            MENU: ORDERS                               W1

                         Order Entry and Invoicing:  Orders Menu

Select one of the following:

 1.  Enter orders
 2.  Enter orders and release immediately
```

```
COMMAND                            MENU: ORDINF                               W1

                    Order Entry and Invoicing: File Information Menu

Select one of the following:

 1.  Display customer status
 2.  Display customer orders
```

```
COMMAND                            MENU: ORDMNT                               W1

                  Order Entry and Invoicing:  File Maintenance Menu

Select one of the following:

 1.  Maintain customer master file
 2.  Maintain item balance file
```

```
COMMAND                            MENU: ORDREP                               W1

                         Order Entry and Invoicing:  Reports Menu

Select one of the following:

 1.  Print open orders by date
 2.  Print open orders by item
```

```
COMMAND                            MENU: ORDPRT                               W1

                      Order Entry and Invoicing:  File Lists Menu

Select one of the following:

 1.  Print customer master file
 2.  Print item master file
 3.  Print ship to master file
 4.  Print contract price file
 5.  Print quantity price file




  Cmd3 - Displays previous menu       HOME key - Displays sign-on menu


Ready for option number or command
```

When a user selects option 1 from the main menu, the MENU control command is run and the Orders Menu appears. When the user selects an option from the Orders Menu, the user sees either another menu if there are additional order processing categories to select, or a display on which the user can begin a job.

The MENU OCL statement and the MENU control command are useful when you are building a menu chain.

The *System Reference* manual has more information about the MENU OCL statement and the MENU command.

When you chain menus, you should allow the user to display the main menu again. You can do so with an option on the menus that are displayed after the main menu, or you can remind the user that command key 3 causes the previous menu (in this example the main menu) to be displayed. For example, on the Orders Menu the user can return to the main order entry and invoicing menu (ORDENT) by pressing command key 3 or the Enter key. Also, you should allow ways for experienced users to bypass the menu chains and directly begin their jobs, perhaps by entering a procedure from the menu.

Creating and Updating Menus

From the programmer's viewpoint, a menu is made up of two library members: the **menu text member** and the **command text member**:

- The menu text member tells the *user* what each option number will do. The menu text member, a display format, describes what will appear when the menu is displayed. The menu text member includes any descriptive text associated with an option number, the placement of the option numbers, and the name and title of the menu.

- The command text member tells the *system* what to do for a selected option number. The command text member, a second-level message member, describes what procedures, commands, or statements will be used to run a job when the user selects an option number.

The system provides two ways for you to create and update your menus:

- Screen design aid (SDA) utility

- BLDMENU procedure

Using SDA to Create a Menu

The **screen design aid (SDA)**, which is part of the optional Utilities Program Product, can lead you through the steps used to create and update a menu. For this reason, you may find that SDA is considerably easier to use than the BLDMENU procedure.

SDA has several advantages over using the BLDMENU procedure to create a menu:

- You can design your menu at the display station. This allows you to see immediately how your menu will look when it is displayed.

- SDA automatically creates the source and the load members required to display a menu. SDA does most of the work for you; you need only supply the menu text and the command text for the menu.

- You can also use SDA to create and update **help text** for each menu option or for the entire menu. For more information about menu help text, see "Creating and Displaying Help Text for Menus" on page 12-12.

The *Creating Displays* manual describes how to use SDA to create and update menus and menu help text.

Using the BLDMENU Procedure to Create a Menu

The BLDMENU procedure, which runs the $BMENU utility program, is part of the SSP. If you choose to use the BLDMENU procedure instead of SDA, you must first determine the design of your menu, perhaps on paper or on a preprinted form. Having designed the menu, you then use the **source entry utility (SEU)** or the $MAINT utility program to enter and create the menu text and command text source members. And finally, you run the BLDMENU procedure to convert the source members into the load members used by the system.

Although SDA is easier to use than the BLDMENU procedure, you might find that you can use SEU or DSU to make minor changes to your menu source members and then run the BLDMENU procedure to compile them faster than if you use SDA to do the same. You may want to use both SDA and the BLDMENU procedure, and then decide which works better for you.

The *Creating Displays* manual describes how to use the BLDMENU procedure to create and update menus.

Creating and Displaying Help Text for Menus

Although each menu option has a description of the job that will be run when the user chooses a particular menu option, a simple job description might not be enough. For example, because a menu is the entry point to the application, you might also want to supply the following kinds of information:

- A summary of the application itself, such as an explanation of when a particular menu option or job should be selected

- A description of the input forms necessary to perform the job

- The work the user will be expected to do while the job is running

- An estimate of how long a job takes to run

- A description of the output produced by the job, and an explanation of what is to be done with that output

You can supply this kind of additional information by using SDA to create **menu help text**.

Creating Menu Help Text

SDA allows you to create menu help text. After you create your menu, you can create help text for the entire menu, for a single option on that menu, or for a range of menu options. For example, if a menu has six menu options, you could create a display of summary help text for the entire menu and two displays of help text describing the menu options in more detail: one display describing options 1 though 3, and the other describing options 4 through 6. After you specify the range of option numbers for which you want to create menu help text, SDA shows you a display on which you can design the help text.

After you create the help text for your menu, you can update it just as you would the menu it describes. Just specify the range of option numbers the help text addresses, and SDA shows you the help text. You can then change the help text.

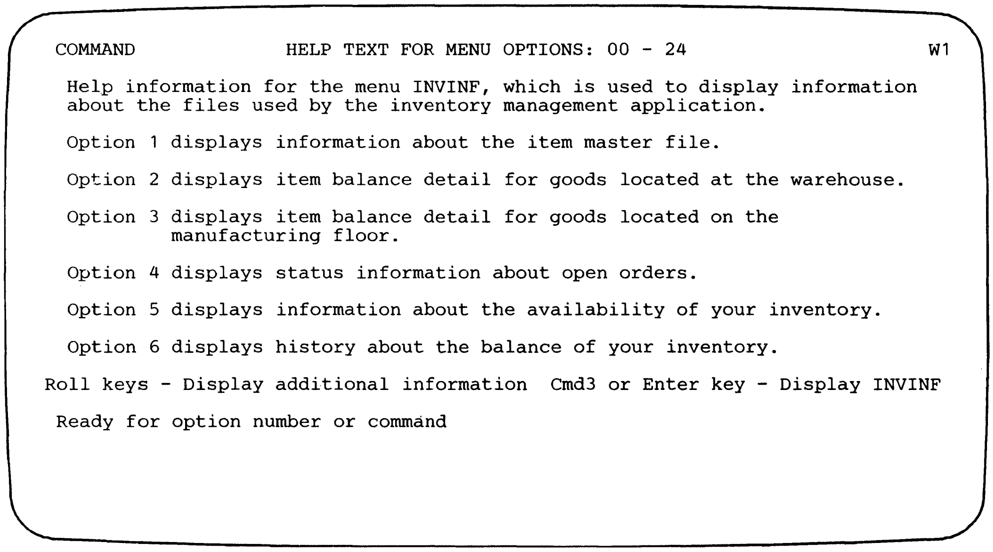
The *Creating Displays* manual has more information about creating and updating menu help text.

Displaying Menu Help Text

The user can display menu help text by doing either of the following:

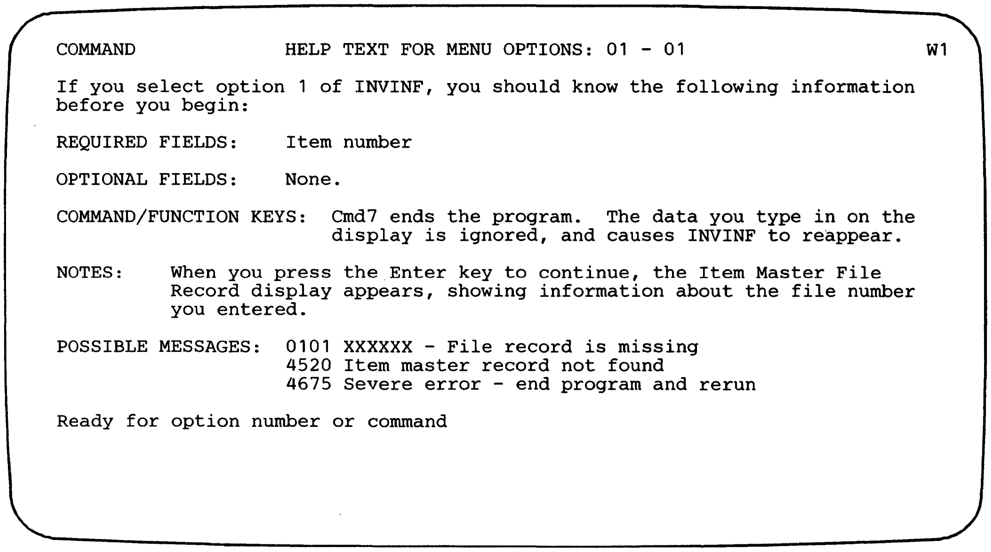
- Pressing the Help key when the menu is displayed
- Typing in an option number on that menu and pressing the Help key

If the user presses the Help key without typing in an option number, help text for the entire menu is displayed. For example, if the user is viewing the menu for jobs that display information about the files used by the inventory management application, the user can press the Help key, and the menu help text summarizing the jobs run by the menu is displayed:



```
 COMMAND             HELP TEXT FOR MENU OPTIONS: 00 - 24                 W1

  Help information for the menu INVINF, which is used to display information
  about the files used by the inventory management application.

  Option 1 displays information about the item master file.

  Option 2 displays item balance detail for goods located at the warehouse.

  Option 3 displays item balance detail for goods located on the
           manufacturing floor.

  Option 4 displays status information about open orders.

  Option 5 displays information about the availability of your inventory.

  Option 6 displays history about the balance of your inventory.

Roll keys - Display additional information  Cmd3 or Enter key - Display INVINF

 Ready for option number or command
```

If the user types in an option number and presses the Help key, more detailed menu help text about that particular menu option is displayed. For example, if the user selects option 1 on the inventory management application menu and presses the Help key, the following would be displayed:



```
COMMAND             HELP TEXT FOR MENU OPTIONS: 01 - 01                  W1

If you select option 1 of INVINF, you should know the following information
before you begin:

REQUIRED FIELDS:    Item number

OPTIONAL FIELDS:    None.

COMMAND/FUNCTION KEYS:  Cmd7 ends the program.  The data you type in on the
                        display is ignored, and causes INVINF to reappear.

NOTES:     When you press the Enter key to continue, the Item Master File
           Record display appears, showing information about the file number
           you entered.

POSSIBLE MESSAGES:  0101 XXXXXX - File record is missing
                    4520 Item master record not found
                    4675 Severe error - end program and rerun

Ready for option number or command
```

In either case, once the help text is displayed, the user can do the same things that were allowed on the menu that was originally displayed. The user can, for example, select an option number or enter a procedure or command. The only exception is that once menu help text is displayed, the user cannot use the Help key to obtain more help information. However, if the user is viewing a portion of the help text supplied for a menu, the rest of the help text can be displayed simply by pressing the Roll Up (Roll↑) or Roll Down (Roll↓) key. The user can return to the original menu by pressing command key 3.

Using Color or Highlighting on Your Menus

Any menu that you design and create for a noncolor display station can also be used for a color display station. If you have a color display station, take advantage of the colors you can use on your menu.

Used properly, color is more pleasing to the eye than noncolor data, and it generally makes the display more interesting. Color can be used to draw attention to menu options that are more significant than others.

Some display stations, depending on the model, can display the following colors:

Green
Red
White
Blue
Turquoise
Pink
Yellow

If you use SDA to create your menus, the menu text member is stored as a display format source member. You can use SDA's **design display format** option to add color to the menu text member (you cannot use the SDA **design menu** option to select color for a menu). Use the SDA procedure and select the option that allows you to create and update display formats. From the SDA Field Attributes display, you can press command key 8; the Color Attributes for Field display appears. You can then select a color for the field that you are defining.

Note: When you use the display format option to update a menu, be careful to update ***only*** *the fields that describe the menu options and change* ***only*** *the field attributes that control color. Do not change the input or output data characteristics for the fields on the menu. Do not add, move, or delete fields on the menu, and do not change the lengths of the fields on the menu. If you do not take these precautions, unpredictable results may occur.*

Once you have used the SDA display format option to add color or highlighting to your menu, do not use the SDA menu option to update your menu; instead, use the display format options of SDA. If you update your menu using the SDA menu option, the color or highlighting attributes will be lost.

The *Creating Displays* manual has information about using SDA to create and update display formats.

Even if you do not have color display stations, you can still use the method just described to highlight fields on your menus. You can display a particular field in reverse image, in high intensity, or with column separators, blink or underline the field, or display the field with a combination of these field attributes. The advantage of highlighting on a noncolor display is exactly the same as using color on a color display: the menu becomes more pleasing, more interesting, and easier to use.

Chapter 13. Displays

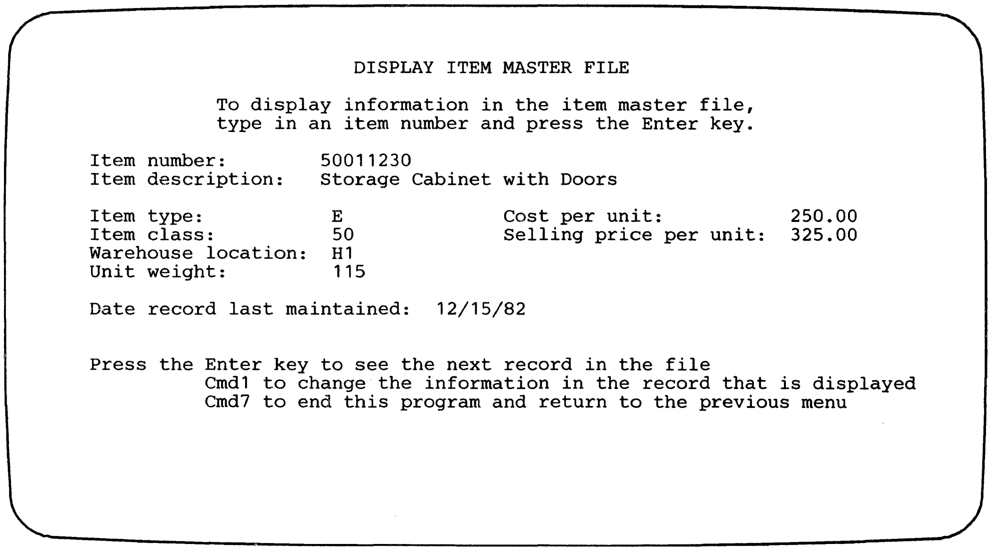
The purpose of this chapter is to:

- Explain the basic concepts of displays and display formats
- Describe how you can design and create display formats
- Describe how display formats are used by procedures and programs

An application user uses the display station to communicate with the system and to run application programs. Sometimes the user enters data on the display for the system or a program to use, and at other times the system or a program uses the display screen to show information needed by the application user.

The information shown on the display screen is called a **display**. That information and its use is defined in a **display format**. Using the display defined in a display format, a program can prompt the application user to enter information. That program can also use the display to show requested information to the application user and to explain what actions are to be taken.

The following is a sample display:



```
                      DISPLAY ITEM MASTER FILE

           To display information in the item master file,
           type in an item number and press the Enter key.

    Item number:        50011230
    Item description:   Storage Cabinet with Doors

    Item type:          E               Cost per unit:            250.00
    Item class:         50              Selling price per unit:   325.00
    Warehouse location: H1
    Unit weight:        115

    Date record last maintained:  12/15/82


    Press the Enter key to see the next record in the file
              Cmd1 to change the information in the record that is displayed
              Cmd7 to end this program and return to the previous menu
```

This sample display contains 24 lines of 80 characters each. For a 3180 Display Station, the display may contain 27 lines of 132 characters each.

Benefits of Displays and Display Formats

Displays are useful for the following reasons:

- They make it easier for the application user to enter and get data.
- They improve productivity.
- They are defined separately from the program or procedure that uses them.

Using a display, the user can easily recognize what information is required. Because display formats are separate from the procedure or program that uses them, one set of display formats can be used by different procedures or programs. And because display formats are defined separately, they usually can be changed without recompiling your programs.

The following describes how your programs and the system use displays and display formats.

Work Station Data Management

Programs that communicate with display stations are called **interactive programs**. Interactive programs use a system function called **work station data management** to write data to and read data from a display station. Work station data management communicates with the interactive programs by using display formats to get data to the program and to write data from the program on the display screen.

Work Station Data Management Operations

Work station data management uses an area in main storage called the **work station buffer** as work space for operations involved with reading or writing display formats.

The size of the work station buffer changes based upon the number of input or output requests to the display station and the size of the display format being written to or read from the display station. The system automatically maintains the size of the work station buffer.

A program must be able to do the following basic operations in order to use display formats:

- The program must specify the load member or members containing the display formats to be used. The display format load member can be thought of as a type of file. Like any other file, the file associated with the display format load member must be defined and opened. The file is identified and described by some specification, statement, or subroutine.

- The program must be able to select the specific display formats to be used and must be able to send data to the display. That data is the output needed by the user running the program. The sending of data to the display is often called an **output** operation, or a write or put operation.

- The program must be able to accept data entered on the display. That data is the input needed by the program. The accepting of data entered by the user is often called an **input** operation, or a read, invite, or get operation.

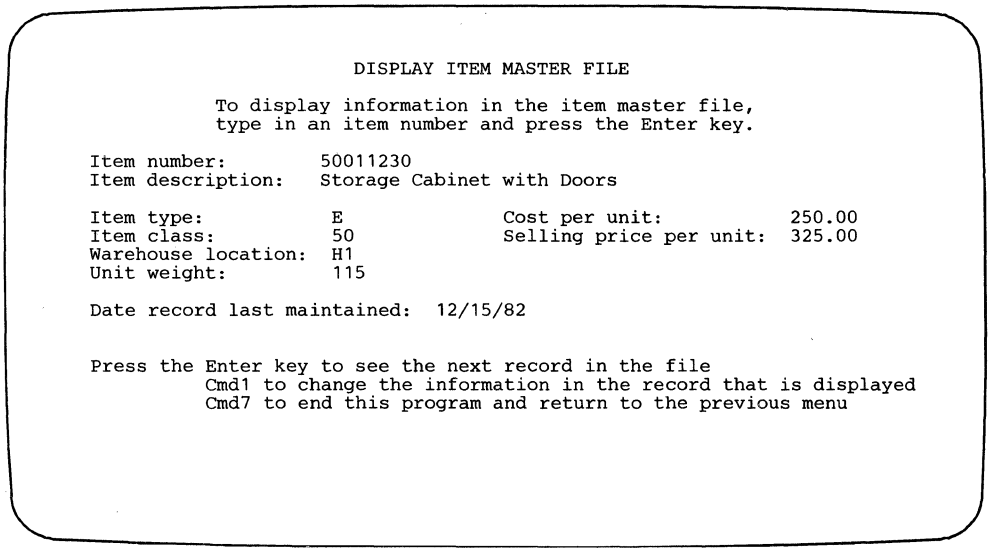
In addition to the operations listed above, the program must be able to control other functions such as indicators and the command and function keys.

Output Operations and Output Fields

For output operations, work station data management prepares a data stream to be transmitted to the display station by merging data supplied by the program with the display format. This information is placed in the work station buffer, and work station data management sends the contents of the buffer to the display station.

Output fields contain information that the application user cannot change on the display. The contents of output fields are not returned to the program. Output fields can be fields containing data supplied by the program, or they can be **prompts** or **constants**, defined by the display format. A prompt is a request for information or action from the application user. A prompt can tell the user what type of information is to be entered or displayed, the form in which that information is to be entered or displayed, and the options or values that are allowed as input for that data field.

The following shows examples of output fields. In this example, all the output fields are prompts defined by the display format.



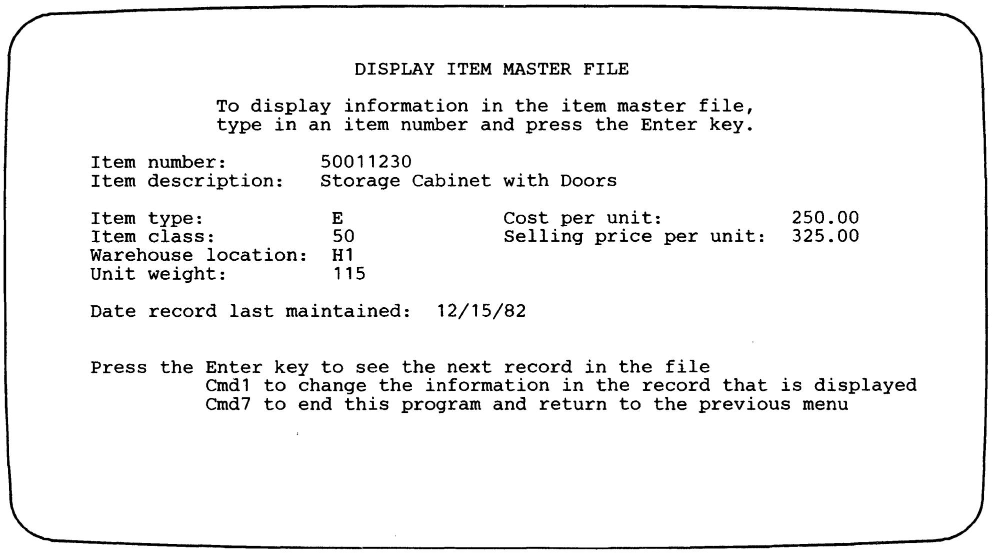
```
                        DISPLAY ITEM MASTER FILE

           To display information in the item master file,
           type in an item number and press the Enter key.

Item number:        50011230
Item description:   Storage Cabinet with Doors

Item type:          E               Cost per unit:           250.00
Item class:         50              Selling price per unit:  325.00
Warehouse location: H1
Unit weight:        115

Date record last maintained:  12/15/82


Press the Enter key to see the next record in the file
          Cmd1 to change the information in the record that is displayed
          Cmd7 to end this program and return to the previous menu
```

Input Operations and Input Fields

For input operations, the data entered into the format is placed into the work station buffer when the operator presses the Enter key. When a program does a read operation from the display station, work station data management moves data from the buffer to the user's record area.

Input fields are fields in which the application user can enter data on the display. When a program shows a display, input fields are usually blank (although in some cases, a default value may already be supplied). The user can type data into the input field; its contents are sent to the program when the application user presses the Enter key. This data is then used by the program to do an operation, such as a calculation or a file update.

The following shows an example of an input field.



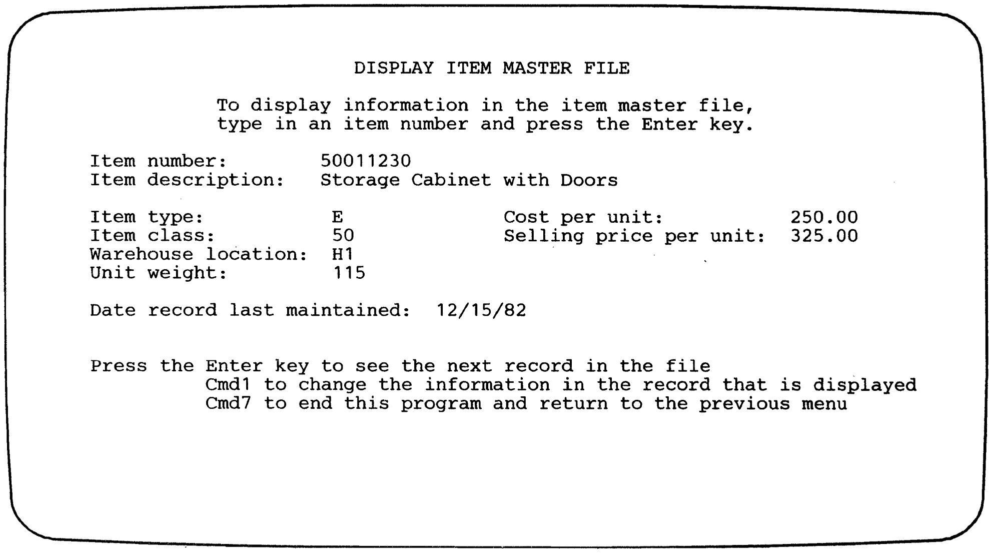
```
                       DISPLAY ITEM MASTER FILE

           To display information in the item master file,
           type in an item number and press the Enter key.

Item number:        50011230
Item description:   Storage Cabinet with Doors

Item type:           E              Cost per unit:          250.00
Item class:          50             Selling price per unit: 325.00
Warehouse location:  H1
Unit weight:         115

Date record last maintained:  12/15/82


Press the Enter key to see the next record in the file
          Cmd1 to change the information in the record that is displayed
          Cmd7 to end this program and return to the previous menu
```

In this example, the user typed an item number, 50011230, in the input field. When the user pressed the Enter key, the contents of the input field were sent to the program.

Input/Output Fields

Input/output fields allow data to be entered by the user, and they allow data to be displayed to the user. The user can type in new data in a blank input/output field, or the user can change data that might already be displayed in an input/output field. Data displayed to the application user can be supplied by the program or specified by the display format itself. Data contained in an input/output field is returned to the program when the user enters the display.

The following shows examples of input/output fields. In this example, the program supplied information about the requested item number. The user can accept that information or change it. The user can change the information in any of the input/output fields, and then press command key 1 to enter those changes.



```
                        DISPLAY ITEM MASTER FILE

           To display information in the item master file,
           type in an item number and press the Enter key.

Item number:        50011230
Item description:   Storage Cabinet with Doors

Item type:           E               Cost per unit:           250.00
Item class:          50              Selling price per unit:  325.00
Warehouse location:  H1
Unit weight:         115

Date record last maintained:  12/15/82


Press the Enter key to see the next record in the file
          Cmd1 to change the information in the record that is displayed
          Cmd7 to end this program and return to the previous menu
```

Data Types

If you define a particular field as an input or input/output field, you must also define the type of data that can be entered in it. For example, you can specify that an input field is to accept numeric data only; that is, the field accepts only the digits 0 through 9, commas, decimal points, plus signs and minus signs. This definition is useful if a field requires the entry of such information as inventory amounts or account balances.

You can specify that a field is to accept alphameric data only: the characters A through Z, special characters, and any numeric data. This definition is useful for a field that requires both alphabetic characters and numeric digits, such as a customer's address.

Other data type definitions include:

- Alphabetic data. Only the letters A through Z, and certain special characters, are allowed.

- Digits only. Only the numbers 0 through 9 are allowed.

- Signed numeric. Only the numbers 0 through 9 are allowed; the sign is determined by which key is pressed, Field+ or Field-, after the numbers are entered.

- Magnetic stripe reader. This field contains data to be read from the magnetic stripe reader.

- Right-to-left field. If the appropriate national language PRPQs are installed, the cursor moves from right to left within this field as the application user types in data.

- Katakana data. This field can contain Katakana characters.

- Ideographic data. For the ideographic version of the SSP, ideographic characters can be entered and displayed.

- Numeric shift fields. On data entry keyboards, the keyboard will be automatically shifted to numeric shift when the cursor is in this field. On keyboards that are not data entry keyboards, the field defaults to an alphameric field.

Attributes

Besides defining the field and the data type, you also define the physical characteristics, or **attributes**, of a field. The attributes that you can specify include:

- High intensity. Data shown in high intensity is brighter than that shown in normal intensity. In effect, your data looks as if it is displayed in **boldface**. High intensity is useful for drawing attention to important information, such as display titles or column headings.

- Blink field. If a field is a blink field, data displayed in that field blinks. A blinking field is easy to see and can be used to draw attention to important information. Blinking fields, however, are difficult to read and might annoy the application user if used too often.

- Nondisplay. If data is typed in or sent to a nondisplay field, it does not appear on the display. A nondisplay field is useful for information needed by the program but not by the user, such as a display or record ID, or for information that must remain confidential, such as a password or security code.

- Reverse image. Normally, data on the display appears as light characters on a dark background. If a field has the reverse image attribute, the data in that field appears as dark characters on a light background. Reverse image is a good way to show the user the location of a field or to draw attention to an error message or to fields in which the user has entered incorrect data.

- Underline. An underlined field can be used to emphasize information, like this , or it can be used to show the length of an input field, like this: _______________ . Showing the application user the length of an input field is important, because a keyboard error message results if the user attempts to type data outside the input field.

- Column separators. Column separators are useful for showing the number of positions in an input field. Column separators appear as dots or vertical lines (depending on the type of display station on which the display is shown) on either side of each character position within the field. Column separators do not require character positions of their own. An input field with 5 character positions, for example, would look like this:

  i.n.p.u.t.

Field attributes are allowed in combinations. For example, you can specify that a field be displayed in high intensity and reverse image. Also, you can specify that a field attribute always be used, or that the field attribute is controlled by the program. A special kind of switch called an **indicator** can be used by the program to turn an attribute on or off.

Special Work Station Data Management Operations

When you define a display format, you can specify the following special operations not normally performed by work station data management:

- Erase input fields
- Override fields
- Suppress input

These special operations can be useful to you in improving the performance of your display stations, display formats, and programs. These operations can be performed each time a display format is used, or they can be performed when needed by setting on an indicator.

Erase Input Fields Operation

For an erase input fields operation, work station data management blanks out the contents of unprotected input and input/output fields on the display. Work station data management then invites input from the display. The display format is read from disk but is not sent to the display station on an erase input fields operation. This operation sends only the control characters required to remove the contents of the input fields rather than sending all the data required to display the entire format again.

You might want to request the erase input fields operation when an application using remote display stations requires an application user to enter information in the same fields time after time. In such an application, you should specify an indicator to control the erase input fields operation. The first time the program displays the format, that indicator should be off. The program should then turn on the indicator for the next and succeeding times it issues the display. Each time the display is issued with the indicator on, the input fields are blanked out, and the user can again enter data in them. This operation can be important when a program communicates with a remote display station because the amount of information transmitted to a remote display station directly affects the performance of the jobs using the communications line.

See "Designing Displays for Remote Display Stations" on page 13-19 for remote display station considerations.

Override Fields Operation

For an override fields operation, work station data management:

- Transmits the contents of conditional output fields (output fields for which an indicator is specified for the output data attribute) if an output indicator is on.

- Retransmits only the attribute bytes for all field attributes controlled by indicators; except for the protect field attribute, which will be retransmitted.

The override fields operation allows the program to override (modify) output fields on a display without retransmitting the entire display. This operation can be valuable when a program communicates with a remote display station, in that it reduces the amount of information transmitted over the communications line.

See "Designing Displays for Remote Display Stations" on page 13-19 for remote display station considerations.

Suppress Input Operation

For a suppress input operation, work station data management does *not* invite input from the display station after transmitting the format to the display station. The application user can enter information into input fields on the display if the keyboard is not locked, but this information is not sent to the program until the program requests work station data management to read input from the display station.

This operation is usually used with display formats that contain only output fields. If multiple display formats are displayed before input is returned to the program, the suppress input operation should be used. When multiple formats are sent, the suppress input operation should be specified on all but the last format displayed.

See "Designing Displays for Remote Display Stations" on page 13-19 for remote display station considerations.

Display Design Considerations

When you design displays, you are concerned primarily with the way data is displayed to the user and with the way the user responds to this data. Input displays should be designed for ease of data entry, and output displays should be designed for ease of reading. Displays that show output and also allow input are the most challenging to design because you must try to make a balance between ease of data entry and ease of reading.

Displays must be clear, complete, and understandable. A well-organized and descriptive display can help to improve user productivity. When you design a display, you must consider:

- How the display will be used
- What kind of information you want the display to process
- The source documents used as input for the display
- The level of experience or responsibility of the application user who uses the format
- The size of the display (24 lines of 80 characters, or 27 lines of 132 characters for the 3180 Display Station)

General Display Design Guidelines

When you design your display, remember to:

- Make the user feel productive
- Identify the displays and provide meaningful headings
- Design displays that are easy to read
- Display a small amount of information at one time or provide one idea for each display
- Maintain consistencies among displays
- Keep user responses short
- Respond to user input
- Make error correction easy
- Provide help information
- Document your displays

The *Creating Displays: Screen Design Aid and System Support Program* manual has more detailed information these design guidelines.

Additional Display Design Guidelines

Choosing the Appropriate Form for Your Display

Displays should be designed so that they suit your particular application. The following are general types of displays:

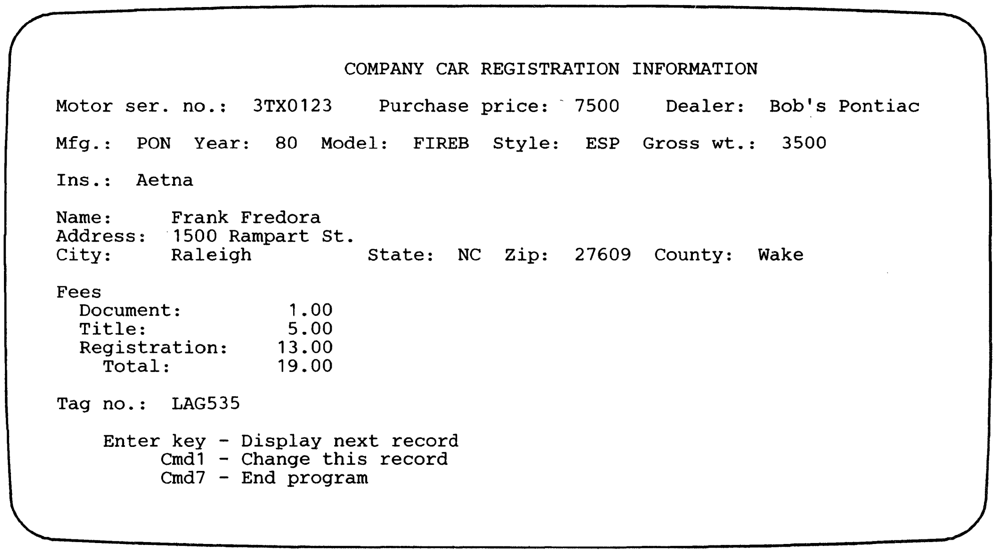
- Fixed form
- Adjacent form
- Free form
- Menu form
- Code link form

You should choose the form that works best for your particular application.

Fixed Form Displays

Using a fixed form display, the user supplies input in response to prompts on the display. In fact, a fixed form display is usually designed to resemble the source documents that contain the data to be entered into the system. Prompts and input fields on a fixed form display are arranged the same way the corresponding fields are arranged on the source document. Because of this similarity, the inexperienced user can easily master fixed form displays.

The following is an example of a fixed form display:

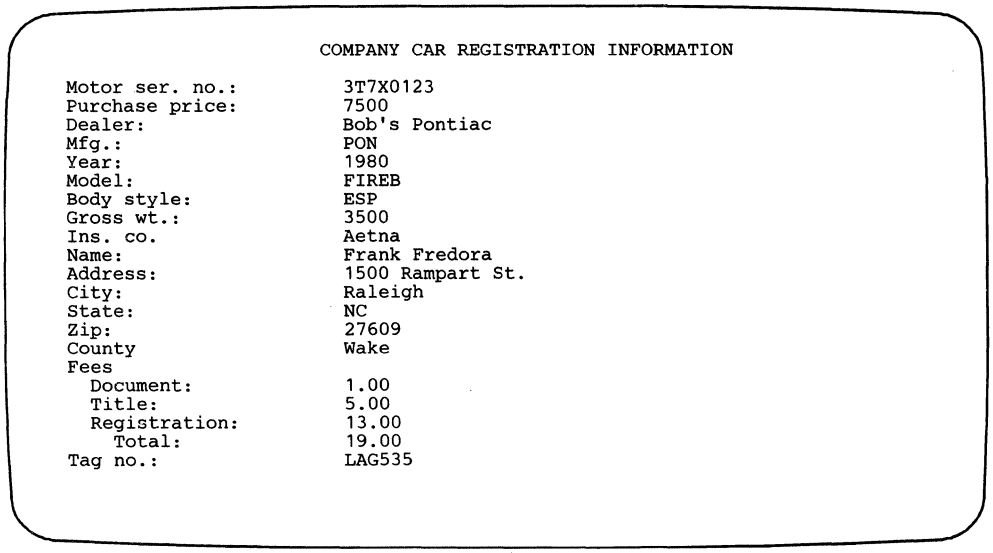


```
                          COMPANY CAR REGISTRATION INFORMATION

Motor ser. no.:  3TX0123    Purchase price:  7500    Dealer:  Bob's Pontiac

Mfg.:  PON  Year:  80  Model:  FIREB  Style:  ESP  Gross wt.:  3500

Ins.:  Aetna

Name:     Frank Fredora
Address:  1500 Rampart St.
City:     Raleigh          State:  NC  Zip:  27609  County:  Wake

Fees
  Document:         1.00
  Title:            5.00
  Registration:    13.00
    Total:         19.00

Tag no.:  LAG535

    Enter key - Display next record
         Cmd1 - Change this record
         Cmd7 - End program
```

Adjacent Form Displays

In an adjacent form display, data is arranged and entered in columns. The first column contains prompts for the kind of information to be entered or displayed, and the second column contains the fields that are to receive the input data or show the output data. A third column could be used to allow the user to change information displayed in the second column. Adjacent form displays are easy to read at a glance, and are good for the occasional user.

The following is an example of an adjacent form display:



```
                      COMPANY CAR REGISTRATION INFORMATION

Motor ser. no.:         3T7X0123
Purchase price:         7500
Dealer:                 Bob's Pontiac
Mfg.:                   PON
Year:                   1980
Model:                  FIREB
Body style:             ESP
Gross wt.:              3500
Ins. co.                Aetna
Name:                   Frank Fredora
Address:                1500 Rampart St.
City:                   Raleigh
State:                  NC
Zip:                    27609
County                  Wake
Fees
  Document:             1.00
  Title:                5.00
  Registration:         13.00
     Total:             19.00
Tag no.:                LAG535
```

Free Form Displays

On a free form display, data is entered into long unprotected fields. Data is typed in a string of characters; individual fields or records are separated by a special predetermined character. For example, a customer's name and address could be entered as follows, the separating character being a semicolon (;):

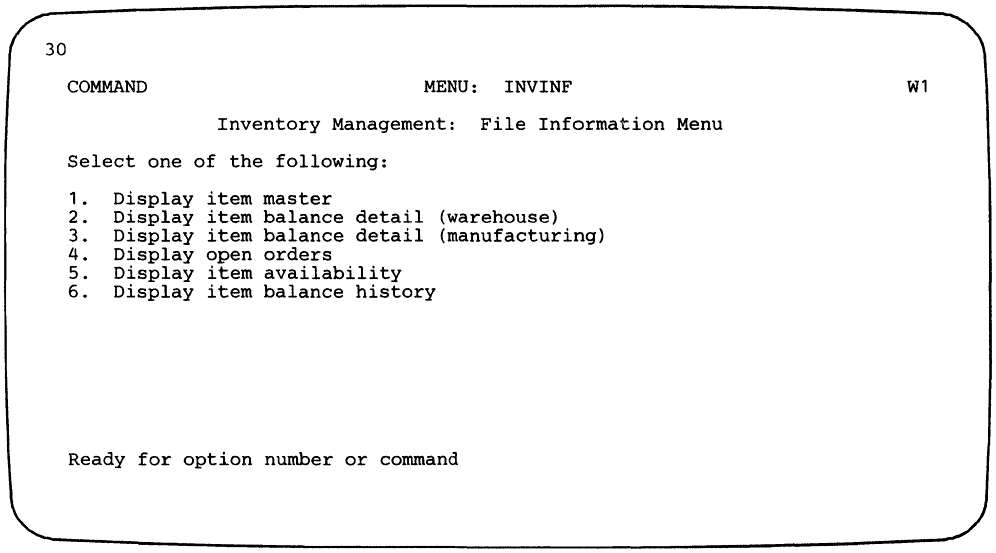
```
last name;first name;initial;address;city;state;zip code
```

Free form displays are especially useful for more experienced users and all rapid data entry. One line of constant header information is usually all that is needed to aid the user in remembering the position of individual fields or records.

Your program must be able to interpret the format of that data; special input array processing might be required.

Menu Form Displays

One of the first displays that an application user sees within an application is a menu used to select a particular job. Menus are generally easy to use, and their format can be applied to the displays that your programs use. Many applications have several predefined transactions for which the same kind of data or actions are used time after time. To eliminate unnecessary keying of data, you can use a menu form display on which the user simply types in a selection from a displayed list of options. Based on that selection, data is appropriately displayed or processed by your program.

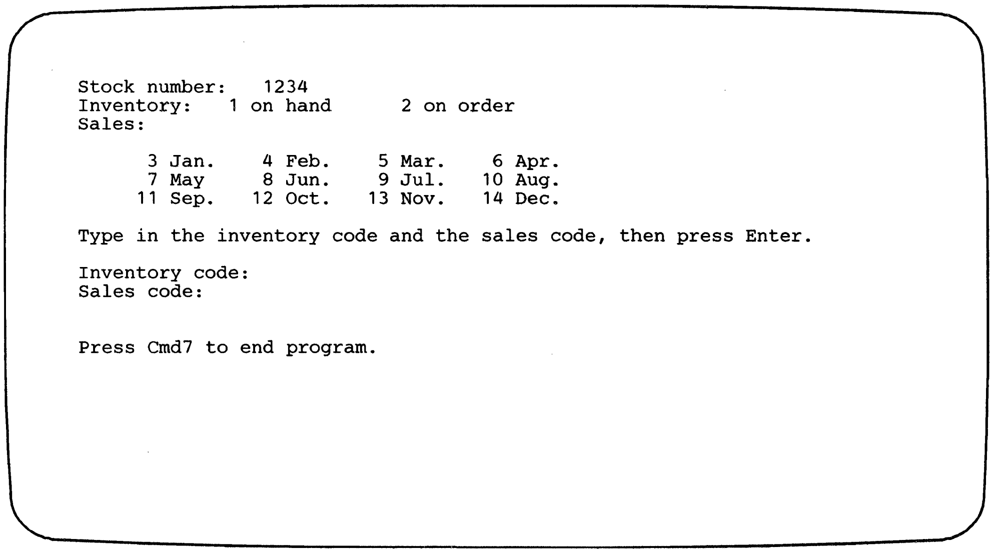


```
30

 COMMAND                         MENU:  INVINF                           W1

              Inventory Management:  File Information Menu

 Select one of the following:

 1.  Display item master
 2.  Display item balance detail (warehouse)
 3.  Display item balance detail (manufacturing)
 4.  Display open orders
 5.  Display item availability
 6.  Display item balance history






 Ready for option number or command
```

Code Link Form Displays

The code link form display is an extension of both the free form and menu form displays. In a code link form display, a code number is selected from the menu and a value associated with that selection is entered in free form. Based on the selected code number, the program must interpret and process the entered data (input array processing might be required).

The following is a sample code link form display:



```
Stock number:   1234
Inventory:   1 on hand       2 on order
Sales:

      3 Jan.    4 Feb.    5 Mar.    6 Apr.
      7 May     8 Jun.    9 Jul.   10 Aug.
     11 Sep.   12 Oct.   13 Nov.   14 Dec.

Type in the inventory code and the sales code, then press Enter.

Inventory code:
Sales code:


Press Cmd7 to end program.
```

Using Color to Highlight Data

Any display that you design for a noncolor display station can also be used for a color display station. If you have a color display station, take advantage of the colors you can control using the display format.

Used properly, color is more pleasing to the eye than noncolor data, and it generally makes the display more interesting. Color can be used to draw attention to fields that need user attention, such as a request for user input or a response to an error condition.

Some display stations, depending on the model, can display the following colors:

Green
Red
White
Blue
Turquoise
Pink
Yellow

If you use SDA to create and update displays, you can select colors for each field from the Color Attributes for Field display. The *Creating Displays* manual has a description of how to select color attributes.

When you design a display that uses color, consider the following recommendations:

- Determine whether the display format will be used on both color and noncolor displays. If you are designing displays that will be shown at both color and noncolor display stations, use the Limit Color Select option to preview the finished display. The Limit Color Select option, a special keying sequence, reduces the number of colors displayed to two. This preview lets you see how a display designed for color will look when it is displayed at a noncolor display station.

- Use each color for a particular purpose. When you choose a color, define what it means, and use it in the same way on every display. For example, if white is used to highlight important output fields or error messages, use white for that purpose throughout the displays that you design.

- Use a limited number of colors. Too many colors on a display can confuse the user. The fewer colors used, the more effective each color becomes.

- Group colors. If colors are grouped in a recognizable and consistent manner, the user can easily organize and follow information. Too many colors spread over a display can hide the meaning of the display.

The *Display Station Programmer's Guide to Using Color* has more information about color display stations if you have a color display station.

Designing Multiple Formats

Information from several display formats can appear on the display at one time. This capability is useful if some information on the display is to remain unchanged while other information is to be replaced. If you use multiple formats, the following are true:

- Response time is improved because no unnecessary information is sent to the display station.
- Less coding is required because the specifications for each format are generally simpler. This reduces the amount of coding.
- Because fewer specifications are required for each format, less disk space is needed and data is not duplicated.

When using multiple formats, be careful not to clear or replace any information that should remain on the display.

When multiple formats are displayed before information is read from the display, the system reads only those input fields from the last format displayed with input fields. This means that when all or a portion of a display format is replaced by a new display format with input fields, the data contained in all previous input fields cannot be read, if the input fields on the new display format are at different locations on the display screen.

You should take certain precautions to avoid a display station error and to reduce system workload when multiple formats are displayed. If the formats do not define any input fields, the suppress input operation should be specified for *all* but the last display format. In addition, the last format *must* define at least one input field if *all* of the following are true:

- A value other than 24, 27, or blanks is specified as the number of lines to clear.
- The format replaces or clears a line that contains an input field created by a previous format displayed with input fields defined.
- Suppress input is not specified (the user can enter data on the display).

The program will be suspended if **all** of the following are true:

- An RPG II, COBOL, or BASIC MRT program displays multiple formats.
- The suppress input operation is specified on any format prior to the last format.
- The user interrupts the program by pressing the Attn key before the last format is displayed.

No other display stations can use or start the program during the inquiry request. The system does not inform application users at other display stations that the program has been suspended.

Designing Displays for Remote Display Stations

Program performance can vary depending upon whether a format is displayed at a local display station or at a remote display station. Remote display stations communicate with the system at a slower rate than do local display stations; consequently, response time at remote display stations is increased. Because of the slower transmission rate, remote display stations can also tie up overall system activity. If remote display stations cause poor performance, using one of the following techniques might significantly improve performance:

- Reduce the amount of data that is transmitted over the communications line.

- Increase the line speed.

Of these techniques, reducing the amount of data transmitted is the only one that can be directly controlled by programming techniques. You can reduce the amount of data transmitted by following these suggestions when coding your display formats:

- Send only the data that the user needs to efficiently use the application. Consider creating help display formats that can be used to request additional information.

- Avoid displaying the same data and prompts again.

- Specify N (no) for the return input operation. This reduces the amount of data transmitted over the communications line, and also reduces the response time.

  If you have an identification field on a display format that is being read by a program, you should specify Y (yes) for return input operation.

- Use an erase input fields operation to remove the contents of an input field rather than displaying the entire format again.

- Avoid using the read-under-format technique (described later in this chapter) to pass information from one job step to another. Instead, consider using a temporary disk file, the local data area, or data structures or arrays to pass the information.

- Use an override fields operation to display error messages. Display the error message and only the fields in error. This technique avoids transmitting unnecessary data when errors occur.

- In the display format specifications, define the fields in the same left-to-right, top-to-bottom order that they will appear on the display. (Additional control characters must be transmitted for any out-of-sequence fields, thus increasing the response time.)

- If you do use multiple formats, specify the suppress input operation on all but the last format. Specifying suppress input reduces the turnaround time before each new format is displayed. (Turnaround time is the time between receiving and transmitting data or between transmitting and receiving data.) If input is not suppressed, several line turnarounds are required after each format is displayed.

When designing and coding display format for remote display stations, you should be careful to use the proper coding techniques. The following table lists the approximate number of bytes that are transmitted when a display format is sent to and received from a display station.

Item	Bytes Required
Each display format	4 bytes minimum + 4 bytes if the format clears a portion of the previous format + 3 bytes for each field for which the position cursor attribute is specified + 5 bytes if any input or input/output fields are contained in the format
Each input field	8 bytes minimum + 2 bytes if the field is out of sequence + 2 bytes if the field is self-checking
Each input/output field	8 bytes minimum + 2 bytes if the field is out of sequence + 2 bytes if the field is self-checking + The length of the output data
Each output field	4 bytes minimum + The length of the output data

As you can see, it is to your advantage to avoid the unnecessary transmission of data. In a typical application with five display formats in which input data is read and output data is displayed, the total characters transmitted over the communications line might be, for example, 2700 characters. With the proper use of the return input, erase input, override fields, and suppress input operations, the total characters transmitted over the line can be reduced to approximately 2100, a reduction of over 20%. The response time is better, and more important, the rate at which an application user can perform the work is improved.

Using Message Members with Your Display Formats

Display formats allow you to use message members to display constant information. Message members are especially useful for fields that show error messages or conditional instructions. For example, if a program detects an input error, the program can select the appropriate error message from a message member and display that message. The program only has to specify the message identification code (or MIC number) for the particular message to be displayed; the program or the display format does not have to define the text of the error message. With this capability, you can define one message member for your application and have all the programs within the application use the same messages in that member. This provides consistency within the application and frees you from the coding of message text within your programs or display formats.

The message to be displayed can be identified by the display format itself, or it can be specified by the program using the display format. If a MIC number and a message member identifier are specified for an output field in your display format, the corresponding message is displayed in the output field. If a MIC number and a message member identifier are not specified for the output field, the program supplies that information in the output record area. Then, when a write operation is sent to the display format, the message corresponding to the supplied MIC number and message member identifier is displayed in the output field.

For more information about creating and using message members, see Chapter 14, "Messages and Message Members." The *Creating Displays* manual has more information about the specific entries made in the display format for displaying messages from a message member.

Using Self-Check Digits

Self-checking digits can provide some protection against data entry errors and fraud. Self-checking provides a method of verifying the contents of an input field at the same time it is entered. This method is especially useful if an application requires the entry of numeric data, such as account numbers. Unless the person attempting the fraud knows the system, the chance of an invented number matching an allowed number is reduced.

Note: You must have the **extended function feature** *installed on remote 5251 or 5294 display stations to use self-check digits. If this feature is not installed, a program error will result when self-check is specified and the operator tries to leave the field.*

The system offers two methods of self-checking: modulus 10 and modulus 11. If a self-checking method is specified for an input field, the system determines a self-check digit for the field's contents using the specified self-check method. That self-check digit is compared to the rightmost position of the input field. If the self-check digit matches the rightmost position of the input field, the contents of the input field are allowed, and the user can continue. If those numbers do not match, the contents of the input field are not allowed, and a keyboard error is displayed. The user must then enter an allowed number before continuing.

The *Creating Displays* manual gives more detailed information about how the system checks the contents of an input field according to the appropriate self-check method. After reviewing how the self-check digit is determined, you might want to write a program that generates input numbers that successfully complete a self-check. You can, for example, use the generated numbers as a basis for assigning account numbers, item numbers, or security codes.

Programming Considerations

The following pages summarize:

- How you can create display formats using SDA or the FORMAT procedure
- How you can create help text for your displays
- How the programming languages use display formats

Creating Display Formats

After you design your display, you must create the library member the system uses to show the display. Every display is defined by a **display format**, which is stored in a **display format member**.

The display format member can contain up to 255 display formats. Each display format is made up of **specifications**. These specifications define information about:

- The entire display. This information is defined in the **display control specification** (or S specification).
- Individual fields on the display. This information is defined in the **field definition specifications** (or D specifications).
- Optionally, help text that is available for the display. This information is defined in the **help definition specifications** (or H specifications).

The system provides two ways for you to create your display formats:

- Screen design aid (SDA) utility
- FORMAT procedure (SSP)

Using SDA to Create a Display Format

The screen design aid (SDA), which is part of the Utilities Program Product, can lead you through the steps used to create a display format. For this reason, you may find that SDA is considerably easier to use than the FORMAT procedure.

SDA has several advantages over using the FORMAT procedure to create your display formats:

- You can design your displays at the display station. This allows you to see immediately how the display will look when it is shown.
- You can use SDA to test your displays. By controlling which indicators are on or off and by specifying the order in which a series of displays is shown, you can get a good idea of how your displays will work when the application is run.
- SDA does most of the work for you; you need only supply certain control information for the display, and the location and characteristics of the fields to be displayed.
- SDA also offers some additional options that may help you in creating your displays and coding and documenting your application programs. Using SDA, you can call source entry utility (SEU) or development support utility (DSU) full screen editor to create or update source and procedure members. Furthermore, you can use SDA to create RPG II and WSU program specifications for the display formats that you create.

The *Creating Displays* manual describes how to use SDA to create display formats.

Using the FORMAT Procedure to Create a Display Format

The FORMAT procedure, which runs the $SFGR utility program, is part of the SSP support. If you use the FORMAT procedure instead of SDA, you must first design your display on paper, or on a preprinted form. Having designed the display, you then use either SEU or the $MAINT utility program to enter and create the display format source member. And finally, you run the FORMAT procedure to convert the source member into the load member to be used by your program.

After the FORMAT procedure creates the load member, the system prints information about the display format you have defined. This information is useful for documenting your displays and can be used for correcting problems with your display formats and programs. You can refer to this printout when coding the programs that will use the display formats you have defined.

Although SDA is easier to use than the FORMAT procedure, you may find that you can use SEU to make minor changes to your display format source members and then run the FORMAT procedure to compile them faster than if you use SDA to make changes. You may want to use both SDA and the FORMAT procedure, and then decide which works better for you.

The *Creating Displays* manual describes how to use the FORMAT procedure to create display formats.

Creating Help Text for Your Displays

Using DW/36 (DisplayWrite/36), SDA, or the FORMAT procedure, you can define help text for your displays. Help text can be used to explain all or a portion of a display shown by the application program. To provide help text to your application users, you define help areas on the display used by your application program. Each help area is defined by a specification called the help definition specification, or H specification.

DW/36 lets you define help text and store it as a document in a folder. The *Getting Started with DisplayWrite/36* manual has information about defining help text using DW/36.

For SDA or the FORMAT procedure, the help area and its H specification correspond to help text on a special kind of display format called a **help format**. To use help formats, you must first disable the Help key (so that a return code indicating that the key is pressed is not sent to the program). This allows the system to detect whether the Help key is pressed, rather than allowing the program to detect it. Then, if the cursor is within a help area when the Help key is pressed, the corresponding help format is displayed. Using the Roll Up (Roll↑) and Roll Down (Roll↓) keys, the user can page through other help formats you have defined for the display. Once the user has viewed the help formats, the user can return to the original display by pressing the Enter key or a command key.

Using SDA Application Help Support to Create Help Text

Application help is a part of SDA that allows you to easily define and delete help specifications (H-specs) within your $SFGR format members. This simplifies the creation of help text for your applications. You can use SDA to do the following to your H-specs:

- Add
- Update
- Browse
- Browse all
- Delete
- View

Application help adds three new members to the SDA library (#SDALIB): #SAHF, #SA@HF, and #SA@HH.

Help areas and help formats are a good way of providing user instruction on the system, separate from the application they describe, and ready to use whenever necessary. Although you should design your displays and your application with the idea of supplying help to your users, you can easily add help to an existing application by making a few simple changes to the existing display formats and by creating the help format or formats that contain the help information. Generally, you do not have to rewrite or recompile your programs to support the new help information.

The *Creating Displays* manual has more detailed information about how help areas and help formats are designed and created.

Using Display Formats with the Programming Languages

Each of the programming languages can use display formats. In addition, a procedure can use a PROMPT OCL statement to show a display. The display formats that you create allow a program or procedure to use the display station as an input or output device.

Programs written with the work station utility (WSU) can use display formats. However, WSU requires some special entries to the S and D specifications; those entries and their use are not described in this manual. The *WSU Guide* manual has more information about WSU and how the WSU S and D specifications are coded.

This section briefly describes how programming languages use display formats and how the display format load member is identified and described.

Using Display Formats with RPG II

Programs written in RPG II use a WORKSTN (work station) file to use display formats. RPG II WORKSTN file programs require file description, input, and output specifications. In order to code these specifications correctly, you must use the information printed by the $SFGR utility program after it has created the display format load member. This means that any display formats an RPG II program is to use *must* be designed, coded, and created *before* the RPG II program is written.

The file description specifications for a WORKSTN file identify, among other things:

- The file name assigned to the WORKSTN file.
- An indication that the WORKSTN file is a combined file. A combined file is capable of being both an input *and* an output file.
- The maximum length of the data that is read from or written to the display format.

Most importantly, the file description specifications identify the display format load member that contains the formats used by the RPG II program.

Because a WORKSTN file is a combined file, the data that is read from the display format must be described on the input specifications.

The request for a particular display format and the data to be displayed are identified in the output specifications. The output record contains the program-supplied data that is to be sent to the display format.

The *Programming with RPG II* manual has more information about the use of RPG II WORKSTN files.

Using Display Formats with COBOL

Programs written in COBOL use a TRANSACTION file to read from and write to display stations. The TRANSACTION file associated with the display station must be identified by the FILE-CONTROL paragraph of the CONFIGURATION section of the ENVIRONMENT division. The ASSIGN clause of the FILE-CONTROL paragraph associates the TRANSACTION file with a display format load member to be used by the COBOL program.

A WRITE statement, used in the PROCEDURE division of the COBOL program, identifies the specific format that is displayed. In addition, the WRITE statement is used to send program-supplied data to the display.

A READ statement, also used in the PROCEDURE division of the COBOL program, accepts data entered in the display format.

The *Programming with COBOL* manual has more information about using COBOL TRANSACTION files.

Using Display Formats with BASIC

Programs written in BASIC use a work station file to send or receive formatted data to or from the display station. The display format load member that contains display formats used by a BASIC program is associated with a work station file.

The work station file (or display format) is used with the following statements:

- OPEN (WS, or work station)
- WRITE
- REWRITE
- READ
- REREAD

The OPEN statement assigns a file reference to the display format load member that contains the display formats used by the BASIC program.

The WRITE statement identifies the specific display format that is to be displayed. In addition, the WRITE statement can send data to the display.

The REWRITE statement is similar to the WRITE statement. The REWRITE statement is used to write over fields in the display format with any new data.

The READ statement reads data entered on a display. The REREAD statement rereads data entered on that display.

The *Programming with BASIC* manual has more information about using BASIC work station files.

Using Display Formats with FORTRAN IV

In order to use display formats with your FORTRAN program, you must supply the compiler with certain control information. That information is placed in a source member, and contains the following statements:

- A READ device option statement that specifies DEVICE-KEYBDF.
- A DISPLAY device option statement that specifies DEVICE-CRTF and the name of the display format load member that contains the display formats used by the program.

The DISPLAY device option statement specifies that output is to be directed to the display, and it specifies the name of the display format load member to be used by the program.

The DISPLY subprogram is used to initialize and return control information for the keyboard and the display.

The WRITE statement is used to send data from the program to a device, in this case, the display screen. In addition, the WRITE statement specifies the number of the FORMAT statement that describes the form of the data that is to be sent to the display screen.

The READ statement is used to receive data from a device (in this case, the keyboard) and send it to the program.

The *Programming with FORTRAN IV* manual has more information about how a FORTRAN program uses display formats.

Using Display Formats with Assembler

To use display formats with an Assembler program, you should be familiar with several macroinstructions, including:

- $DTFW, which defines the file for a display station
- $DTFO, which defines the DTF labels, offsets, field contents, and field lengths for all devices and access methods
- $WSIO, which constructs a means of sending output to or receiving input from the display station
- $WSEQ, which constructs labels for display station device-dependent values

The $DTFW macroinstruction identifies the display format load member that contains the formats used by the Assembler program.

The $WSIO macroinstruction builds the code to modify the file for a display station and to issue a call to perform a specified operation. The $WSIO macroinstruction also identifies the specific display format to be used, and specifies the requested operation. Operations that can be requested include a PUT, which sends data to the display, and a GET, which receives data from the display.

The *Programming with Assembler* manual has more information.

Using Display Formats within a Procedure

If you want a procedure to show a display that prompts for input data, use the PROMPT OCL statement. The PROMPT OCL statement allows you to:

- Prompt for up to 64 procedure parameters (or a total of 1024 characters) by using one or more display formats

- Define each parameter for the user

- Assign default parameters

- Control various display format functions

- Show the display format to be read on the first read operation in a program (This is called the **read-under-format** technique.)

When you show a display using the PROMPT OCL statement, any parameters that have a value cause the corresponding display format indicator to be set on. For example, if parameters 1 through 5 and 7 have values (parameter 6 does not have a value), display format indicators 01 through 05 and 07 are set on.

The PROMPT OCL statement in the *System Reference* manual has information about showing display formats and using parameters to set indicators on or off.

You can use this feature to:

- Display defaults for the parameters.

- Highlight a specific field. You could do this when a parameter is entered wrong, to allow the user to identify the field in error.

- Position the cursor to a specific field. You could do this when a parameter is entered wrong, to allow the user to key the parameter again correctly.

Using the Read-Under-Format Technique

The read-under-format technique allows the application user to enter information on a display while the program that uses the display is starting. When the read-under-format technique is used, a program or procedure displays a format, and the next program called reads it. This format is first displayed by a program or a PROMPT OCL statement with PDATA-YES specified. If a SRT program displays the format, it then goes to end of job. A MRT program displaying the format releases the requesting display station. While the next program is being started, the user can enter information on the display. When the user enters the display, the information is sent to the second program.

Note: The program data, if any, on the second procedure call or INCLUDE OCL statement is ignored.

The read-under-format technique can be used with all types of applications. This technique can decrease the size of a program because fewer read and write operations are required. Although the read-under-format technique might increase response time because of the extra work the system does while starting and ending a program, overall performance may be improved. Performance improves because two tasks happen at the same time: while the second program is being started, the user is already entering data for the first input operation in that program.

The following example shows how the read-under-format technique is used with two displays and two programs. The PROMPT OCL statement in the *System Reference* manual has more detailed information.

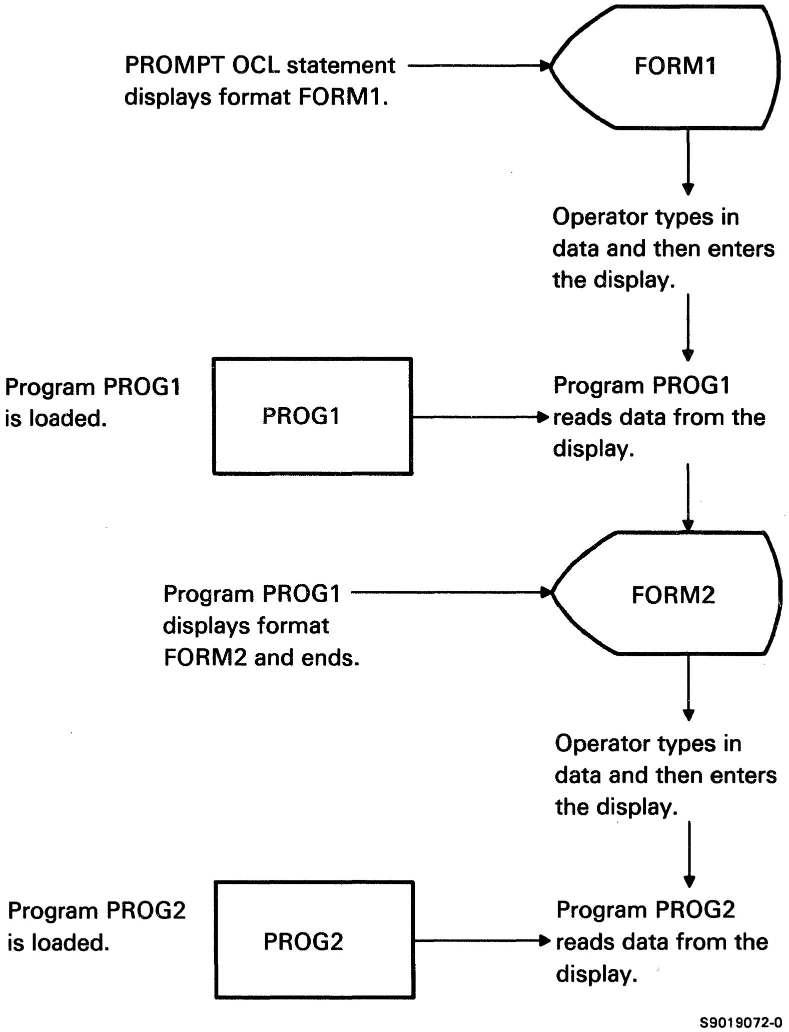

PROMPT OCL statement displays format FORM1.
FORM1
Operator types in data and then enters the display.
Program PROG1 is loaded.
PROG1
Program PROG1 reads data from the display.
Program PROG1 displays format FORM2 and ends.
FORM2
Operator types in data and then enters the display.
Program PROG2 is loaded.
PROG2
Program PROG2 reads data from the display.
S9019072-0

Figure 13-1. Example of Read Under Format

Chapter 14. Messages and Message Members

The purpose of this chapter is to:

- Describe what messages and message members are.
- Describe how you can use messages and message members.

Message Concepts

The system and your programs use messages to communicate with you and your application users.

Displayed messages can be grouped into the following categories.

- Informational messages. These are messages that, for example, indicate the status of a job that is running. An informational message that is displayed could be: `LISTLIBR procedure running` or `Payroll program running`.
- Prompting messages. These displayed messages ask a user to enter some type of information. For example: `Enter the library member name`.
- Error messages. These messages indicate that an error has occurred, and the system waits for a response.

  Some IBM-supplied error messages have automatic responses and severity levels. This allows the system to respond to them automatically, rather than having the operator enter the response. You can also specify automatic responses for your error messages.

Displayed messages also allow a user to communicate with other display station users. The manuals "Operating Your System" and "Using Your Display Station" use the MSG control command to send messages to another display station user.

IBM-supplied printed messages are used to show errors or information about source members, such as programs, display formats, menus, or message members that have been compiled.

Your applications can use printed messages for report headings, for example.

Message Member Concepts

A **message member** is a library member that defines the text of each message and its associated message identification code (MIC). You can use message members to define messages for programs, display formats, and procedures. Using message members allows you to store your messages in **one** place, and then reference those messages by number from several places (for example, from programs, procedures, and display formats) instead of coding the text to be displayed or printed each time. Using message members also ensures that your application users will not see the same message worded several ways, and message members save you time because you can code a MIC number rather than the message text. You typically use messages for:

- Titles of displays or listings. For example, `Stock Status Report.`
- Operator information shown on displays. For example, `Press the Enter key to continue.`
- Application error messages, either printed or displayed. For example, `Customer number must be entered.`
- Procedure substitution using the ?Mmic? expression, (the *System Reference* manual describes the ?Mmic? substitution expression).

The system allows you to create first-level and second-level message members.

You can create messages that allow data to be inserted when a procedure is running. "Inserting Variable Data into Displayed Messages" on page 14-13 has information about this function.

For the ideographic version of the SSP, the system allows you to create message members that have the message in two languages; for example, Katakana and Kanji. The *System Reference* manual has more information about creating these types of message members. Chapter 20, "Ideographic Data Concepts and Considerations" describes ideographic characters.

First-Level Message Members

These message members allow messages that have from 1 to 75 characters of text. First-level messages can be used by all programming languages and by procedure control expressions.

A sample first-level message source member is shown below:

```
1 MSGSAMPL,1
2 0001 Enter your name
  0002 Enter yesterday's date
  0003 ACCOUNTS PAYABLE APPLICATION
```

The sample indicates:

1 The name that will be assigned to the message load member is **MSGSAMPL**, and the member contains first-level messages (indicated by the 1 following the name).

2 The numbers (0001, 0002, and 0003) are **message identification codes** or **MICs**. These numbers identify the message text to be used for the MIC number. For example, to display or print the message `Enter your name`, you would use MIC 0001.

This example shows the message member in its source form. In order for your programs and procedures to use these messages, you must create a load member from the source. The *System Reference* manual describes how you use the CREATE procedure to create source and load message members.

Second-Level Message Members

These message members allow messages that have from 1 to 225 characters of text. Second-level messages can be used by some programming languages; see the appropriate programming language manual for more information.

Designing Message Members

When you are designing your applications, you may want to:

- Create a single message member and put all the application messages in it.
- Create several message members and group messages by:
  - Program type. For example, you could place the order entry messages into a member named ORDERMSG.
  - How they are used. For example, you could place the displayed messages into a member named MSGDISP, and the printed messages into a member named MSGPRINT.

Providing Automatic Responses for Messages

You can have the system automatically respond to displayed system and application messages, by using the RESPONSE and NOHALT procedures. When a message has an automatic response, the message is not displayed; instead, the response is immediately given by the system. This allows you to define a specific response that is to be given automatically by the system, rather than having an operator enter a response. In order for a message to have an automatic response, the message must be in a message member.

Some IBM-supplied displayed messages already have automatic responses and severity levels assigned; you can use the RESPONSE procedure to change these values. For your application's displayed messages, you can assign your own automatic responses and severity levels by using the RESPONSE procedure. Your application messages that have automatic responses should be displayed by using the ERR procedure.

IBM-supplied displayed messages allow the following responses: 0, 1, 2, 3, D (Dump), and H (Help). The automatic response you choose is valid only if that response is allowed by the message. For example, if a message had only options 2 and 3, an automatic response of 1 would not be valid, and the message would be displayed. The H (Help) option cannot be specified as an automatic response.

The automatic response facility allows option N so that you can select the IBM-supplied response. You would use this option when you have changed the IBM-supplied response and you want to change it back.

When you define or change an automatic response for a message (either IBM-supplied or one of your application messages), you specify:

1. The MIC of the message to be responded to. If the message is one of your own application messages, you must specify the load member containing the MIC. If the message is an IBM-supplied message, the system automatically determines the load member.

2. The automatic response to be used.

3. The severity level for the automatic response.

The IBM-supplied system messages have a particular severity level (1 through 5). The severity levels and automatic responses are shown in the messages manuals. When you assign an automatic response to your messages, you also specify a severity level (unless N is specified as the automatic response). The system uses these severity levels to respond to the messages defined as automatic response messages. For example, you may allow an automatic response for messages with a severity level of 3 or lower, but require a manual response for messages with a severity level of 4 or 5.

The following table lists suggested severity levels for different categories of messages. These severity levels are used by IBM for system messages. When assigning automatic responses for your own messages, you can use these severity levels as a guideline.

Severity Level	**Explanation**
1	Informational messages that require a response (option 0 only).
2	Messages with one option (like a warning message). Also, messages with two or more options where one of the options is to retry the function being performed.
3	Program error messages; these messages usually have more than one option for the operator to choose.
4	Messages for severe errors, such as device errors or permanent input/output errors.
5	No automatic response is defined for the message.

The *System Reference* manual describes how to use the RESPONSE procedure to assign automatic responses and severity levels to messages. The *System Reference* manual also describes how to use the NOHALT procedure or OCL statement to specify a severity level for a job, for a session, or for the system.

The following sample OCL statements show how to assign a severity level for a job. The NOHALT OCL statement causes all messages with a severity level of 2 or lower to receive an automatic response for the duration of the INVPROG program.

```
// NOHALT 2,JOB
// MEMBER USER1-MSGDISP,LIBRARY-INVLIB
// LOAD INVPROG,INVLIB
// RUN
```

The NOHALT OCL statement specifies a severity level of 2 for the job. The MEMBER OCL statement specifies the message member to use (MSGDISP) and the library containing the member (INVLIB). If the program being run (INVPROG) displays a message that has an automatic response that matches one of the displayed options, and if the severity level of that message is 1 or 2, the system automatically responds to the message.

Note: Any system message with a severity level of 1 or 2 will also be responded to automatically.

Considerations for Automatic Responses to Messages

The following are some considerations you should be aware of when using the automatic response capabilities of the system:

- Both the message and the automatic response of the system are written to the history file. The message is not displayed at the display station.

- Do not create an automatic response to error messages that are displayed by the RESPONSE procedure or the $ARSP utility program. Doing so could cause an error in your automatic response statements to go undetected.

- Do not create an automatic response to system messages indicating that the system is retrying an operation. For example, if invalid data is found on a diskette, a user can select an option to retry the reading of the diskette. If you specify the retry option as an automatic response, the system will continually retry reading the diskette, without giving the user a chance to stop retrying. (Specifying a retry option might fill up the history file very quickly.)

- Use caution when creating an automatic response to informational messages. If you want an operator to see the message, the message should require a response.

- Do not create an automatic response to messages that require the user to do something before the message is responded to. For example, if the operator has to align forms in the printer and you specify an automatic response to the message indicating that the forms are to be aligned, the user may not have an opportunity to align the forms in the printer correctly.

- Do not create an automatic response to messages indicating that there are serious problems with the system such as device errors and messages that indicate a service representative should be called.

The *System Reference* manual lists additional considerations that apply when you are using the RESPONSE procedure to set up automatic responses.

Unattended System Operation

You can use the automatic response capabilities of the system to have the system process programs without any operators present. For example, you can place certain programs in priority level 0 of the job queue during the day, then start the job queue in the evening, and have the system process these programs during the night. Any messages generated by the system or the programs should have automatic responses created for them.

If you are planning to run programs while the system is unattended, you should consider the following:

- Printed output:

  - Have the proper forms in the printer.

  - Be sure the forms are aligned correctly in the printer.

  - Have enough forms so that all jobs can print.

  You can have reports written to the spool file and not be printed until an operator is present. You can do this by specifying PRIORITY-0 on the PRINTER OCL statement.

  You can also have the system print your output while no operators are present and have one copy stored in the spool file. You can do this by specifying HOLD-YES on the PRINTER OCL statement. This allows you to print your output but, if something goes wrong with the paper or the printer, you can print the output that is saved in the spool file.

- Diskette processing:

  - Have the diskettes placed in the correct slots or have the diskette magazine loaded for any job using diskettes.

  - Do not run any jobs that require diskettes to be removed from or inserted in the diskette drive.

- Tape processing:

  - Have the tapes placed in the correct tape drives.

  - Do not run any jobs that require tapes to be removed from or inserted in the tape drive.

- Program processing:

  - Run programs that do not use a display station.

  - Run jobs that have been tested and are working correctly.

  - Have some method of restarting your program in the event of program errors.

Programming Guidelines for Message Members

This section describes how you can create, change, and use message members.

The *System Reference* manual has more information.

Creating or Changing Message Members

You create message members by first creating a message source member. Use the source entry utility (SEU) or Development Support Utility (DSU) to create the source member. Message source members must be entered in a special format. Also, SEU has special display formats to help you enter message members.

After the message source member is created, use the CREATE procedure to compile the source member into a message load member. The load member is what is used by the system to display or print the message.

If you want to change a message:

1. Use SEU or DSU to change the message source member.

2. Use the CREATE procedure to generate the new message load member.

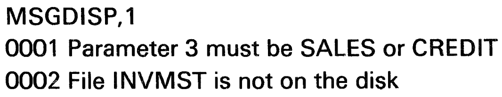
The *System Reference* manual describes how to use the CREATE procedure to create message members and the special format for message members.

Assigning Automatic Responses and Severity Levels

You assign automatic responses to messages in message members by first creating a response source member. Use the source entry utility (SEU) or the Development Support Utility (DSU) to create the source member. The *System Reference* manual describes how to use the RESPONSE procedure to create automatic responses for messages and the special format for the response source members.

The following shows a source message member and its corresponding response source member for a sample application. The library INVLIB contains the message member MSGDISP. The response source member specifies automatic responses for two messages, MIC 0001 and MIC 0002; and it specifies that the MICs are in message member MSGDISP in library INVLIB.

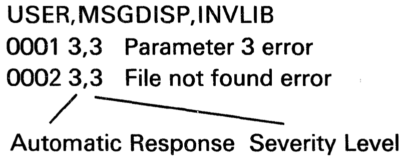
Message Member MSGDISP in Library INVLIB



```
MSGDISP,1
0001 Parameter 3 must be SALES or CREDIT
0002 File INVMST is not on the disk
```

S9019131-0

Response Source Member



```
USER,MSGDISP,INVLIB
0001 3,3  Parameter 3 error
0002 3,3  File not found error
```

Automatic Response
Severity Level

S9019123-0

You can assign automatic responses to as many of your messages as you want. You may have some messages for which there are no automatic responses.

After the response source member is created, use the RESPONSE procedure to assign the automatic responses and severity levels to the message load member. The NOHALT procedure or OCL must be run before the RESPONSE procedure will be effective.

If you want to change a response or severity level:

1. Use SEU or DSU to change the response source member.

2. Use the RESPONSE procedure to assign the new values.

Note: If you recompile the message source member (by using the CREATE procedure), the automatic responses no longer apply. You will then have to run the RESPONSE procedure again to apply the automatic responses and severity levels.

Specifying a Message Member to Be Used within a Procedure

The MEMBER OCL statement assigns message members to a job. The messages in the assigned member can be used in the procedure, by programs run by the procedure, and by display formats. For example, the following MEMBER OCL statement:

```
// MEMBER USER1-DISPMSG,LIBRARY-INVLIB
// LOAD PROG1
// RUN
```

assigns first-level message member DISPMSG from library INVLIB to the program PROG1.

Displaying Messages from Procedures

You can use several statements and procedures to display messages from procedures.

// * (Informational Message) Statement

This statement displays a message from a procedure to the operator running the procedure. For example, the following statement:

```
// * 'Enter today''s date:'
```

displays the message `Enter today's date:`

// ** (System Console Message) Statement

This statement displays a message from a procedure on the system console. For example, the following statement:

```
// ** 'Procedure PROC1 is running'
```

displays the message `Procedure PROC1 is running`

Displaying Your Messages in the Same Format as System Messages

The ERR procedure displays your error messages in the same format as system-displayed messages. For example, if the following statements are in a procedure:

```
// MEMBER USER1-DISPMSG,LIBRARY-INVLIB
ERR 0001,23
```

the MEMBER statement assigns a message member to the job. The ERR procedure displays message 0001, and allows options 2 and 3 to be taken. If message 0001 is `Parameter 3 is invalid`, the ERR procedure displays:

```
USER-0001 (  23)
Parameter 3 is invalid
```

For options 0, 1, and 2, the system sets a return code that can be tested using the ?CD? substitution expression. For option 3, the job is immediately canceled. The *System Reference* manual describes the ?CD? substitution expressions.

Ensuring That Required Parameters Are Entered

The ?nR'mic'? substitution expression displays a message when the nth parameter does not have a value. You could use this expression when you are checking the parameters of a procedure to ensure that a required parameter has been entered. In the following example, the EVALUATE statement processes the substitution expression.

```
// MEMBER USER1-DISPMSG,LIBRARY-INVLIB
// EVALUATE ?1R'0004'?
.
.
.
```

If the first parameter was not entered when the operator started the procedure, message 0004 from message member DISPMSG would be displayed. The operator would then be prompted to enter the parameter.

Using Messages with Programs

You can use messages with programs. You can either code the text in the program or use message members. See the appropriate language manual for more information.

Using Messages with Displays

You can use messages with display formats. You can either code the text in the display, have the program or procedure display the message text, or use message members. See Chapter 13, "Displays," for more information.

Inserting Variable Data into Displayed Messages

If you code a first-level message such that it contains one or more fields of # signs, you can use the ERR procedure to insert variable data into the message when it is displayed. A parameter of the ERR procedure specifies the data to be inserted.

The data is substituted for # signs contained in the message text, where each # sign indicates a character to be inserted. For example, if MIC 0003 in message member DISPMSG is:

```
Procedure ######## is on the job queue
```

and the following statements are in a procedure:

```
// MEMBER USER1-DISPMSG,LIBRARY-INVLIB
ERR 0003,0,INVJOB
```

the message displayed would be:

```
USER-0003 Options (0     )
Procedure INVJOB   is on the job queue
```

Because the characters from the ERR procedure are substituted one for one for the # signs, you can make messages that have more than one field of insert data. For example, if MIC 0005 in message member DISPMSG is:

```
Job ######## requires file ########
```

and the following statements are in a procedure:

```
// MEMBER USER1-DISPMSG,LIBRARY-INVLIB
ERR 0005,0,'INVJOB  CUSTMST'
```

the message displayed would be:

```
Job INVJOB   requires file CUSTMST
```

Note that the variable fields in the message must be long enough to handle the data to be inserted. Also, notice that blanks must be inserted if the data is shorter than the variable field in the message.

Another way to use the ERR procedure to insert data in displayed messages is to create one message that contains 75 # signs. This allows you to insert the entire message from the ERR procedure. For example, if MIC 0004 in message member DISPMSG is:

```
##################  ...  #####
```

75 # signs.

and the following statements are in a procedure:

```
// MEMBER USER1-DISPMSG,LIBRARY-INVLIB
ERR 0004,13,'A required file cannot be accessed.'
```

the message displayed would be:

```
USER-0004 Options ( 1 3 )
A required file cannot be accessed.
```

The *System Reference* manual describes the ERR procedure.

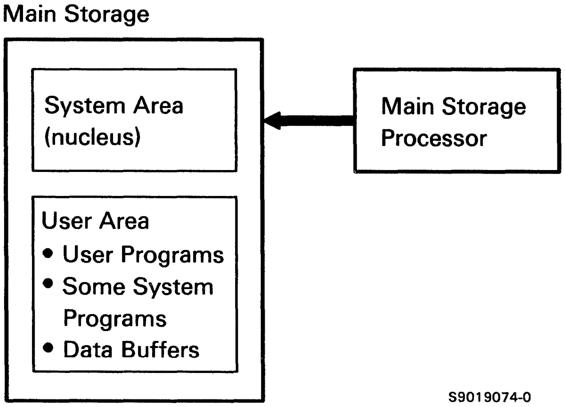
Chapter 15. Main Storage

This chapter describes the areas of main storage and the main storage processor.

Main Storage Concepts

Main storage contains programs, data buffers, and instructions for the system. Main storage also contains work areas that are used by both the system and your application programs.

Main storage is divided into a system area (called the **nucleus**) and a user area of main storage.



The system has a main storage processor that processes the system and application program instructions in main storage. The following information describes the nucleus, the user area, the data buffers, and the main storage processor.

System Area (Nucleus) of Main Storage

The nucleus contains parts of the SSP (System Support Program Product) that must be in main storage all the time. The system uses these parts of the SSP to manage system resources such as:

- Disks
- Printers
- Display stations

The nucleus consists of two separate portions. The size of one portion, the **fixed-sized portion**, does not vary. The size of the other portion of the nucleus, the **variable-sized portion**, can vary in size.

Expansion and Contraction of the Nucleus

Because the nucleus is composed of two portions (the fixed-sized portion and the variable-sized portion), the size of the nucleus can vary. The size of the nucleus affects how much main storage space is available for user programs in the user area of main storage.

The system checks the new size of the user area whenever it takes space away from the user area to use as additional nucleus space (expansion).

The system will send a message to the system operator when it is running out of user storage. The purpose of the message is to let the operator know what is happening so the operator will stop or cancel some jobs. Canceling or stopping jobs will stop nucleus expansion and/or cause the nucleus to contract.

The system will prevent the initiation of new jobs once the user area is reduced below 24K.

Fixed-Sized Portion of the Nucleus

The fixed-sized portion of the nucleus is a 27K-byte area of main storage that is reserved for use by the system. Included within the fixed-sized portion of the nucleus are:

- Work spaces used by control storage programs.
- Disk data management.
- Work station data management.
- Command processor program.
- The transient area. This area is a 4K-byte area of main storage that contains SSP routines that do special functions. Only one routine can use the transient area at a time, and the various routines take turns using the transient area.

Variable-Sized Portion of the Nucleus

The amount of storage in the variable-sized portion of the nucleus depends on the options selected during configuration or IPL (initial program load). Within the variable-sized nucleus are:

- Nonswappable system routines and work areas/buffers
- Assign/free area and space for system control blocks and pointers

This section includes several charts describing the sizes of system programs. Use these charts to help determine the amount of user storage available for your programs and data buffers, depending upon the current configuration and status of your system.

Nonswappable System Routines

These routines are selected during system configuration. When you request the support that uses these routines, they run in the user area of main storage, yet are considered part of the nucleus. These routines are loaded when requested and remain in storage until they are no longer needed.

The following table lists the nonswappable system routines and their sizes.

Nonswappable System Routines	Description	Size (in bytes)
Batch BSC Interrupt Handler	Used when batch BSC (binary synchronous communications) is configured and active.	4K
SDLC Interrupt Handler	Used when SDLC (synchronous data link control) communications is configured and active.	8K
SSP-ICF BSC Interrupt Handler	Used when a BSCEL, BSC CCP, BSC CICS, or BSC IMS/IRSS subsystem is enabled and active.	12K
BSC 3270 Interrupt Handler	Used when BSC 3270 device emulation is enabled and active. This routine also requires 4K bytes of task work space that must be resident.	8K
MSRJE BSC Interrupt Handler	Used when MSRJE BSC is active in the system.	10K
X.25 Interrupt Handler	Used when X.25 communications is active in the system.	42K
Printer Data Management and Spool Intercept Routine	Used when print spooling is active.	1K
Folder Management and I/O Router	Used to route control to the folder management I/O routines	2.25K

Assign/Free Area

The assign/free area contains areas used by the system for job processing. Control blocks and buffer spaces used by the system are in the assign/free area. The size of the assign/free area varies depending on:

- The number of programs running in main storage
- The number of active display stations
- The number of active printers
- The number of files being processed
- Data communications being active
- Disk cache being active

Each program that runs from a display station, that uses one disk file, and prints one report, uses about 2K bytes in the assign/free area. The following table shows the approximate amount of assign/free area that is used for the following items:

Item	Amount of Assign/Free Area Needed For Each Item
Each Active Program	512 bytes
Each Active Display Station	512 bytes
Each Active Printer	512 bytes
Each Active File [1]	512 bytes
Data Communications	Variable amount based upon type of communications active
Disk Cache Resident Code	512 bytes

[1] When a file with multiple indexes is used, each index is an active file (even when that file is not used by the program).

The assign/free area is rounded up to 2K bytes. The system also tries to keep 2K bytes available for the next time that space is needed in the assign/free area.

User Area of Main Storage

To run your programs, the system loads them into the user area of main storage.

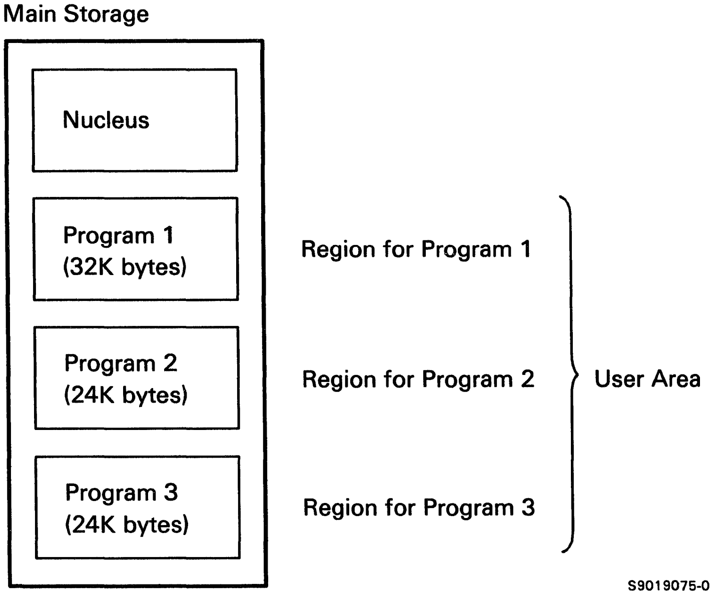
Organization of the User Area

The user area consists of all main storage that is not currently part of the nucleus. The user area of main storage is divided into 2K-byte segments called **pages**.

The system uniquely identifies these pages to keep track of both programs and data used by programs. When combined in main storage, the pages for each program form what is called a **region**. The size of the region used by a program is called the **region size**. The default region size assigned to each program is 24K bytes. The largest region size that can be assigned to a program is 64K bytes. You can use the REGION OCL statement or the SET procedure to change the default region size to a larger or smaller value. However, when the program runs, the system assigns only as much storage as is needed for the program and its data. For example, if you set the default region size for a program to 24K bytes but the program actually requires only 16K bytes of main storage, the system assigns only 16K bytes (8 pages) to the program when the program runs.

Several programs can run in main storage at the same time. The system uses a method of swapping programs into and out of main storage to run jobs. For more information about job processing and program swapping, see Chapter 17, "Jobs and Job Processing."

The following example shows the user area of main storage separated into three regions. Program 1 has a 32K-byte region, and programs 2 and 3 each have a 24K-byte region.



When the nucleus expands, some pages in the user area are claimed by the system and are not available for use by application programs. When these pages are no longer needed by the nucleus, they are returned for use in the user area. When the additional pages are claimed, more swapping of programs may occur.

Many system programs run in the user area of main storage. For example:

- System utility programs (such as $COPY or $MAINT)
- Transient system routines

Transient System Routines

Some system routines do not have to be resident in main storage. These transient (or temporary) routines are loaded into the user area of main storage from the system library as they are needed to run programs. If the routine is not needed, the main storage space used by the transient program is made available for other user or system programs.

The following table lists some of the important transient system programs and their sizes:

Program Name	Description	Size (in bytes)
Work Station Data Management (GET operation)	Used to get data to the program from the display station	4K
Work Station Data Management (PUT and PUT override operations)	Used to display data from a program to the display station	4K
SSP-ICF Management	Used for the Interactive Communications Feature (SSP-ICF)	8K

This table does not list every system program, because the amount of main storage required for some of these programs varies according to how the program is being used. To find out how much main storage a particular program actually uses at a given time under certain conditions, you can use the System Measurement Facility (SMF). The *System Measurement Facility Guide* has information about how to use SMF.

Buffers Used by Programs

Data buffers are not compiled into the program load members. Instead, when the program is loaded into main storage and run, the buffers are placed into main storage **outside** the load member.

Each disk file used by a program requires a buffer when the program is run. The buffers are allocated by the system as the files are opened. A disk file buffer consists of:

- A control block, which is used by disk data management to read and write file data.
- A data block, which contains one or more sectors of disk file data.
- An index block (for indexed files).

Buffers are also needed for print spooling intercept buffers when print spooling is used.

You can define two types of blocking for disk files: record blocking and index blocking. Record blocking can be specified in your program or on the FILE OCL statement. Index blocking can only be specified on the FILE OCL statement.

Blocking affects the amount of data that the system must read from and write to the disk in one disk operation. By increasing the blocking size, the number of disk operations may be reduced because the system can read and write more records during a single input/output operation.

Record blocking is useful if you are using a consecutive processing method, but is probably not an advantage if you are using a random processing method. See Chapter 8, "Files" for more information about blocking.

Buffer Allocation

The system assigns buffers to programs in either of the following ways:

- By appending the buffer area to your program and increasing your program's size by the amount of the buffer. This method is used when the addition of the buffer does not make the total program size larger than 64K bytes or the available user storage.
- By placing the buffer in the disk file work space of main storage, which is an area that is separate from the program's region. This method is used when the addition of the buffer makes the total program size larger than 64K bytes or the available user storage.

Note: If space is not available in the program's region, the spool intercept buffers are placed in the assign/free area.

By sizing the buffers such that they are appended to the program rather than separate from the program, the time the system takes to run the program can be reduced because the system does not have to swap two separate areas of main storage. Also, when the buffers are separate from the program, disk data management must do extra work to access these buffers. This also takes extra time.

Because of the way program size is determined, the buffers are allocated to the program as the disk and printer files are opened. Buffers that are used the most by a program should be opened first, so that the space used by the buffer has a greater chance of being allocated to the program's region size, rather than the disk file work space.

See Chapter 8, "Files," for more information about buffers.

System-Assigned Program Attributes

Each program is assigned, by the system, one processing attribute and one storage attribute.

The **processing** attributes are:

- Reentrant. Specifies that the program code can be used by several users and that each user has his own set of data. This attribute saves space.

- Reusable. Specifies that the program code can be used by only one user at a time. If the code is being used, any other requests must wait. However, the code does not have to be loaded again for the next user.

- Reloadable. Specifies that a separate copy of the program code is loaded for each user. This attribute is assigned to user programs.

The **storage** attributes are:

- Swappable. Specifies that the program can be swapped (or stored) temporarily on disk when its main storage space is needed by another program.

- Nonswappable. Specifies that the program remains in main storage until it is completed. This attribute is allowed only for system programs.

- Refreshable. Specifies that the program code, at a certain point, is equivalent to the code stored in the library. This attribute is allowed only for system programs and program products. When the storage used by a refreshable program is needed, the storage is released. The system does not require an image of the program to be saved on disk. When the code is again needed in main storage, a new version of it is loaded from the library.

The following table shows the combinations of processing and storage attributes that are used on the system:

	Reentrant	Reusable	Reloadable
Swappable			Utilities Program Product, Language Program Products, System Programs, User Programs (except BASIC programs)
Nonswappable	Resident System Programs		
Refreshable	System Programs, BASIC Programs, Query/36	System Programs	Query/36 Data Entry Facility

Main Storage Processor

The system uses the main storage processor to process application program instructions and system commands.

The more user program instructions the main storage processor can process, the more work the system can do. The system is designed to let the main storage processor process as many program instructions as possible without having to do other things such as obtaining data for programs or controlling the input or output of data to the devices used by the system. A separate processor called the control storage processor is responsible for supervisory functions and the input and output of data.

Chapter 16. Programs

This chapter describes the various types of programs you can design, the reasons for choosing each type, and some suggestions for designing your applications.

This chapter has three main sections:

- Program concepts, which describes batch and interactive programs
- Designing applications, which presents general suggestions about the factors you should consider when deciding which type of program is appropriate
- Programming considerations, which provides more specific suggestions to help you implement your design choices

Program Concepts

This section defines major program concepts that you should keep in mind when you design a program or application:

- Batch programs.
- Interactive programs. For interactive programs, the following factors can affect your program or application design:
  - Program size
  - Number of users that can communicate with the program
  - Number of users that can request the program

Batch Programs

A batch program is one that processes records with little or no operator interaction. Typically, a batch program processes a group of related transactions that have accumulated over a given period of time. An example of a batch program is one that prints invoices at the end of the day, rather than when the order is entered.

Interactive Programs

An interactive program is one that receives requests from one or more display stations and may respond to each request as it is received. The program processes individual records or transactions at the time the request is received, rather than processing requests that accumulate over a period of time. An example of an interactive program is one that processes orders and prints an invoice at the time each order is entered.

Note: Throughout this chapter, the term **display station** *means a display station or an SSP-ICF session. Refer to the appropriate subsystem reference manual for more information about SSP-ICF sessions.*

Interactive programs use a display station file to communicate with a user. The following table shows how a display station file is identified in each programming language. For information about display station files for a particular programming language, refer to the manual for that language.

Programming Language	Display Station File
Assembler	Work station file
BASIC	Work station file
COBOL	TRANSACTION file
FORTRAN IV	READ DEVICE-KEYBD and DISPLAY DEVICE-CRT device option statements
RPG II	WORKSTN file

Typical Uses of Interactive and Batch Programs

Most applications include both interactive and batch programs. The following list shows typical uses of interactive and batch programs for order entry, accounts receivable, and inventory control applications:

Order Entry Applications	**Program Type**
Order entry	Interactive
Open order inquiry	Interactive
Inventory allocation	Interactive or batch
Print invoices	Interactive or batch

Accounts Receivable Applications	**Program Type**
Cash receipts	Interactive
Account status inquiry	Interactive
Open items	Interactive or batch
Monthly statements	Batch

Inventory Control Applications	**Program Type**
Receipts/adjustments	Interactive
Status inquiry	Interactive
Vendor code changes	Interactive or batch
Parts requisition	Batch

Program Size

An application usually includes many logical steps. A large program includes all or most of the logical steps that make up an application. If all the display stations use a large program at the same time, main storage would not be large enough to contain a separate copy of the program for each display station. Therefore, the system has to swap programs frequently. As a result, large programs tend to have poorer performance.

A small program is one logical step of an application; several small programs make up a single application. If all the display stations use a small program at the same time, main storage would be large enough to contain a separate copy of the program for each display station. Therefore, swapping occurs less often and performance improves.

Number of Users That Can Communicate with a Program

Programs can communicate with any number of display station users. Some of those users can call a program, and some can be assigned to the program. Users that call the program are called **requesters**. Users that are assigned to the program are called **acquired display stations**. An acquired display station cannot call the program.

In terms of the number of users allowed, the system supports the following types of programs:

- One-user programs

- Multiple-user programs

- No-user programs

One-User Programs

A one-user program is an interactive program that can communicate with only one user at a time. A one-user program has a display station file that is limited to one display station.

In RPG II, you can limit the number of users to one by specifying a value of 1 for the NUM continuation-line option on the WORKSTN file description specification. In other programming languages, there is no way to limit the number of users.

When a one-user program is a single-requester-terminal (SRT) program, the requester is the only display station that the program can communicate with.

Multiple-User Programs

A multiple-user program is an interactive program that can communicate with more than one user at a time. A multiple-user program, like a one-user program, has a display station file. However, the display station file for a multiple-user program allows two or more users. Those users can be requesting display stations, acquired display stations, or both.

No-User Programs

A no-user program is a batch program because it does not have a display station file. An example of a no-user program is a program that prints a disk file.

Number of Users That Can Request a Program

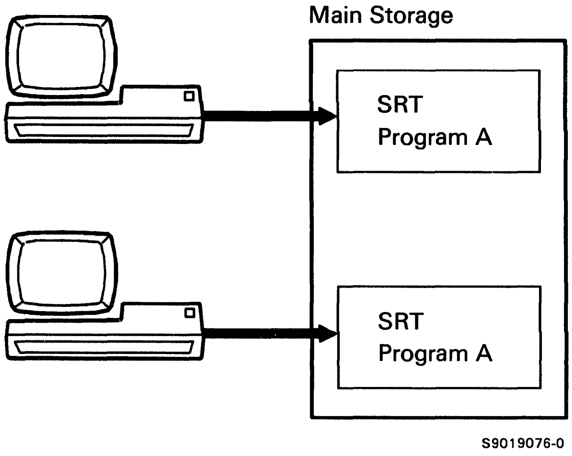
The number of requesters affects the application design because it affects the OCL statements and procedures you use to call the programs in the application. For example, if a program has more than one requester, you must use a procedure to call it.

In terms of the number of requesters, the system supports the following types of programs:

- Single-requester-terminal (SRT) programs
- Multiple-requester-terminal (MRT) programs
- Nonrequester-terminal (NRT) programs

Single-Requester-Terminal Programs

A single-requester-terminal (SRT) program can interact with only one requesting user. More than one user can request a SRT program, but, each time a SRT program is requested, a separate copy of the program is loaded into main storage. For example, if two users request SRT program A, two copies of the program are loaded into main storage.

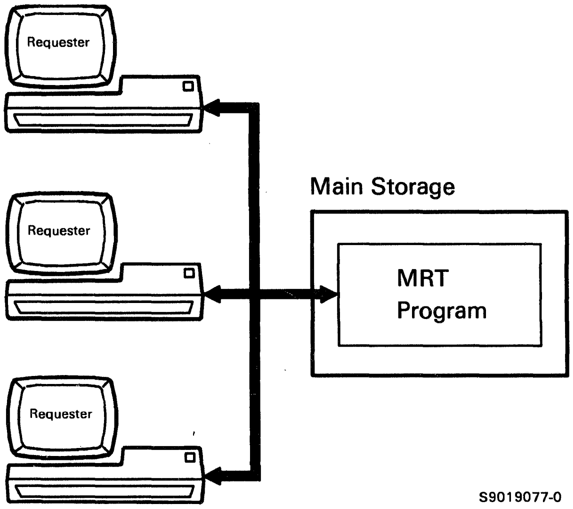


Specifying a SRT Program: Any program that is not called as a NRT program or specified as a MRT program is a SRT program. No coding is required to specify that a program is a SRT.

Difference between a SRT Program and a One-User Program: In a SRT program, the user is always a requester. In a one-user program, the user may be either a requester or an acquired display station. A one-user program is usually a SRT program, but it could be a nonrequester-terminal (NRT) program with one acquired display station.

Multiple-Requester-Terminal Programs

A multiple-requester-terminal (MRT) program is an interactive program that processes requests from more than one user at the same time, using a single copy of the program. Each user appears to have its own copy of the program, but in fact they all share the same copy of the program. A MRT program uses a display station file.



The work station utility (WSU) is a programming language that helps make coding MRT programs easier. The *WSU Guide* has more information about WSU.

Difference between MRT Program and Multiple-User Program: The difference between a MRT program and a multiple-user program lies in whether the users are requesters or acquired.

A MRT program can have as many users as permitted by the display station file definition (in RPG, that number is the NUM value). More than one of those users can be requesters.

A multiple-user program that is not a MRT can also have any number of users. However, no more than one of those users can be a requester. The other users must be acquired.

Specifying a MRT Program: Both the program and the procedure that calls the program must be specified as MRTs. You specify the program as a MRT by assigning a MRTMAX value when the program is compiled. You specify the procedure as a MRT on the replacement display for the source entry utility (SEU), the exit options display of DSU full screen editor, or the MRT parameter of the $MAINT utility program. For information about MRT procedures, refer to Chapter 18, "Procedures."

MRTMAX Value: The MRTMAX value specifies the maximum number of requesters that can be active at any given time for a MRT program. When the number of requesters equals the MRTMAX value, a subsequent requester has to wait until a current requester is finished using the program.

You can decrease the MRTMAX value by using the ATTR OCL statement when you run the program. To increase the MRTMAX value, you must recompile the program.

A program must be able to handle any number of requesters. The only control the programmer has over the number of requesters is that the number of requesters cannot exceed the MRTMAX value specified by the programmer.

NUM Value: The NUM value specifies the sum of the MRTMAX value plus the number of acquired display stations that can communicate with an RPG program. Thus, an RPG programmer can control the total number of users and the maximum number of requesters. For example, if a NUM value of 3 is specified and the program has two requesters, no more than one display station can be acquired.

The NUM value is defined on the continuation line for the WORKSTN file on an RPG file description specification.

In other programming languages, the only way to control the total number of users is to code the logic to check the number of users.

Nonrequester-Terminal Programs

A nonrequester-terminal (NRT) program has no requesters. A program becomes a NRT program when the requester purposely separates the program from the display station by using a command or OCL statement. For example, if a program is called by an EVOKE OCL statement, the system immediately separates the program from the requester; the user is free to do other work, and the program has no requesters.

Comparison of Program Types

The following table compares the characteristics of six types of programs:

One-User Program. Designed to use only one display station, which can be a requester or acquired. Can be a SRT, a NRT, or a MRT (MRTMAX = 1).	**Single-Requester-Terminal Program**. When nothing else is specified, a program is a SRT. Can be a one-user program, a no-user program, or a multiple-user program.
Multiple-User Program. Users can be any combination of requesters or acquired display stations. Can be a NRT, SRT, or MRT.	**Multiple-Requester-Terminal Program**. Typically designed to communicate with more than one user at the same time (MRTMAX = 2 or more), but can be restricted to one user (MRTMAX = 1).
No-User Program. Has no users and cannot acquire any. Can have a requester but cannot communicate with it. Can be a NRT or a SRT.	**Nonrequester-Terminal Program**. Has no requesters but can acquire any number of users. Can be a one-user, multiple-user, or no-user program.

Summary Table of Users and Requesters

The following table lists the program types that you would probably use based on the combinations of requesters and acquired display stations:

Number of Users	Number of Requesters	Type of Program	Description
1	1	SRT	Most common situation.
1	1	MRT	MRTMAX = 1. No acquired display stations.
1	0	NRT	Can acquire one display station.
More than 1	1	SRT	In RPG, can acquire up to NUM - 1 display stations. In other languages, can acquire any number of display stations.
More than 1	More than 1	MRT	In RPG, requesters + acquired display stations ≤ NUM. In other languages, the total can be any number. Common MRT situation.
More than 1	0	NRT	In RPG, can acquire up to NUM display stations. In other languages, can acquire any number.
0	0	SRT	Common situation for batch programs. Cannot communicate with users.
0	0	MRT	Meaningless combination.
0	0	NRT	Cannot communicate with users.

Designing Applications

This section presents information that can help you select the appropriate program types. It discusses all the program types defined in the previous section, but it emphasizes the considerations for interactive programs.

This section is organized around the following questions that you answer when you design a program or application:

- Should this be a batch application, interactive application, or a mixture of both?
- How should I structure my application? Do I want:
  - One large program for each user?
  - One large program shared by all users?
  - Several small programs, each user having a separate copy of one of the programs?
  - Several small programs, all users sharing one set of programs?
  - A mixture of the two preceding possibilities?
  - More than one user for each program? If so, do I want:
    - More than one requester?
    - Any acquired display stations?
- What attributes should this program have?
- How can I get the best performance possible from my programs?

Batch versus Interactive Programs

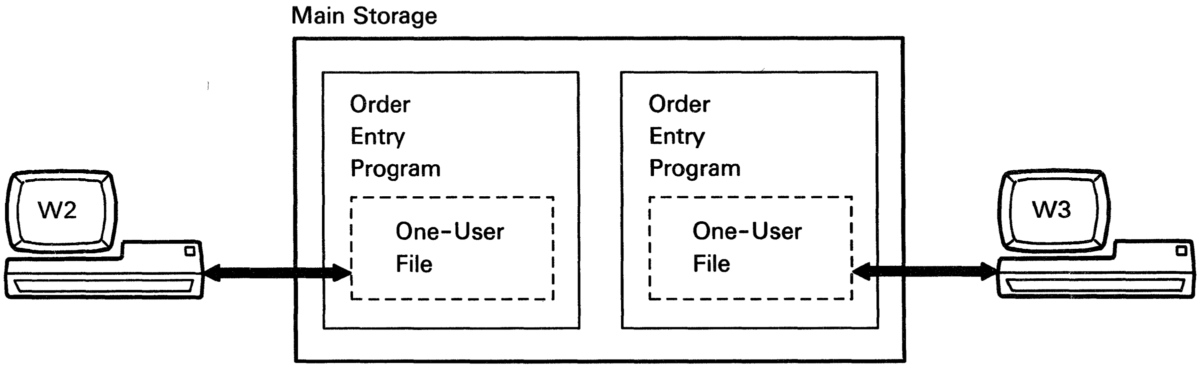
Sometimes a batch program is the best choice even though an interactive program seems preferable. For example, you may want to update your inventory file as orders are entered. However, this interactive updating has potential problems. One such problem is the difficulty of recovering an inventory master file if the system terminates prematurely. For an interactive program, it is difficult to determine whether the last transaction update actually occurred for each user. A batch program does not normally share files, so you can probably tell whether the last update actually occurred.

Application Structure

When you design an application that is run from only one display station at a time, the design decision is whether the application should consist of one large SRT or several small SRT programs. But when you design an application that is typically run from two or more display stations, you must decide whether the application should consist of programs that interact with two or more display stations concurrently or programs that have a separate copy for each active display station.

One Large Program for Each User

Most programmers initially write an application that consists of one large program. If the application is not very complex, if main storage is plentiful, and if performance is acceptable, this may be the best choice.

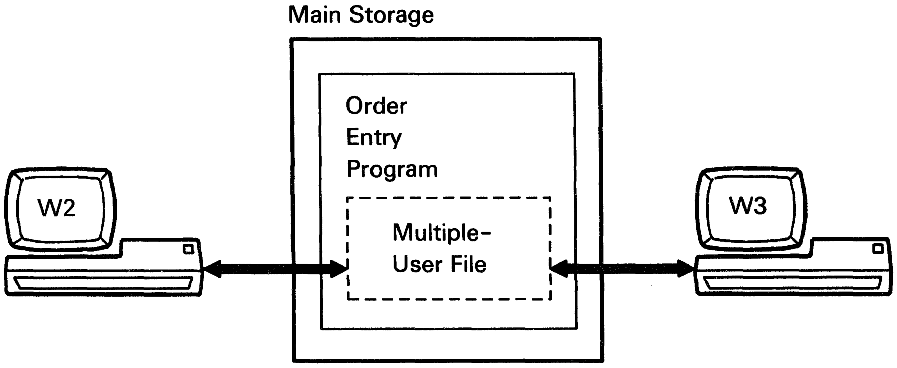


S9019078-0

One Large Program Shared by All Users

System response time may not be acceptable when one large program is shared by all users. Such a program could be more difficult to code because you may have to design tables that reflect the status of each user. However, this additional difficulty may be offset by the performance improvement because the program requires less main storage, thus reducing or eliminating swapping of the program.

WSU creates these tables as part of the program generation process. The *WSU Guide* has more information.

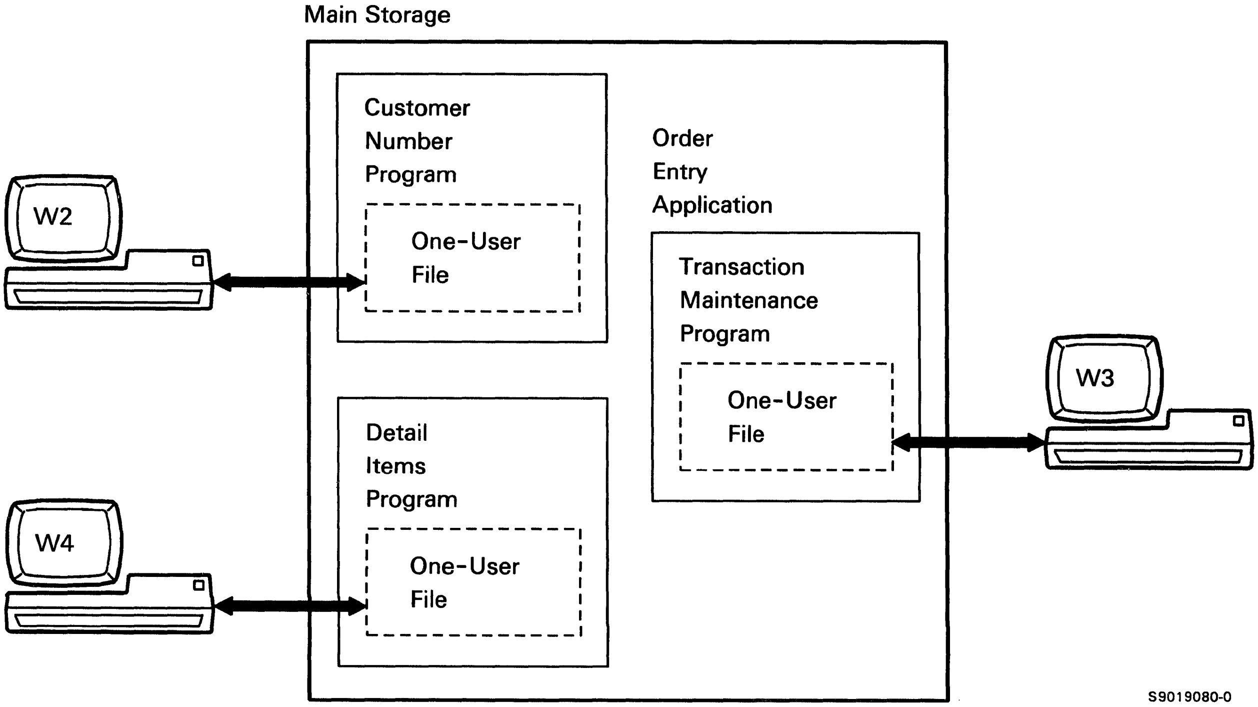


S9019079-0

Several Small Programs, Each User Having a Separate Copy of the Programs

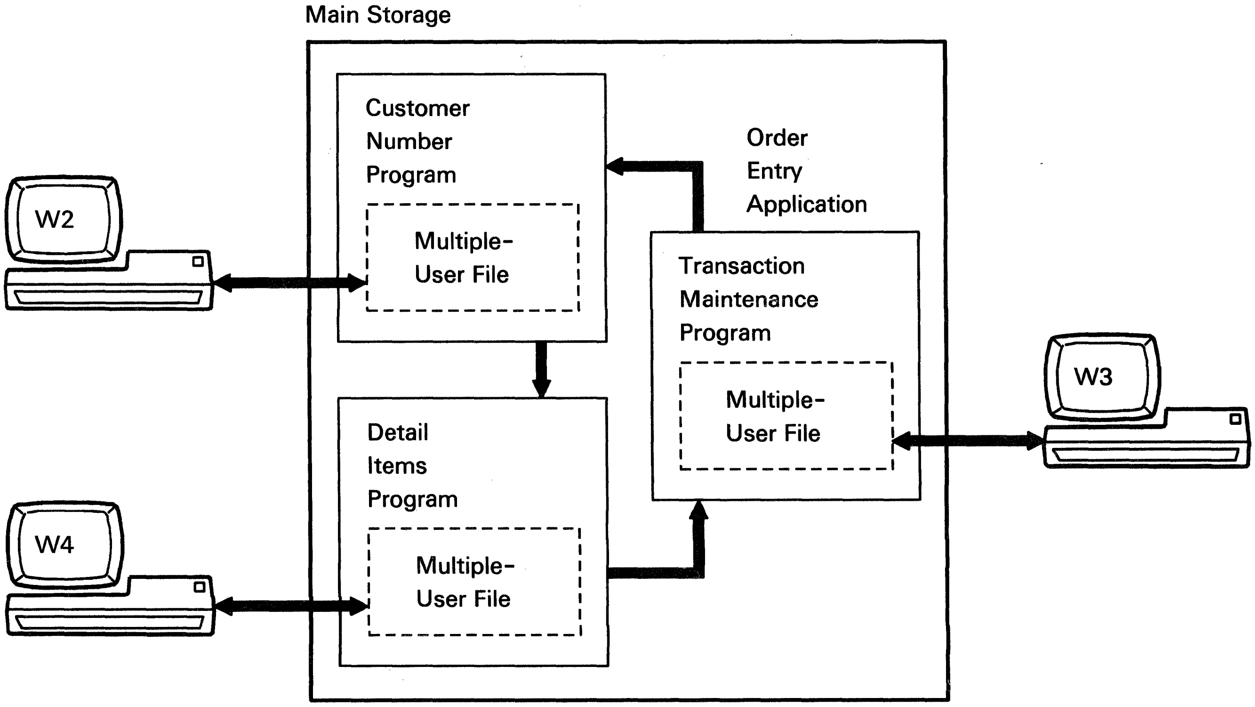
An application that consists of one large program for the entire application has two disadvantages. First, it is probably very difficult to code properly. Second, if the number of concurrent users becomes too large, the performance advantage may be lost.

To retain the simplicity of coding and the performance advantage, you can divide the application into several programs. Even though a separate copy of any one program exists in main storage for each user, the average amount of main storage required for each user is far less than the amount required for one large program for each user.



Several Small Programs, All Users Sharing One Copy of the Programs

A large program with many concurrent users may have poor performance if it includes many logical steps and if it takes a long time between successive input operations. An application that consists of several small programs, one set for the entire application, reduces these performance problems because each program has fewer users at any given time. Such an application should also be simpler to code than one large program for the entire application.



S9019081-0

A Mixture of the Two Preceding Possibilities

If you create a separate program for each application step rather than one large program for the entire application, you may want to use some small programs with a separate copy for each user and some small programs with one copy shared by all users. You would probably want a single copy of the programs used most often, and a separate copy for each user of programs used less often. Such a mixture provides both simplicity of coding for infrequently used programs and performance advantages for frequently used programs. For example, in an order entry application, the program that enters the orders would be used frequently and would be coded as a single copy shared by all users. On the other hand, a program that prints invoices only at the end of the day would be used infrequently and would, therefore, be coded as a separate copy for each user.

Number of Users of Each Program

If you decide to use one large program for the entire application or some small programs with a single copy shared by all users, you must decide how many users can share that single copy. The decision about the number of users can be based on the number of display stations available for a given application. The decision can also be based on observed performance. For example, you may have noticed that response times increase significantly when the number of users exceeds a given number.

One User: The user of a one-user program could be either the requester or a display station acquired by the program for input or output.

The main advantage of one-user programs is simplicity of program design. The program handles only one transaction at a time, whereas a multiple-user program must be able to handle several transactions at a time. Handling multiple transactions concurrently can increase the complexity of the program.

Because a one-user program is less complex, it usually uses less main storage per copy than a multiple-user program for the same application.

Multiple Users: The main advantage of multiple-user programs is performance. Because multiple-user programs can be initiated or terminated only once a day, they generally run faster than one-user programs.

Another advantage is that, when the application is run from two or more display stations at the same time, a multiple-user program uses less main storage. As a result, there is less contention for system resources and, therefore, improved performance.

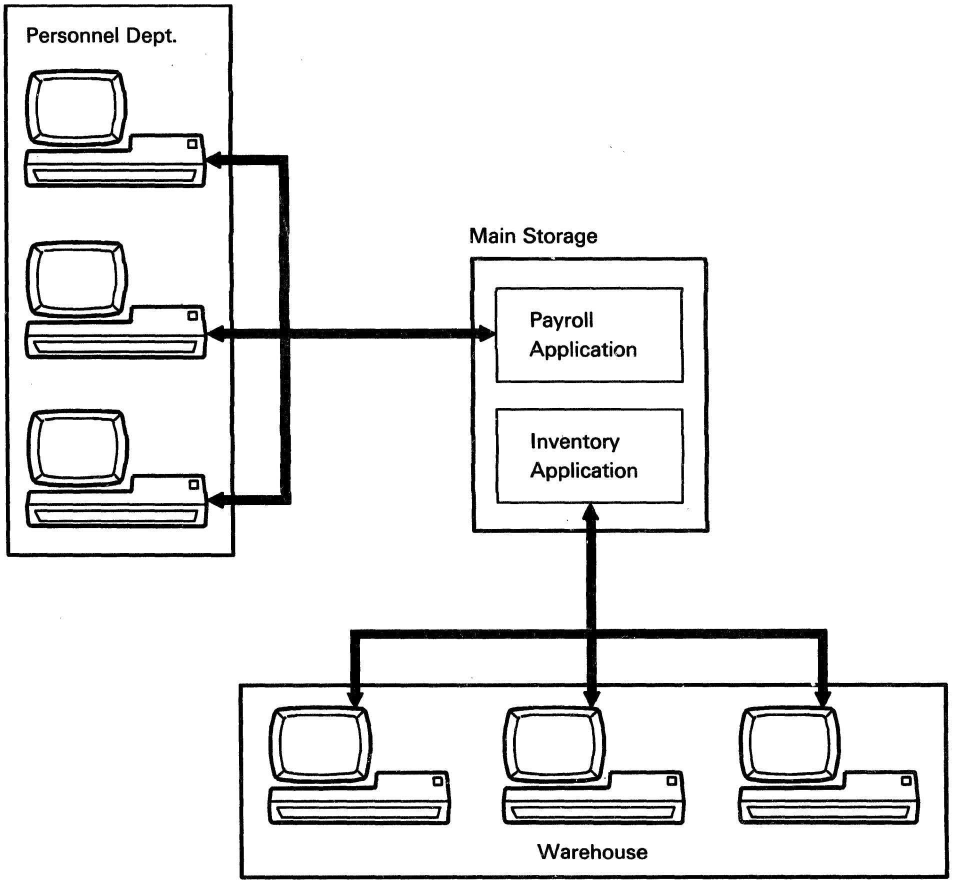
Occasionally, a multiple-user program may be advantageous because a single copy of it can manage a nonshared file more efficiently. Sharing a file with other programs uses more system resources than using a nonshared file. Sharing a file with other users of the same program may require you to keep tables of information for each user. The program could use this information to tell other users that a requested record in a shared file is being used. This information can help prevent a file deadlock for a multiple-user program. For information about file deadlock, see "File Deadlock Conditions" on page 8-84.

Sharing a file with other programs requires the other users to wait an unpredictable length of time whenever two or more programs attempt to access the same record in a shared file.

No User: A no-user program cannot interact with a display station. An example of a no-user program is a program in an order entry application that is evoked to update an inventory file while the operator is entering the next order.

Number of Requesters and Acquired Display Stations

If you decide that more than one user should be able to interact with the program, you must also decide how many of the users can request the program. Normally, all users are requesters. However, sometimes acquiring display stations is an advantage because acquiring them allows the application to control who and where its users are. For example, suppose a company has display stations in two or more separate locations, as shown in the following:



S9019082-0

In this example, the payroll application is run from the display stations in the personnel department, and the inventory application is run from the warehouse, which may be across the hall or across the country. Having the program acquire the display stations provides a form of security that does not depend on user identification. The payroll application can be run only from the display stations in the personnel office, and the inventory application can be run only from the display stations in the warehouse.

Another case for acquiring a display station involves two separate computers that communicate by SSP-ICF. If one computer is running an order entry application, it might need to obtain inventory status information from the other computer. If the program finds that the inventory on hand is insufficient to fill an order, it acquires an SSP-ICF session, evokes a program in the second computer, and receives a response about inventory on hand at the second location.

Summary of Differences between SRT and MRT Programs

Having read the preceding explanation of one-user and multiple-user SRT and MRT programs, you probably foresee each of your applications as being one or a combination of the following general types of programs:

- Large one-user SRT. The entire application consists of a single program. Each requesting device has a separate copy of the program. This is the most straightforward type of program. This type is good if the program is simple and run infrequently, but performance tends to be poor if the program is run frequently and the system is busy.

- Small SRT. The application consists of several small application steps instead of one large program. Each requesting device has a copy of one of the application steps at a given time. More than one copy of a program segment can be active at a given time. This type of program is the simplest and the easiest to maintain. It gives reasonable performance except when the system is very busy.

- Large multiple-user program (either a large SRT that acquires devices or a large MRT). All users share the same copy of the program. This type of program may give the best performance, but it may be the most complex and the most difficult to maintain.

- Small MRT. The application consists of several small application steps. Each requesting device is attached to a copy of one of the application steps at a given time. No more than one copy of any application step is active at a given time. This type of program gives good performance and is relatively simple to design and maintain.

The following summary lists the advantages in general terms of simplicity, maintenance, and performance for each of these four types. The comparison of performance assumes a busy system.

Small means that the average program size is less than the average amount of main storage user space available for each program, assuming that main storage is divided equally among all the users. For example, if you have 80K bytes of user space available and five active display stations, 16K bytes could be allocated for each user. A small program decreases the probability of swapping.

Large means that the average program size is much larger than a small program. Therefore, a large program significantly increases the probability of swapping.

Program Type	**Simplicity**	**Maintenance**	**Performance**
Large One-User SRT	More complex than a small SRT. Much simpler than a large multiple-user program. Probably more complex than most small MRTs.	More difficult to maintain than a small SRT. About as difficult to maintain as a large multiple-user program. More difficult to maintain than a small MRT.	Comparable to several small SRTs. Poorer than a large multiple-user program. Somewhat poorer than several small MRTs.
Small SRT	Somewhat simpler than a large SRT. Much simpler than a large multiple-user program. Simpler than a small MRT.	Easier than a large SRT. Easier than a large multiple-user program. Could be easier than a small MRT.	Comparable to a large SRT. Poorer than a large multiple-user program. Poorer than several small MRTs.
Large Multiple-User SRT or MRT Program	More complex than a large SRT. Much more complex than a small SRT. More complex than a small MRT.	As difficult as a large SRT. More difficult than several small SRTs or MRTs.	Usually better than a large SRT. Usually better than several small SRTs. May be better than several small MRTs.
Small MRT	May be less complex than a large SRT. Slightly more complex than several small SRTs. Simpler than a large multiple-user program.	Easier than a large SRT. Slightly more difficult than several small SRTs. Easier than a large multiple-user program.	Usually better than a large SRT. Usually better than several small SRTs. Usually slightly poorer than a large multiple-user program.

Program Attributes

Any program can have the following user-assigned attributes:

- Never-ending
- Inquiry

Never-Ending Programs

The never-ending program (NEP) attribute implies that the program is going to run for a long time. Therefore, when the NEP owns a nonshared file that a second program attempts to use, the system issues an error message to the second program. The error message enables the operator to retry or cancel the job. If the file owner is not a NEP, the second program waits for the file to become available. If the running program is active for a long time, the second program will wait a long time. A MRT program, a program on the job queue, or a program run by the EVOKE OCL statement is treated as though it were a NEP for file sharing (unless otherwise specified by NEP-NO on an ATTR OCL statement).

For a MRT program, the NEP attribute has additional meaning. When a MRT program is not a NEP and releases its last requester, the program is given a return code instructing it to go to end of program. At that point, the program is in effect no longer an active MRT. Therefore, if another device requests that program, a new copy is initiated. If the MRT program does not terminate rapidly, the NEP attribute may cause the new copy to wait until the terminator frees the resources assigned to the MRT program.

When a MRT program is a NEP, it is called a MRT-NEP and it is not instructed to go to end of program when the last requester is released because, by definition, the NEP is expected to run for a long time and presumably does not want to know when it temporarily has no requesters.

The NEP attribute can be canceled by using the STOP SYSTEM command. Thereafter, all MRT-NEP programs are instructed to go to end of program when they release their last requester.

One reason for assigning the NEP attribute to a MRT program is so that the program has consistently good response time.

Inquiry

The inquiry attribute allows you to specify whether the program can be interrupted so that another program can be run from the same display station. The inquiry attribute specifies whether Inquiry display option 1, which calls the command display, is permitted.

You can use the ATTR OCL statement to specify the inquiry attribute; see "Preventing Jobs from Being Canceled or Interrupted" on page 17-34 for more information. In RPG, the inquiry attribute can also be specified in column 37 of the control specification. In BASIC, you can prevent inquiry from a SRT program by coding ON ATTN IGNORE. The inquiry attribute cannot be specified in the other high-level languages.

It is possible to code a MRT program that causes all other requesters to wait indefinitely. When a MRT program writes more than once to a display station before allowing input, for each output except the last you should specify suppress input in columns 35 and 36 of the S specification for the $SFGR utility. If you do not specify suppress input, the program and all other devices requesting it may wait indefinitely if the operator presses the Attn key.

Programming Considerations

This section contains considerations that might help you code more efficient programs. This section has three parts, organized from most general to most specific:

- Programming considerations for any program
- Programming considerations for multiple-user programs
- Programming considerations for MRT programs

Programming Considerations for Any Program

The following topics apply to all programs:

- Acquiring a display station
- Releasing a display station
- File sharing
- Transaction file design
- Memo updating
- Printed output
- Inquiry menu options
- Calling the program
- Read under format
- External switches
- Local data areas

Acquiring a Display Station

To be acquired, a display station must be showing the Standby display (the word STANDBY appears in the upper left corner). If the operator is using a command display station, the operator can enter the MODE command to show the Standby display. You can also use the CNFIGSSP procedure to cause a display station to be a data display station, which can then be acquired. The *System Reference* manual has information about the MODE command. The *Changing Your System Configuration* manual has information about the CNFIGSSP procedure.

There are two ways that a program can acquire a display station. The first way is to use the WORKSTN OCL statement with REQD-YES and a display station identification specified. The second way is to use the programming language statement that acquires a display station. The following table lists the appropriate statement for each programming language:

Programming Language	**Statement for Acquiring Display Station**
Assembler	$WSIO macroinstruction with OPC-ACQ
BASIC	OPEN statement with display station ID
COBOL	ACQUIRE statement
FORTRAN IV	Cannot acquire display stations
RPG II	ACQ operation code

For more information, refer to the appropriate language manual.

Note: *In RPG, if REQD-YES is specified on a WORKSTN OCL statement or if REQD-YES is not specified and a display station is available, the display station appears to be a new requester with blank input fields when the display station input operation occurs. You can use the return code in the INFDS to determine whether the display station is a requester or an acquired display station.*

Releasing a Display Station

Display stations should be released from a program when the program is no longer using them. A SRT program cannot release its requester, but it can release acquired display stations.

A MRT program can release both requesters and acquired display stations. If a MRT program goes to end of job before releasing a requester, the requester terminates abnormally. The following table lists the statement for releasing a display station in each programming language:

Programming Language	Statement for Releasing Display Station
Assembler	$WSIO macroinstruction with OPC-REL
BASIC	CLOSE statement
COBOL	DROP statement
FORTRAN IV	Cannot release display stations
RPG II	REL operation code in calculation specifications R in column 16 of output specifications

For more information, refer to the appropriate language manual.

File Sharing

When a file is shared, two problems must be prevented: one is loss of data, and the other is file deadlock.

Data can be lost when two or more users update the same record of a file shared within a single copy of a program. When the program writes the second updated record back to the file, that record is written over the first updated record. Therefore, the first update is lost.

File deadlock can occur when two or more programs try to update records in two or more files at the same time. For information about file deadlock, see "File Deadlock Conditions" on page 8-84.

One way to prevent file deadlock is never to own more than one record at a time. Each time you read a record from a file shared for update, write the record back to the file before you read a record from another file shared for update. This technique may have a significant effect on how you design your application because you may have to divide the application into several small programs.

A second way to prevent file deadlock is to design your application so that each shared file is accessed by only one program. This technique also requires you to write an updated record back to the file before you read another record.

This technique also prevents loss of data because one update cannot be partially completed while another update of the same record occurs. For example, if two display station operators enter orders for the same inventory item at the same time, the second update may replace the first update. To prevent both loss of data and file deadlock, treat each display station input as a separate transaction. Do not assume that values that were valid at the previous display station input are still valid for the current display station input.

Transaction File Design

A transaction file contains data, such as customer orders, that is usually used to update a master file.

*Note: This transaction file is not the same as a COBOL TRANSACTION file, which is an input/output file used to communicate with display stations and SSP-ICF sessions. The manual **Programming with COBOL** has information about COBOL TRANSACTION files.*

There are two basic ways to design your transaction file:

- Use a single file for all transactions
- Use a separate file for each device

The method you choose will likely depend on how the transaction file is to be used and how you plan to recover from errors. For information about error recovery, see Chapter 19, "Error Prevention, Detection, and Recovery." For more information about transaction files, see "Transaction File" on page 8-44.

Single File for All Transactions: If you decide to use a single file for all your transactions, you still have another decision to make about how that file will be organized. You can either put all the transactions in the file in the order in which the transactions arrive, or you can put the transactions for each device in a separate part of the file. For example, you could allocate 3000 records for the file, partitioned as follows:

Record 1	Control record for W1 (display station 1)
Records 2-1000	Data records for W1
Record 1001	Control record for W2 (display station 2)
Records 1002-2000	Data records for W2
Record 2001	Control record for W3 (display station 3)
Records 2002-3000	Data records for W3

Separate File for Each Device: The second way to design a transaction file is to assign a separate file to each device. Thus, for the preceding example, you could assign a separate file, each containing 1000 records, to each device. Because the FILE OCL statement is read only for the first requester of a MRT program, this approach requires a separate FILE statement for each user.

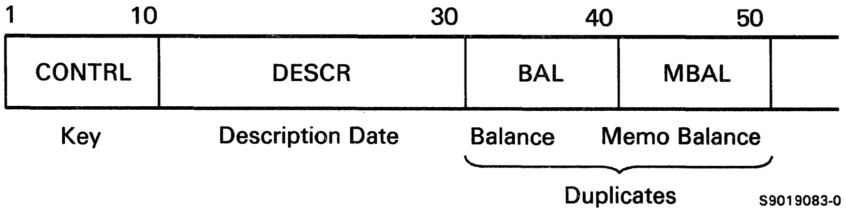
Memo Updating

An advantage of an interactive environment is that operators always have access to up-to-date information in the master files. For example, suppose an operator enters a transaction that reduces the quantity on hand of an item in an inventory master file. If another operator inquires for the quantity of that item on hand, he can see the value that reflects previous changes made to it.

Interactive updates to files should be done carefully because recovery from a system or program failure can be difficult if you do not know which updates are reflected in the file and which updates need repeating.

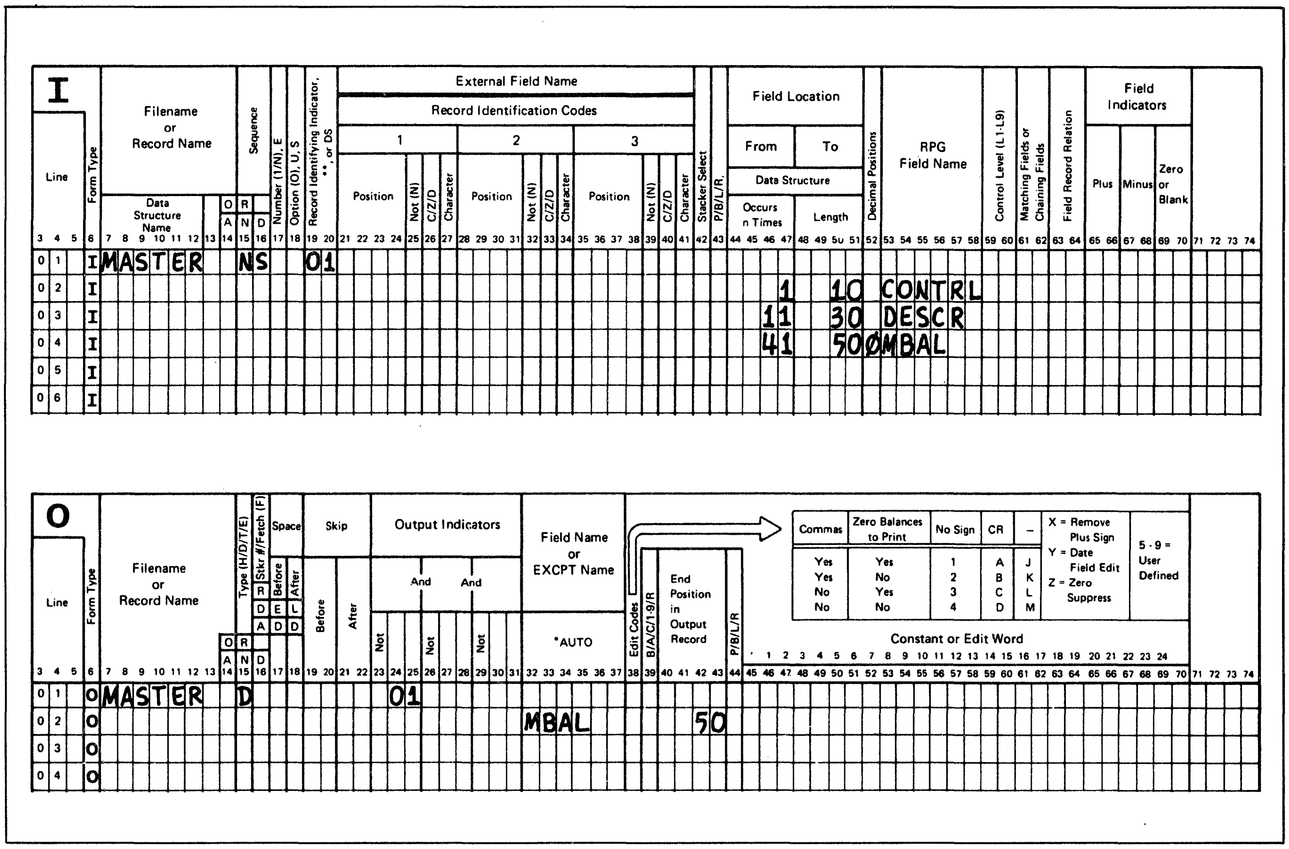
Memo updating is a technique that allows interactive updates to your master files and provides batch processing to check that the updates have been applied correctly.

For this technique, master file records must allow duplicate fields for those fields that can be updated interactively. For example, the field named MBAL (memo balance) could reflect interactive updates, and the field named BAL (balance) could be used for batch processing.



Initially (for example, at the beginning of the day), these two fields should be equal. The transactions made during the day are applied only to the memo balance field.

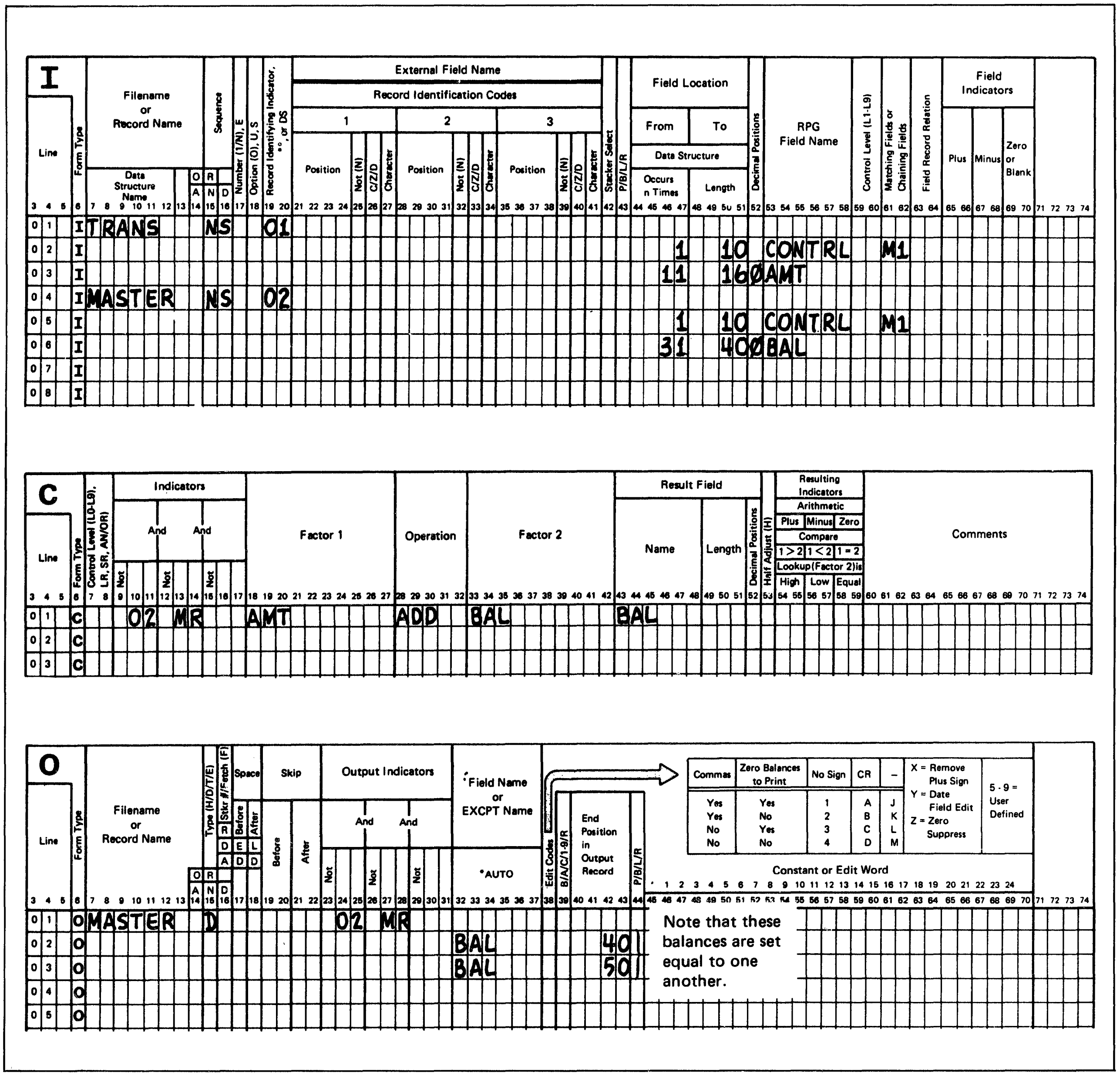
The following RPG input specifications and output specifications could be used for the master file by interactive data entry and inquiry programs. Notice that these specifications ignore the balance field. The memo balance field should always reflect the current balance.



```
  01 IMASTER  NS  01
  02 I                                          1  10 CONTRL
  03 I                                         11  30 DESCR
  04 I                                         41  500MBAL
  05 I
  06 I
```

```
  01 OMASTER  D        01
  02 O                         MBAL    50
  03 O
  04 O
```

Later (for example, at the end of the day), the transaction file is processed by a batch edit program. The transactions are posted to the balance fields in the master file by a batch update program, as the following segment of the program shows:



```
01 ITRANS   NS  01
02 I                                        1  10 CONTRL  M1
03 I                                       11  160AMT
04 IMASTER  NS  02
05 I                                        1  10 CONTRL  M1
06 I                                       31  400BAL
07 I
08 I
```

```
01 C   02 MR   AMT       ADD  BAL       BAL
02 C
03 C
```

```
01 OMASTER  D        02 MR
02 O                         BAL       40
03 O                         BAL       50
04 O
05 O
```

Note that these balances are set equal to one another.

The transaction files must be backed up periodically, usually each day. The master files can be backed up frequently if the file is constantly being updated. Recovery can be done by reloading the master files and processing all subsequent transactions. (For information about file recovery, see Chapter 19, "Error Prevention, Detection, and Recovery.") To bring the memo balance field to its current value, run a program that updates the memo balance field with the transactions. After the memo balance field has been updated, all current activity has been accounted for and normal operations can continue.

A variation of the memo updating technique is to set the memo balance field to 0 at the start of the day rather than to the value of the balance field. As for the previous method, interactive updates would be made only to the memo balance field.

The memo balance field would reflect the day's activity for that item. If no transactions for the item occurred, the memo balance would remain 0. In order to determine the current balance, an inquiry program would have to add or subtract the memo balance from the balance in the master file.

Printed Output

If the application does not have exclusive use of the printer, you must decide whether the output can wait until the application is completed or should be printed as soon as it is available.

If you want the printed output as soon as possible without having exclusive use of the printer, you must decide whether to use the PRINTER OCL statement to defer printing. For example, if you are printing invoices and you specify DEFER-NO on the PRINTER OCL statement, you might cause the printer to be unavailable to anyone else for a long time. On the other hand, if you specify DEFER-YES, you cannot print until your job is completed.

To have invoices printed as soon as they are available and still have the printer available for other jobs, design the application so that the print program runs as required. A good way to do this is to use the EVOKE OCL statement to call the print step, which is therefore a NRT program.

The advantage of this technique is that it runs only when the output is ready to print. Also, the operator can continue processing because, as soon as the NRT initiates, the application continues with the next step. If the print step is not a NRT, the operator has to wait until the program initiation, allocation, and termination, as well as the printing, are complete.

The disadvantage of making the print program a NRT program is that the spool file may become overloaded with numerous, small entries.

Inquiry Menu Options

When you press the Attn key, the inquiry menu is displayed. The three options that you can select from that menu differ, depending on whether your program is a SRT or a MRT. Those three options are 2, 3, and 4.

For a SRT program:

- Option 2 terminates the job and closes files.
- Option 3 terminates the job. The system retains data created by previous job steps. Any records that were added or updated to existing files by this job step are also retained. Records that were deleted by this job step remain deleted. However, any **new files** created by this job step are lost.
- Option 4 is used by high-level languages to perform special processing functions. See the appropriate language manual for more information.

For a MRT program:

- Options 2 and 3 have no effect on the program. They release the requester from the program. Option 2 sends the requester to the next step in the job stream, whereas option 3 terminates the job. You can use ?CD?=3721 on a procedure control expression to test whether the MRT job step was terminated by option 2.
- Option 4 is not allowed.

If INQUIRY-NO is specified on the ATTR OCL statement for either the MRT program or the procedure that called the program, the MRT step cannot be interrupted by using option 1.

If CANCEL-NO is specified on the ATTR OCL statement for either the MRT program or the procedure that called the program, the MRT step cannot be canceled by using options 2 or 3.

Calling the Program

A program can be called in several ways. The operator can:

- Enter the OCL statements at the display station
- Enter the procedure name at the display station
- Select a menu option at the display station

An SSP-ICF session can call a procedure.

A MRT program can be called only from a MRT procedure. The system checks by procedure name to see whether the MRT program is already active.

When you create a procedure, you can specify that the data following a procedure name is either data for the program or parameters for substitution in the OCL statements. For MRT programs, the procedure can be called only with data, not with parameters for substitution. Anything following the MRT procedure name is saved until the MRT program does its first input operation. The *SEU Guide* has more information about calling the procedure with data. A description of the PDATA parameter is on the SEU end-of-job display, or on the $MAINT utility in the *System Reference* manual.

Read under Format

The read-under-format technique allows an operator to enter information on a display while the program that uses the display is initiating. When the read-under-format technique is used, a program or procedure displays the format, and the program called next in the procedure reads it. The format is displayed by a program or a PROMPT OCL statement with PDATA-YES specified. If a SRT program displays the format, it then goes to end of job. A MRT program displaying the format releases the display station. While the next program is being initiated, the operator enters information for the display. When the operator presses the Enter key, the input from the display is sent to the second program.

For more information about this technique, see "Using the Read-Under-Format Technique" on page 13-33.

External Switches

Eight external indicators (switches) are available for each requester. You can set or change these switches by using the SWITCH OCL statement, the SWITCH procedure, or the PROMPT OCL statement with UPSI-YES specified. By setting these switches, you can affect how your job is processed by the system. For example, if a certain error condition occurs, you can set a switch on and bypass those job steps that are in error.

The following table shows how to access these switches in various high-level languages:

Programming Language	**How to Access External Switches**
Assembler	$INFO macroinstruction
BASIC	UPSI$ intrinsic function
COBOL	IF and SET statements
FORTRAN IV	Data switch subprogram
RPG II	U1-U8 or SUBR20

The following example uses the SWITCH OCL statement to affect how the system processes a job stream. In the example, the SWITCH statement sets switch 1 to 1 (on). Switch 1 determines whether the program is a daily or weekly run. In the procedure, a FILE OCL statement for the daily run differs from the FILE statement for the weekly run.

```
// IF SWITCH-1 FILE NAME-A
// ELSE FILE NAME-A,LABEL-B
```

When switch 1 is on, file A is used. When it is off, file B is used.

For a MRT program, a separate copy of the switch settings is kept for each requester's job stream, and control is passed to the next job step when the requester is released. Also, the switch settings are normally the first requester's settings. Some high-level languages allow you to access a specific requester's switch settings.

The *System Reference* manual has more information about using the external switches.

Local Data Areas

A local data area (LDA) is 512 bytes that can be used to receive data from previous job steps or to pass data to later job steps. You can use the LOCAL OCL statement to change the LDA. Local data can be substituted in OCL statements and procedure control expressions.

For a MRT program, a separate copy of the LDA is kept for each requester's job stream, and control is passed to the next job step when the requester is released. Also, the local data area is normally the first requester's data. Some high-level languages allow you to access a specific requester's local data area.

Whenever a program is placed on the job queue, called by the EVOKE OCL statement, released by the ATTR OCL statement, or called during inquiry, the external switches and the LDA have the value that was in effect at that time.

Each high-level language has a separate way of accessing the LDA. The following table shows how to access the LDA in various high-level languages:

Programming Language	How to Access LDA
Assembler	$INFO macroinstruction
BASIC	LOCAL, a 512-byte file
COBOL	ACCEPT and DISPLAY statements
FORTRAN IV	Local data area subprogram
RPG II	SUBR21 or special data structure

Programming Considerations for Multiple-User Programs

If you decide to use a multiple-user program, you should also be aware of the following considerations:

- Creating a table of separate variables
- Sequential processing of multiple records with duplicate keys
- Changing a one-user program to a multiple-user program
- Response time

Creating a Table of Separate Variables

Because a multiple-user program can handle more than one transaction at a time, you may require separate variables and work areas for each active display station. For example, you may need separate copies of:

- Indicators, flags, or switches
- Current record identification for each disk file
- Current display format

WSU automatically creates these tables as part of the program generation process. The *WSU Guide* has more information.

To keep a multiple-user program relatively simple and easier to maintain, try to design the program such that an entire transaction is completed between two successive display station input operations. This eliminates the need for saving the above information.

Indicators, Flags, or Switches: Because a multiple-user program may process transactions that require several display station input operations, it may be necessary for the program to switch from one display station to another before completing a transaction. Therefore, the program may have to store the status of each display station whenever the program returns to the input operation for the display station file. Usually, you can code an array or table in which each element consists of the display station identification and the appropriate status fields.

Current Record Identification for Each File: Among the status fields in the table elements could be the key or record number for the last record processed in each disk file. When the program is reading duplicate keys sequentially and not all of the keys have been processed, the first of the series may have to be saved.

Each time the program reads from a display station file, it should search the table for an element that matches the display station identification. If the program finds a match, it uses the element for that identification. If it does not find a match, it should allocate a new element. The program should reinitialize the identification field when it releases the display station.

Another method for maintaining multiple current record IDs (one for each work station) is to define multiple logical files. One logical file could be defined for each work station using the file. For information about how to define multiple logical files, see "Using One File As Two Or More Logical Files" on page 8-89.

When a disk file is shared, the table of variables should not include current record identification because the only valid file information is that which you read after the most recent display station input. For more information, see "Sharing Files" on page 8-73.

Current Display Format: For similar reasons, you may want your program to store the last operation for each display station. The operation could include the display format name.

This table element could also include fields for such variables as local data area, external switches, and separate totals.

Note: Keeping track of these variables can be complicated, because completion of a transaction might require several input operations from a display station, and those input operations might not be consecutive. Therefore, if possible, you should try to avoid relying on tables of variables between display station input operations. To avoid using tables, you may be able to obtain all the necessary variables from the preceding display format. For example, a part number or customer number might be on the preceding display format. To use these fields, you may have to change them from output-only to input/output fields. You may also have to write as nondisplayed fields other information that otherwise would be stored in the table of variables. This method also requires you to read the master file again for each display station input operation, which is a good technique.

Sequential Processing of Multiple Records with Duplicate Keys

When you use the generalized access method to process multiple records with duplicate keys in one file shared among users of the same program, you may need to keep track of which record you last processed for each user so that you can continue processing from that point. Each time you access the file for a given key, you access the first record with that key value. If other records have the same key, you must continue reading the file sequentially until you find the record you want. If you must interrupt the sequence of read operations to read from the display station file, you might not be able to find the last record you processed unless you create a table to store the last record processed for each file by each display station. For information about the generalized access method, see "Generalized Access Method" on page 8-41. For information about duplicate keys, see "Duplicate Keys" on page 8-27.

Changing a One-User Program to a Multiple-User Program

To change a one-user program to a multiple-user program, you may have to change the program logic. In a one-user program, there is no need to keep separate copies of variables and work areas for each device. In a multiple-user program, a table or array is usually necessary for this purpose.

If the program is so simple that you always complete a transaction before receiving input for the next transaction, such as a read-only inquiry program, it probably requires no additional logic as a multiple-user program than as a one-user program, except to release the requesters.

Usually, you would be converting a SRT to a MRT, and the logic would have to be changed to account for multiple concurrent transactions. To convert a SRT to a MRT, you must also change the response to the prompt about the maximum number of requesters on the procedure used to compile the program.

The general steps to change a one-user program to a multiple-user program are:

1. Create a data structure to save unique information required by individual display stations from one cycle to another if necessary. The data structure search argument should be the display station identification.

2. Always reread the disk record to be updated if a display station input operation occurred after the previous disk read.

3. Make sure that the program logic can handle the maximum number of users.

Response Time

If a multiple-user program has considerable input/output or processing, the average response times for the display station operators could become poor if there are many concurrent users. For this reason, a technique often used to ensure reasonable response time is to group the display stations. This multiple-user program is usually a MRT program, so each group has a separate copy of the MRT program. This technique reduces the number of display stations trying to use the program. One way to group the display stations using a MRT program is to use the NEWNAME parameter on the LIBRLIBR utility to create a new copy of the MRT procedure within the same library. Then, to ensure the response time, you can use the MRTMAX parameter to limit the number of display stations using each copy.

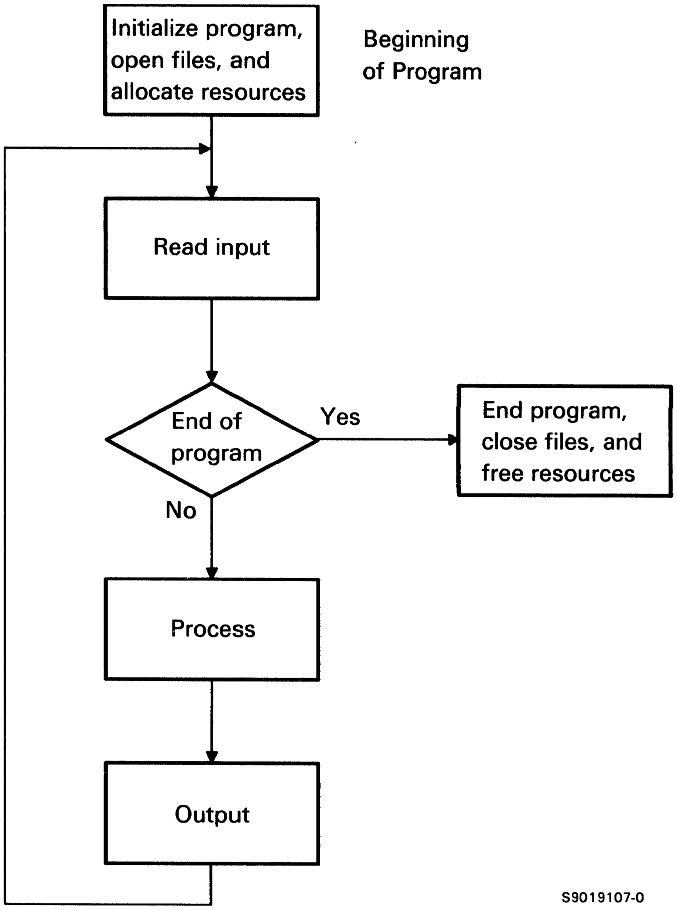
Programming Considerations for MRT Programs

The following considerations apply primarily or exclusively to MRT programs. These additional considerations include:

- Job stream
- Modular applications
- MRTMAX value
- Limiting the number of users
- First-requester considerations
- Summary of MRT considerations

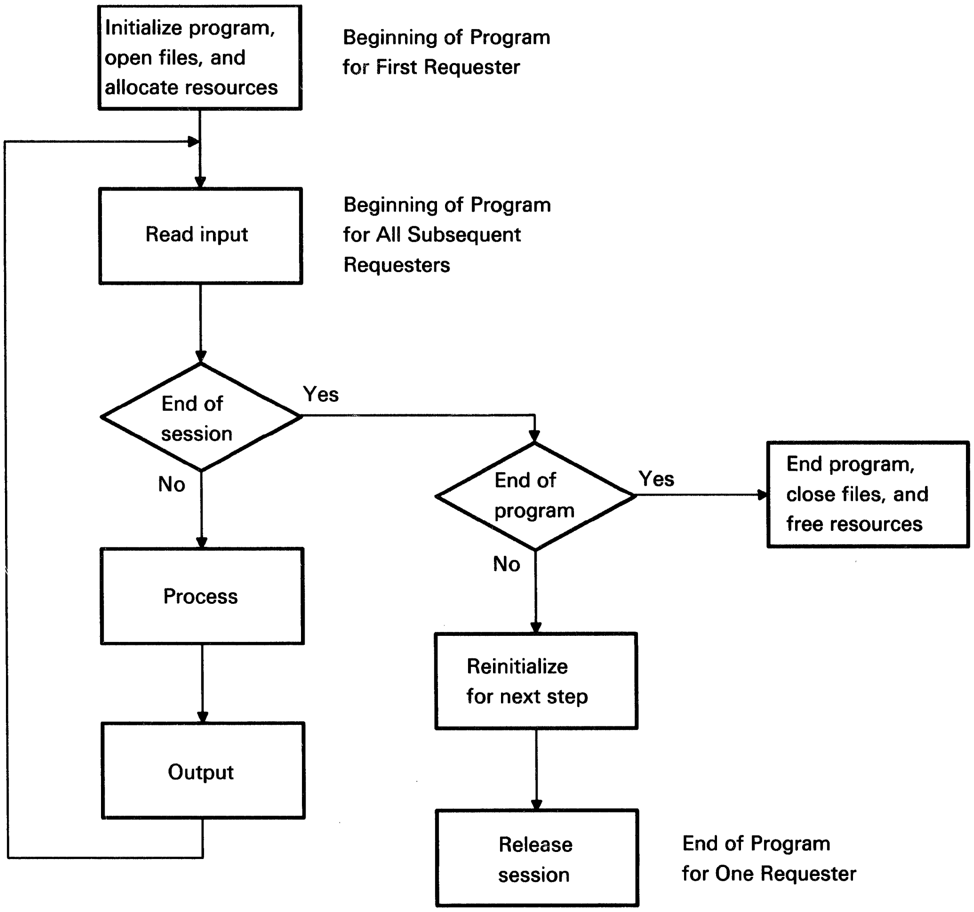
Job Stream

The following diagram represents a normal program flow:


Initialize program, open files, and allocate resources
Beginning of Program
Read input
End of program
Yes
End program, close files, and free resources
No
Process
Output
S9019107-0

A MRT program has this same flow, except that:

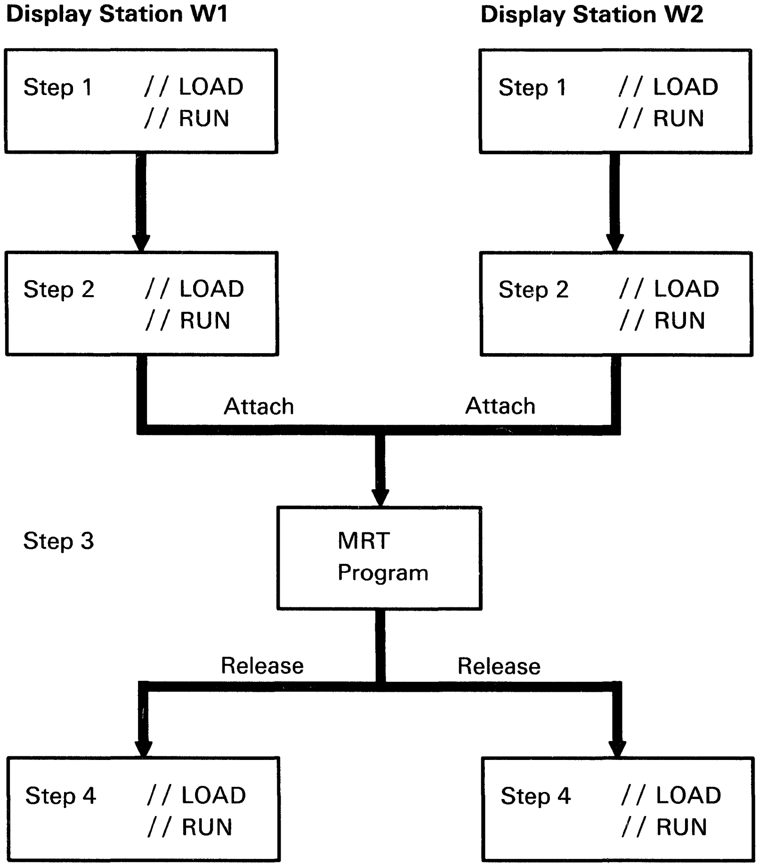
- All but the first requester start by reading input
- All requesters end by releasing the session



Initialize program, open files, and allocate resources
Beginning of Program for First Requester
Read input
Beginning of Program for All Subsequent Requesters
End of session
Yes
No
Process
Output
End of program
Yes
No
End program, close files, and free resources
Reinitialize for next step
Release session
End of Program for One Requester

S9019106-0

If you design an application that interacts with two or more display stations, there is a separate job stream for each display station. The job stream consists of a series of steps, each step including LOAD and RUN OCL statements, typically FILE statements, and possibly other OCL statements. When a step is a MRT, the MRT is normally already active, so the system merely attaches to the MRT program rather than going through the complete program initiation that would be required for a non-MRT. When the step is complete for that job stream, the MRT merely releases the requester rather than going through a time-consuming program termination.



S9019084-0

For additional information about jobs and job processing, see Chapter 17, "Jobs and Job Processing." For additional information about procedures, see Chapter 18, "Procedures."

Modular Applications

If you modularize a large application into a set of small, simple programs, the performance might be poor if the programs are SRTs because of the time for initiation and termination. The performance is probably better if the programs are MRTs.

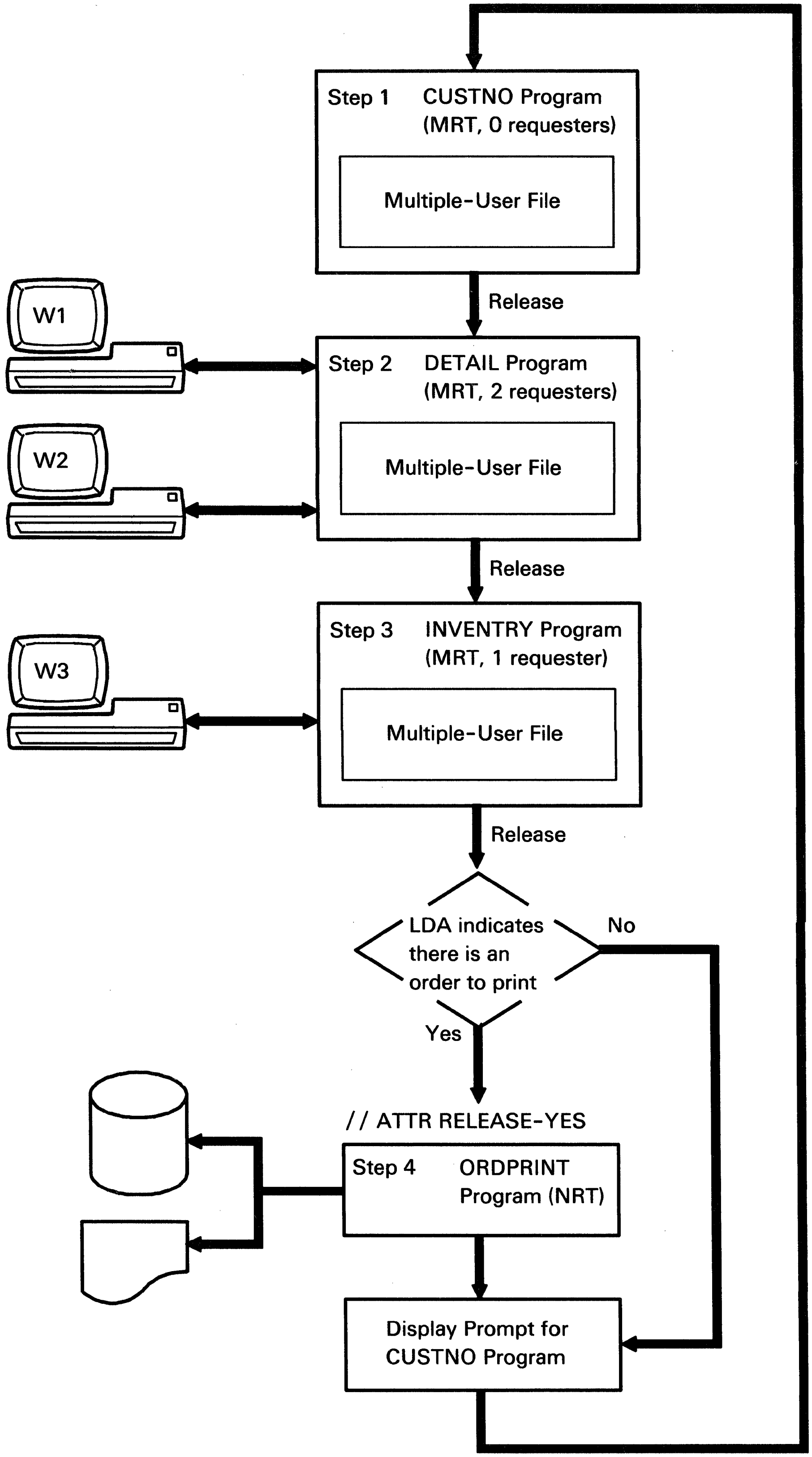
In the following example, an order entry application is divided into four steps:

1. Read the customer master file (CUSTNO program)

2. Enter the order (DETAIL program)

3. Update the inventory file (INVENTRY program)

4. Print the invoice and picking slip (ORDPRINT program)


Step 1 CUSTNO Program (MRT, 0 requesters)
Multiple-User File
Release
W1
W2
Step 2 DETAIL Program (MRT, 2 requesters)
Multiple-User File
Release
W3
Step 3 INVENTRY Program (MRT, 1 requester)
Multiple-User File
Release
LDA indicates there is an order to print
No
Yes
// ATTR RELEASE-YES
Step 4 ORDPRINT Program (NRT)
Display Prompt for CUSTNO Program

S9019005-0 S9019085-0

This modular design is easier to maintain than one large, complex, program would be. Moreover, because the devices merely attach to and release from already active programs, the MRT programs have acceptable response times.

The procedure for this example could be coded as follows:

```
// TAG  TAGB                 Start of procedure.
// PROMPT format name        Initial order entry format.
CUSTNO                       Procedure to read customer file.
DETAIL                       Procedure to enter order.
INVENTRY                     Procedure to update inventory file.
// IF ?L'1,4'?/0 GOTO TAGA   Test for successfully completed order.
*                            First 4 bytes of LDA are set to nonzero
*                              value when order is canceled.
// GOTO TAGB                 Start next order.
// TAG TAGA                  Normal processing continues here.
// ATTR RELEASE-YES          Create NRT program to print invoice
// LOAD ORDPRINT               and picking slip while beginning
// RUN                         to process next order.
// GOTO TAGB                 Process next order.
```

In this example, the print job step is shown as a NRT program in order to ensure the best performance for the procedure and maximum availability of the printer.

MRTMAX Value

A MRT can be either a one-user or multiple-user program. The MRTMAX value is specified on the procedure that compiles your program, and it can be decreased on the ATTR OCL statement.

If a MRT is a one-user program, it must have a MRTMAX value of 1. A MRTMAX value of 1 can be used to ensure that only one copy of a program is active at a given time. This technique is sometimes used to control the sharing of a resource.

If your program already has the maximum number of requesters and another display station or SSP-ICF session requests it, the new requester must wait until a previous requester is released. You can test whether the MRTMAX value has been reached with a procedure control expression. You can use an IF MRTMAX procedure control expression that could issue a prompting message and try the IF test again before calling the MRT. If the problem becomes persistent, you can raise the MRTMAX value or use separate copies of the program. The *System Reference* manual has information about procedure control expressions.

Indicating the Run Time of MRTMAX

You can use the MRTWAIT value in the ATTR OCL statement to indicate the run time of MRTMAX.

The MRTWAIT value is specified in the procedure that compiles your program and can be changed on the ATTR OCL statement.

If you specify MRTWAIT-YES, and a request causes the number of active users to exceed the MRTMAX value for the MRT, you will wait until the MRT is attached.

If you specify MRTWAIT-NO, and a request causes the number of active users to exceed the MRTMAX value for the MRT, you will get back control and return code 2045 is issued. This return code can be tested by using the ?CD? substitution expression. You **must** specify MRTWAIT-NO in a procedure. It will be ignored if entered anyplace else.

Each time the MRT is processed, the value is reset. The //ATTR statement with MRTWAIT-NO is required before each invocation of the MRT procedure in order to use this function.

If an invalid MRTWAIT parameter is specified, a system error is issued.

The following is an example of MRTWAIT-NO specified in a procedure.

```
   //ATTR MRTWAIT-NO
MRTPROC
//IF ?CD?=2045 GOTO PROC2   /*TRY PROC2*/
//RETURN
//TAG PROC2
//ATTR MRTWAIT-NO
//IF ?CD?=2045 GOTO PROC3   /*TRY PROC3*/
//RETURN
//TAG PROC23
   .
   .
   .
```

Note: If a requester is trying to use an existing MRT that is not fully operational, return code 2045 is issued.

Limiting the Number of Users

Usually, the number of users of a MRT program means the number of requesters. To limit the number of requesters, you use the MRTMAX value at compile time. At run time, you can use the ATTR OCL statement to reduce the MRTMAX value.

Program users can also include acquired display stations, which can be acquired either by using statements in the high-level language program or from the job stream by using the WORKSTN OCL statement. Except in RPG, there is no way to control the number of display stations acquired at run time. In RPG, the NUM continuation-line option can be used to limit the total number of requesters plus acquired display stations.

First-Requester Considerations

The following considerations apply only to the first requester of a program:

- Many of the execution-time variables (such as the parameters on the FILE OCL statement) are specified by the first requester. You might not know if you are the first requester or a subsequent requester. If you are a subsequent requester and you specify a file name, for example, it may already be defined.

- Although a MRT is logically a separate job, it can be used as a step in any other job. Whenever possible, variables that initialize the job status come from the CNFIGSSP procedure or from initial program load. These variables include date, date format, forms number, and lines per page. Other variables must come from the first requester. These variables include priority, NEP, MRTMAX, log, external switches, current library, procedure library, local data area, and step region size.

Summary of MRT Program Considerations

Unlike a SRT program, a MRT program cannot write to a requesting display station before reading from the display station. A MRT program can be called only from a MRT procedure.

The system checks by procedure name whether the MRT is already active. Because it would take more time, the system does not check by program name or by procedure name qualified by library name. Therefore, you should have only one copy of a MRT procedure in the entire system.

When a MRT program (that is non-NEP) releases its last requester, the MRT procedure is indicated as no longer active. If another request is made for that MRT procedure and program, but the MRT program has not yet ended, there could be a delay in beginning the subsequent MRT if it requires exclusive use of some system resources.

For non-MRT programs, you can specify that the data following a procedure name is either data for the program or parameters for substitution in the OCL statements. For MRT programs, the procedure can be called only with data, not with parameters for substitution. Anything following the MRT procedure name is saved until the MRT program does its first input operation.

If a MRT program reads from a specific display station as opposed to reading from any display station that has input ready, all other users wait until the specified display station input operation is complete. For example, if an RPG program processes a NEXT operation for display station W2, no other display station input is processed until the operator at display station W2 presses the Enter key.

When an ATTR OCL statement with RELEASE-YES is specified for a MRT program, the MRT program becomes a MRT program with zero requesters, not a NRT program. This technique is advisable for MRT-NEP programs.

The EVOKE OCL statement cannot be used to initiate a MRT procedure. However, the SSP-ICF EVOKE operation code can be used for this purpose. The SSP-ICF EVOKE-EOX operation code can be used to initiate a MRT program, but all the program can do with the display station file is to determine that the requester is an SSP-ICF session.

When a SRT program runs, most system messages go to the requesting operator. For a MRT or NRT program, all system messages go the system operator.

You can use ?CD? = 3721 on a procedure control expression to test whether a MRT job step was terminated by inquiry option 2.

If a MRT program writes more than once to a display station and does not suppress input on all but the last of the write operations, the operator might suspend the program if he presses the Attn key.

Even if a MRT procedure specifies that the OCL statements should be logged to the history file, the statements are not displayed unless HISTORY LIST,ALL or HISTORY CRT,ALL is specified. Furthermore, if you have system security installed, you can specify ALL only if your security classification is operator or higher.

Chapter 17. Jobs and Job Processing

This chapter describes how the system processes jobs and how you can affect the way the system processes jobs.

Jobs and Job Steps

A **job** is a unit of work to be done by the system. Usually a job is composed of one or more programs. For example, an order entry job might run one program to process orders and then a second program to print reports about the orders.

A **job step** is a unit of work done by one program. A job that runs two programs has two job steps. A job step usually begins with a LOAD OCL statement and usually ends with a RUN OCL statement. The following procedure contains one job step because only one program is loaded and run:

```
// LOAD PROG1
// RUN
```

This next example has two job steps because two programs are loaded and run:

```
// LOAD PROG1
// RUN
// LOAD PROG2
// RUN
```

The statements in a procedure control the files, display stations, printers, and other resources used by a program. For example:

```
// LOAD PROG3
// FILE NAME-CUSTOMER
// RUN
```

These statements show the following:

LOAD The program to be run is named PROG3.

FILE A disk file named CUSTOMER is to be used by program PROG3.

RUN The program is to be run. Also indicates the end of the OCL statements for this job step.

Starting and Ending Jobs

How Jobs Are Started

Any of the following actions causes a job to start:

- Entering operation control language (OCL) at the keyboard.
- Entering procedures at the keyboard.
- Entering menu options that run procedures.
- Using the JOBQ control command or OCL statement to place a procedure on the job queue. The job queue is a list of jobs waiting to be processed by the system. Typically, batch jobs, which require no interaction with a user, are placed on the job queue.
- Using the EVOKE OCL statement to start a procedure.
- Using the Interactive Communications Feature (SSP-ICF) to have a remote program evoke a job. The *SSP-ICF Guide and Examples* has more information about SSP-ICF and how to evoke jobs from a remote program.

How the System Assigns Job Names

The SSP assigns a unique job name to each job that is submitted to the system. The job name has the following format:

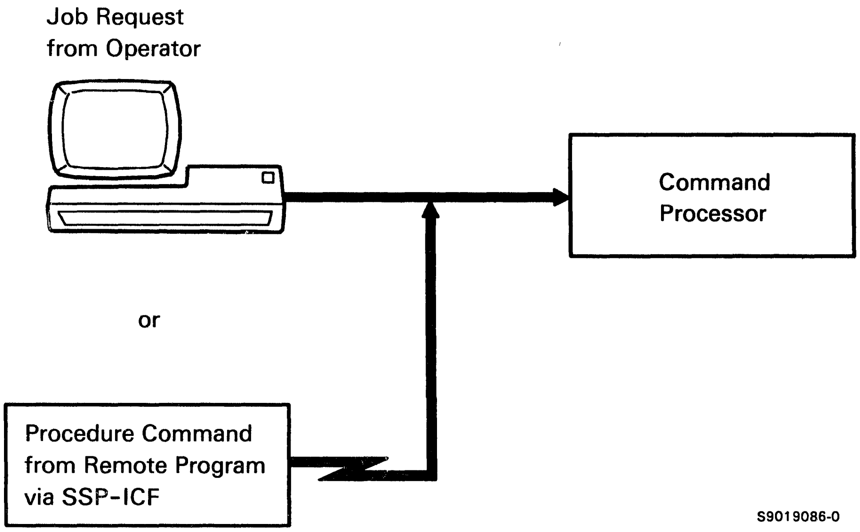
```
wwhhmmss
```

where ww is the display station ID of the requesting display station or the session ID of the associated SSP-ICF session, and hhmmss is the time the job was submitted (in hours, minutes, and seconds) based upon the 24-hour clock set by the system operator during initial program load.

Job names can be displayed by using the STATUS USERS, the STATUSF USERS or STATUS SUBSESS (for SSP-ICF remote programs) control command.

Overview of How the System Runs Jobs

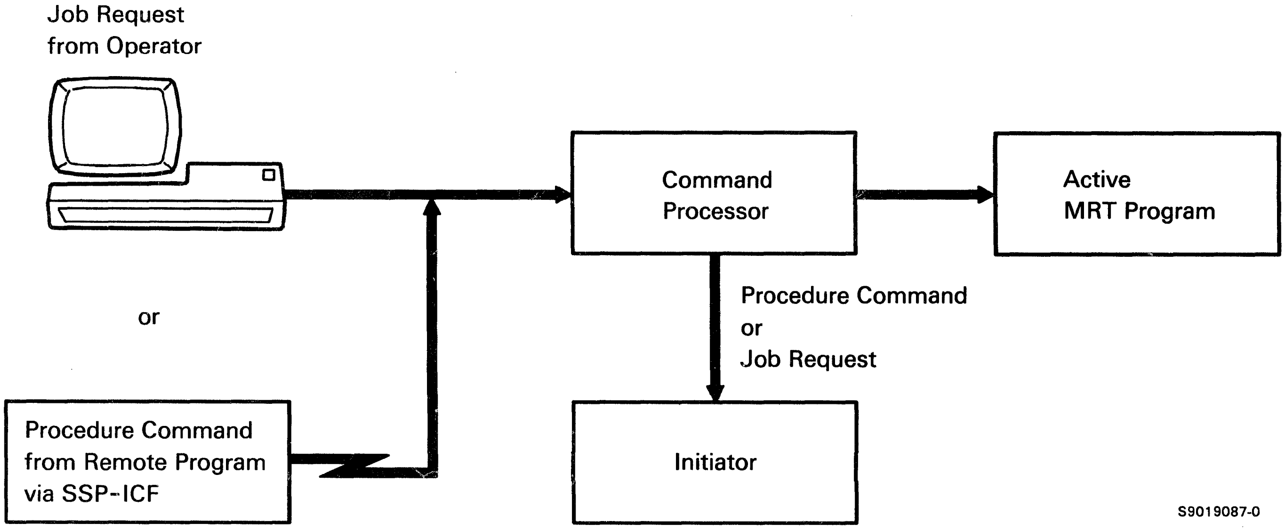
The following is an overview of how jobs are run by the system. When a user starts a job (by selecting an item from a menu, entering an OCL statement, or entering a procedure command) or a remote program requests a job to be run, an SSP function called the **command processor** processes the request.



The command processor either:

- Passes control to another function of the SSP called the **initiator**.
- Attaches the requesting display station to a MRT program, if the procedure command is for an already running MRT procedure.

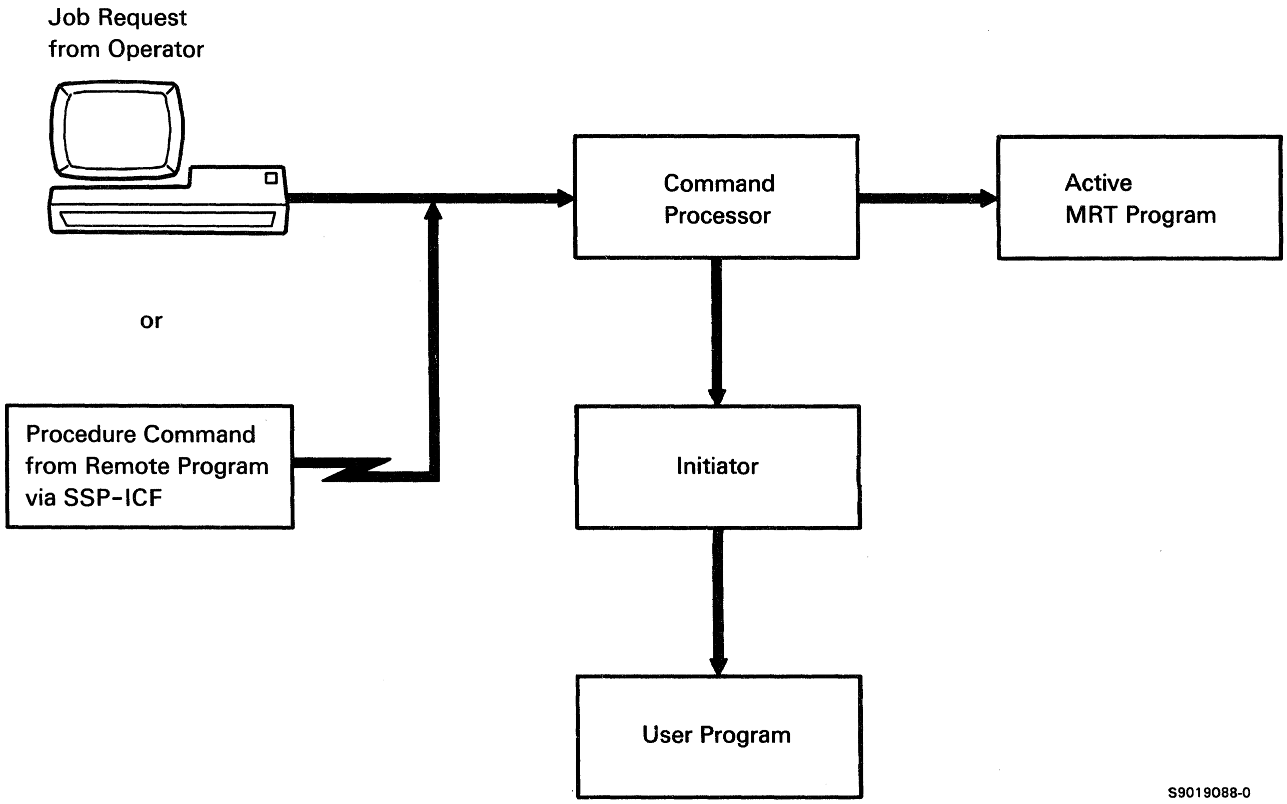
(The command processor processes control commands itself.)



The initiator reads and processes:

- Procedures. This includes MRT procedures that will start MRT programs.
- OCL statements.

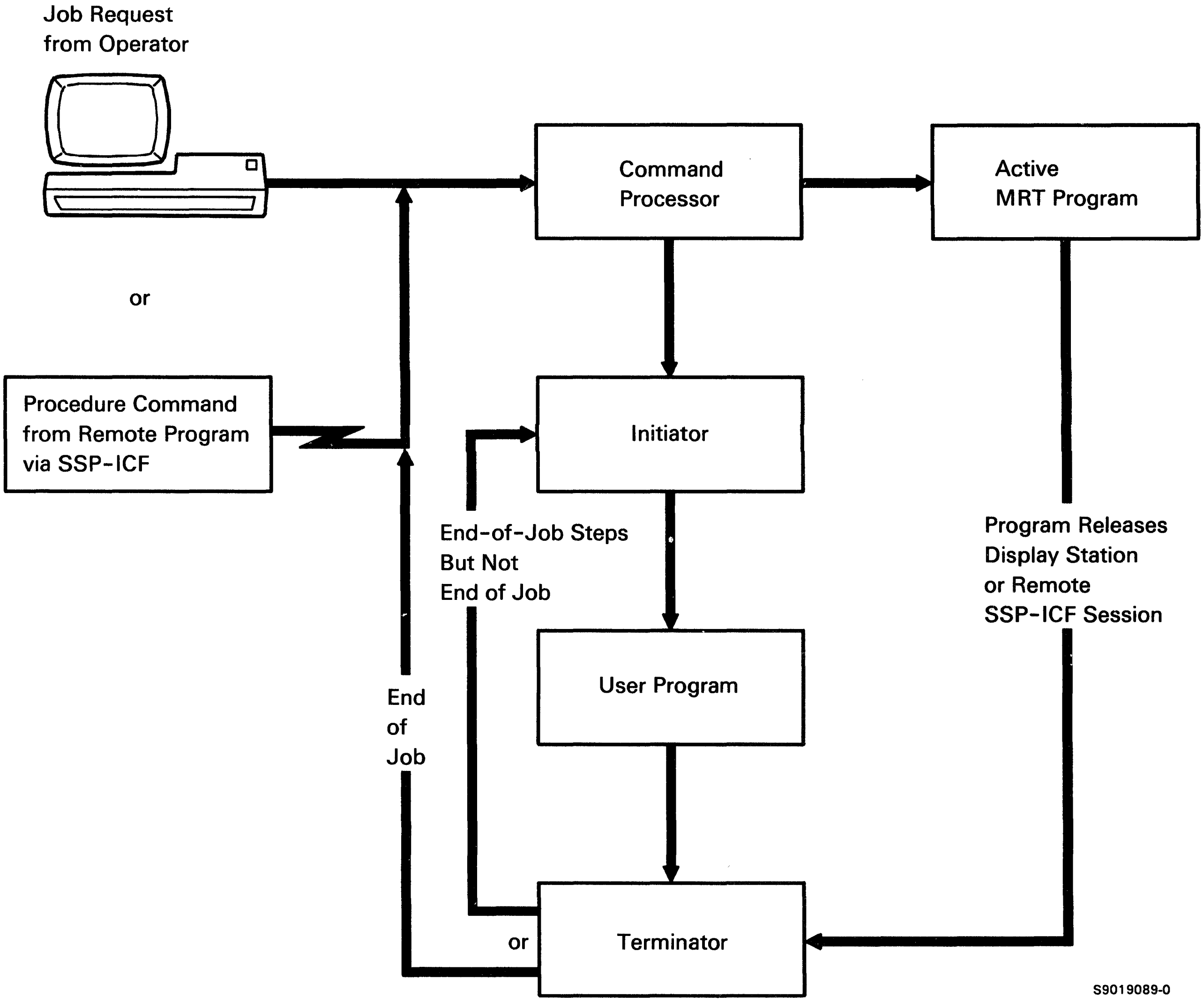
When the initiator processes a RUN OCL statement, the initiator loads and passes control to the program, which then begins running.



When the program ends, the SSP **terminator** function performs the system actions needed to end the job step. These actions include, for example, freeing system resources used by the program. If more job steps follow, the terminator returns control to the initiator.

If no other job steps follow in the job, the terminator ends the job and either:

- Returns control to the command processor for local jobs.
- Ends the SSP-ICF session for remote jobs.



The following sections describe the command processor, the initiator, and the terminator in more detail.

Command Processor

The command processor is the SSP function that first processes information that the user enters. When (1) a user enters a command or selects a menu item, or (2) a remote program sends a procedure command request using SSP-ICF, the command processor checks the command or the command associated with the menu item to determine whether a job should be started.

If the entry or menu item is a **procedure command**, the command processor next checks to see if the procedure command is a request for a currently running MRT program. If it is, the command processor attaches the display station or SSP-ICF session to the MRT program. If the procedure command is not a request for a currently running MRT program, the command processor passes the procedure command to the initiator.

If the entry or menu item is a **control command**, the command processor does not start a new job. Instead, the command processor passes control to the SSP routines that immediately process the control command.

If the entry or menu item is an **OCL statement**, the command processor passes the statement to the initiator.

Initiator

The initiator reserves system resources for the job, finds the programs, and loads and passes control to the programs in a job. In addition, the initiator:

- Ensures that enough main storage space (region size) is available for the job to run.

- Processes procedure control expressions (substitution expressions and conditional tests).

- Processes OCL statements.

- Ensures that required load members exist.

- Ensures that the files needed by the program exist at the specified share level.

- Gets buffer spaces for the job.

- Acquires display stations for which REQD-YES is specified on the WORKSTN OCL statement.

- Releases requesting display stations if RELEASE-YES is specified on the ATTR OCL statement.

How the System Processes OCL Statements and Procedure Control Expressions

A special system function called **system input** processes statements entered from a display station or from a procedure member. After reading the statement, system input performs all the substitutions and the functions specified by the statements.

The statements that control the system input processing are the procedure control expressions. The *System Reference* manual has more information about these statements.

After processing a statement, system input returns the processed statement to the calling function. During job initiation, the calling function is the initiator; therefore, all statements up to and including the RUN OCL statement are returned to the initiator. After a job has started, the statements are returned to the program that requested system input processing. A system utility program such as $COPY is an example of a program that requires system input to process utility control statements.

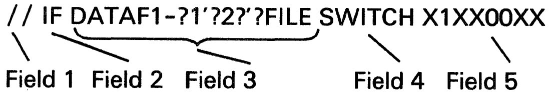
The following example shows how system input processes a typical statement. The example is intended to give a general idea of how system input works; it does not show the detailed logic of system input processing.

Before reading the example, you should be aware of the fundamental rules of system input processing:

- System input processes a statement one field at a time from left to right. Fields are delimited by blanks.
- Each time a substitution expression is evaluated, system input goes back to the beginning of the field and begins processing again. This is done to allow for nested substitution expressions.
- After all substitutions are performed, the length of the generated statement must not exceed 512 characters (including spaces). The actual length of the statement before substitution can be up to 512 characters (including spaces).

System Input Processing Example

In this example, the following statement is being processed by system input:



s9019132-0

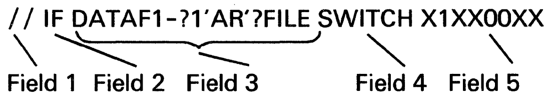
Assume that when the statement is read, parameter 1 does not have a value and parameter 2 has a value of AR. Also, assume that a file named ARFILE exists on disk.

The system input function performs the following steps:

Step 1. Identifies the first field as //.

Step 2. Identifies the second field as IF, a valid procedure control expression.

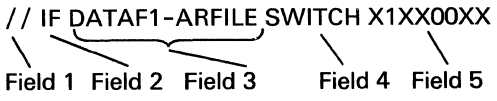
Step 3. Examines the third field and determines that the field contains a nested substitution expression. The innermost substitution is evaluated first. Therefore, system input substitutes the value of parameter 2, AR, into the field. After the substitution, the statement looks like this:



S9019133-0

Because a substitution was performed, system input goes back to the beginning of field 3 and starts processing it again.

Step 4. Examines the third field and determines that the field contains a substitution expression. System input performs the substitution. In this case, parameter 1 does not have a value and the value AR is substituted. Now the statement looks like this:

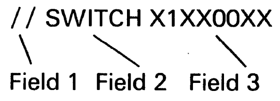


S9019134-0

Again, because another substitution was performed, system input goes back to the beginning of field 3 and starts processing it again.

Step 5. Examines the third field and determines that the field is an existence test for a file.

Step 6. Evaluates the conditional expression formed by fields 2 and 3. The file ARFILE exists on disk (it was one of our assumptions), so the test is true. Because the test is true, system input discards the IF test (fields 2 and 3). Now the statement looks like this:



```
// SWITCH X1XX00XX
```

S9019122-0

After checking each field and determining that no further substitution or system input processing of the statement is required, the statement is passed back to the caller (usually the initiator).

If the file ARFILE had not been on disk (that is, the conditional expression in the original third field had been false), system input would discard the remainder of the statement and process the next statement.

How the System Ends Jobs

A job ends when:

- The last step in a job ends.
- A MRT program releases its last requester.
- The job is canceled by:
  - A user selecting option 3 in response to an error message.
  - The system operator cancels the job using the CANCEL control command.
  - A user selects option 2 or 3 from the Inquiry display.

When a job or job step ends, the **terminator** performs system actions necessary to end the job or job step. If there are more job steps to process in the job, the terminator returns control to the initiator. If the job step is the last one in the job or if a MRT program releases its last requester, the terminator ends the job and returns control to the command processor and ends any active SSP-ICF sessions.

Normal Termination

When a job step ends or when the user selects option 2 in response to an error message, the terminator does the following functions:

- Saves newly created resident files on disk.
- Makes available work areas (such as buffers and main storage space) used by the program.
- Initializes work areas to be used by the next job step.
- Deletes any scratch files (RETAIN-S) used by the job.

If the job step is the last one of the job, the terminator does the following additional job termination functions:

- Deletes any job files (RETAIN-J) used by the job.
- Releases the requesting display station if it is still attached to the job and returns control to the command processor so the user can request another job.
- Ends the requesting SSP-ICF session if the program was requested by a remote program and if the remote session is still active.
- Frees any system resources that were used by the job.

Abnormal Termination

Abnormal termination of a program occurs when any of the following conditions happen:

- A user selects option 3 in response to a displayed error message.
- The user interrupts the program and selects option 2 or option 3 from the Inquiry display for all programs except MRT programs.

  For MRT programs, option 2 releases the display station from the MRT program and continues with the next job step; option 3 releases the display station from the MRT program and cancels the remaining job steps.
- The system detects an error condition during normal termination and cannot let the job normally terminate.
- A user enters the CANCEL command at the system console or at a system service display station.

When one of the above conditions occurs and the program is not a MRT program, the system automatically runs the terminator, and any jobs following the job step in error are not performed. If option 2 was selected from an Inquiry display, the files used by the terminated job are closed. For all other types of abnormal termination, files are not closed, and the following statements are true:

- Files contain all updates done before the abnormal termination.
- Any additions made to shared and nonshared files remain in the file.
- No new entries are added to the disk VTOC.
- If RETAIN-S was specified for a resident file in this job step, the resident file will not be deleted.

When the terminator function ends, it returns control to the command processor and ends any active SSP-ICF sessions.

Job Management and Job Scheduling

The system allows you to assume an important role in managing and scheduling your jobs. For example, you can:

- Affect how your programs use main storage by using different processing priorities.

- Affect the order in which your jobs are to be processed by the system by using different job queue priorities in the job queue.

Processing Priorities

Priority is the relative ranking of items. For example, a job with high processing priority should run faster than a job with medium or low priority.

You can specify the processing priority for a job or job step by using the ATTR OCL statement. To have a job processed at the same priority each time it was run, you would specify the ATTR OCL statement in that job's procedure. Operators can also use the PRTY command to assign processing priorities to the next job they start; system operators can use the PRTY command to change the priority of a currently running job. The processing priorities are:

- High

- Medium

- Normal

- Low

If your job does not specify a processing priority, the system assigns your job a normal priority. The *System Reference* manual describes the PRTY control command and the ATTR OCL statement. The *Using Your Display Station* and *Operating Your System* manuals shows how to use the PRTY command to specify the processing priority of a job you want to start.

The initiator takes into account the following items regarding processing priorities of jobs:

- Each job step (or program) may have its own priority, and the priority becomes effective as soon as the initiator checks the OCL statements.

- A nonrequester-terminal (NRT) program (that is, a program running without a requesting display station) has the same priority as the job that started the NRT program.

- When a job is started by using the EVOKE OCL statement, the evoked job has the same priority as the job that evoked it.

- The priority of a MRT program is not related to any current requesters of the MRT. The priority of MRT programs is determined by:

  - The priority of the MRT procedure.

  - The priority specified by the user that initially requested the MRT procedure.

  - If the MRT procedure was initially requested from a single-requester-terminal (SRT) procedure, the priority specified in that SRT procedure.

Processing Priority Considerations

In general, let the system assign processing and job queue priorities to your jobs; the normal priority can usually handle your job needs.

When assigning priority levels to jobs, your main goal is to have the system process the maximum number of jobs in the least amount of time.

You may want to use the processing and job queue priorities to establish groups of jobs with certain characteristics. For example, you may want to assign all certain jobs a job queue priority or a processing priority. You may want to run your testing jobs with one processing priority and your production programs with another processing priority.

You may want to assign processing priority to programs based upon the following criteria:

- Assign high priority to jobs requiring fast response time at a display station or quick system throughput.

- Assign low priority to jobs that do not use display stations and will run for a long time.

- Assign medium priority to interactive jobs that could be reclassified as batch jobs by the system, based upon the processing requirements of the program.

If you do assign more than one job a high priority, the overall effect might actually be to reduce the actual response time and throughput of the jobs you are trying to make run faster.

Job Queue

The job queue allows you to run 1 through 50 batch jobs at once and still continue to use your display station for other work. During the hours when your system is busiest, you can run very few jobs from the job queue. Later, during the light load hours, run many jobs from the job queue.

Job Queue Priority Levels

You can specify six different job queue priority levels, numbered 0 through 5. Job queue priority 5 is the highest level, and priority 0 is the lowest level. By specifying job queue priority levels, you can determine the order in which jobs are processed from the job queue.

Level 5 jobs are considered first, followed by level 4 jobs, then level 3 jobs, and so on. If you do not assign a priority level to jobs using the job queue, the system assigns each job a priority level of 3. An example of how to use the job queue priority levels would be to have the important batch portions of a printing application (such as the printing of payroll checks) use level 5; but have level 1 used for program compilations.

You can increase the number of jobs to run from the job queue by entering:

```
CHANGE JOBS,JOBQ,# of jobs
```

This implementation also gives you the flexibility to have multiple job queue jobs in addition to the sequential execution of jobs. To have 5 job queue jobs execute concurrently, but also have job queue priority 3 jobs execute sequentially, you would enter the following commands:

```
CHANGE JOBS,3,1
CHANGE JOBS,JOBQ,5
```

The job queue priority of a job is specified when you put the job on the job queue using the JOBQ control command or OCL statement. For example, the following JOBQ command:

```
JOBQ 4,PAYLIB,PAYROLL
```

puts the PAYROLL procedure (from the library PAYLIB) on the job queue with a job queue priority of 4. The *System Reference*, *Using Your Display Station*, and *Operating Your System* manuals describe the JOBQ control command and OCL statement.

Job queue priority is different from processing priority. Job queue priority specifies the order the jobs in the job queue are presented to the system to be run. Processing priority is the order in which jobs **already running** on the system are assigned system resources. Jobs on the job queue can be assigned a processing priority, but it only becomes meaningful when the job is taken off the job queue and starts running.

The following table shows the order in which jobs are considered by the system to be run based upon the order the jobs were placed on the job queue and the job queue priority specified.

Order Placed in Job Queue		Order Considered to be Run by the System	
Job	**Priority**	**Job**	**Priority**
Job 1	3	Job 6	5
Job 2	4	Job 2	4
Job 3	2	Job 1	3
Job 4	3	Job 4	3
Job 5	1	Job 3	2
Job 6	5	Job 5	1

You can start and stop individual jobs within each priority level. For example, if you do not want job 4 within priority level 3 to run, you would use the HOLD JOBQ control command so that jobs would be selected in the following order:

Order Placed in Job Queue		Order Considered to be Run by the System	
Job	**Priority**	**Job**	**Priority**
Job 1	3	Job 6	5
Job 2	4	Job 2	4
Job 3	2	Job 1	3
Job 4	3	Job 3	2
Job 5	1	Job 5	1
Job 6	5		

To run job 4, you would use the RELEASE JOBQ control command so that job 4 could be selected from the job queue.

A job being held is not allowed to run until it is released or explicitly started by using the START JOBQ command. Other jobs in the same priority level that are not held are allowed to run.

You can also prevent entire priority levels from being considered by the system to be run. For example, if you do not want any jobs from priority level 3 running, use the STOP JOBQ control command. Jobs are then presented to the system in the following order:

Order Placed in Job Queue		**Order Considered to be Run by the System**	
Job	**Priority**	**Job**	**Priority**
Job 6	5	Job 6	5
Job 2	4	Job 2	4
Job 1	3	Job 3	2
Job 4	3	Job 5	1
Job 3	2		
Job 5	1		

Later on, you could use the START JOBQ command to allow the priority 3 jobs to be run.

Priority Level Zero: You can assign priority level zero to jobs that might not need to be run for some time, such as, special jobs to be run later in the day or overnight. Priority level zero is the one priority level of the job queue that is initially stopped.

For example, you may submit jobs with this level during the day, and then start priority level zero at night to have the jobs run overnight.

Priority level zero combined with the system's automatic message response capability is an effective way of submitting jobs to be run overnight when the processing load of the system might not be so great and the system could run unattended. For more information about the system's automatic response capability, see Chapter 14, "Messages and Message Members."

Processing Priority of Jobs on the Job Queue

Each job on the job queue can have a different processing priority. The **job queue** priority level determines which jobs are presented to the system to be run, while the *processing* priority determines the order in which jobs are run by the system.

The processing priority of a job placed in the job queue is **normal** unless you use the PRTY command or the ATTR OCL statement to specify a different processing priority. For example, the following PRTY and JOBQ commands:

```
PRTY HIGH
JOBQ 4,PAYLIB,PAYROLL
```

put the PAYROLL procedure (from the library PAYLIB) on the job queue with a job queue priority of 4, and with a processing priority of HIGH.

If you use a procedure to place a job in the job queue, your job has the same *processing* priority as the procedure that placed it in the job queue.

Using the system console, you can enter the CHANGE control command to change the processing priority of a job that is in the job queue. The job queue priority level does not change nor does its position within the priority level change.

Print Queue Manager

The print queue manager provides an interface through which DW/36 can control the job queue. The print queue manager is a part of the SSP and resides in #LIBRARY. Using the print queue manager, some of the functions you can use to control the job queue or spool queue are:

- Search
- Hold
- Cancel
- Move
- Release

For more information about the print queue manager, see the printer control guidelines in Chapter 3, "Printed Output" on page 3-1.

Dispatching and Swapping of Programs in Main Storage

Some of the applications you run on the system are more important than others in terms of throughput or response time. This section describes how the system's dispatching and swapping functions work together in order to optimize performance for your programs. (Not all the details of how dispatching and swapping are described, only those details that are important to you as a programmer are described.) This section also describes some methods **you** can use to affect how the system dispatches or swaps programs.

The system (1) handles multiple display station operations at the same time and (2) supervises the sharing of system resources among programs.

Additionally, the system treats interactive and batch programs differently. Interactive programs tend to spend a relatively large amount of time waiting for users to enter data, thus the system easily recognizes them and optimizes their performance by using the wait times to process other programs.

Batch programs seldom do any display station input operations and sometimes use large amounts of system resources. Even though the system can easily recognize batch programs, the system can do little to optimize their performance (other than overlap processing with input/output operations). What the system *can do*, however, is minimize the effect of batch programs on the performance of interactive programs.

Dispatching is the act of assigning the main storage processor to the most eligible program on the system. The following table shows the sequence in which programs become eligible for dispatching. The sequence is based on processing priority.

Processing Priority	**Dispatching Sequence**
System	First
High	Second
Medium or Normal	Third
Low	Fourth

Only the system can use the system processing priority, for example, for programs such as the command processor. You can specify the other priorities for your applications to control sequence in which programs are assigned to the main storage processor.

The following diagram shows programs with various processing priorities and program status types: waiting and ready. Waiting means the program is waiting for some type of input/output operation; ready means the program is eligible to use the main storage processor.

Program	Processing Priority	Program Status
Program 1	High	Waiting
Program 2	High	Waiting
Program 3	Normal	Waiting
Program 4	Normal	Ready
Program 5	Normal	Waiting
Program 6	Low	Ready

Starting the search from the highest priority, program 4 would be selected because it has the highest priority of all programs ready to be run.

Swapping

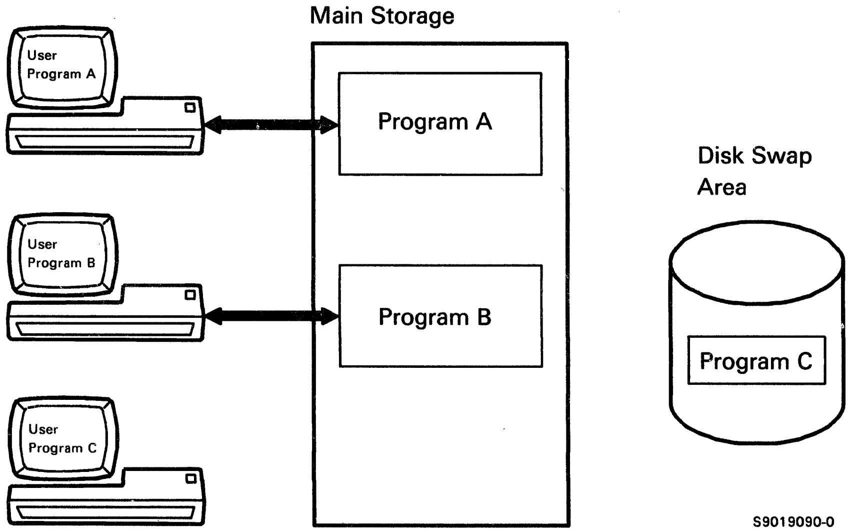
The order of dispatching programs is from high to normal to low. The normal priority also includes the medium priority. To better understand these priorities, you should understand what program swapping is and how it works.

When the system is moderately busy, it is very common that the size of main storage is not sufficient to accommodate all the programs running on the system. The system is processing more programs than can fit in main storage at the same time.

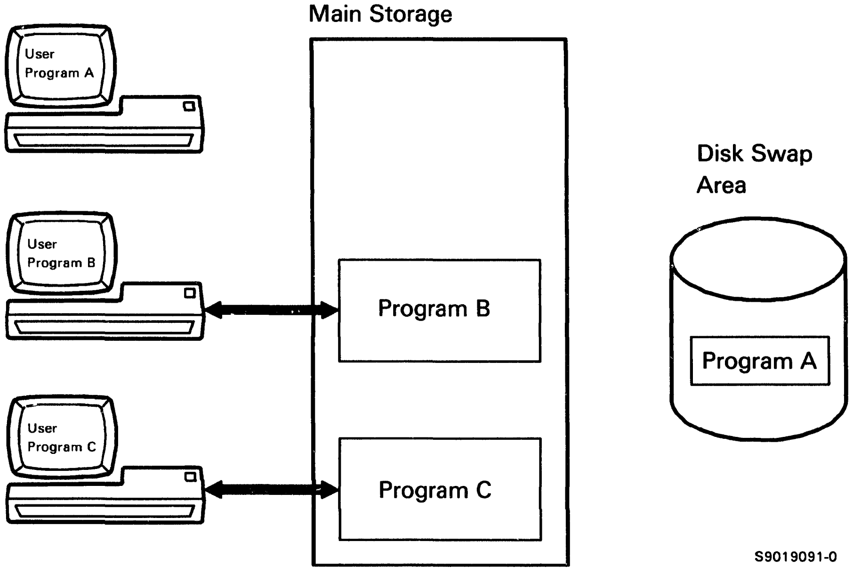
The system moves (or *swaps*) programs that are not busy or less important to disk so that their main storage can be used by other programs. Later, the programs that were swapped out to disk can be swapped back into main storage. Usually, the swapping occurs when a display station input operation has occurred.

The following example shows how swapping can increase the total amount of work that programs running at the same time can do.

Assume that programs A, B, and C are running at the same time and have the same processing priority. Programs A and B are in main storage; program C has been swapped to disk.



The system knows that program C is ready to run when program A requests input from display station W1. Program A will be inactive until the user enters data. Therefore, the system swaps program A to disk and swaps program C into main storage.



Programs B and C then share the main storage processor. After program A becomes ready, it will replace either program B when it requests input from display station W2, or program C when it requests input from display station W3.

Automatic Priority Adjustment of Programs

Although the relative priority of all programs remains the same (that is, high is before normal and normal is before low), there is some adjustment of priorities within the priority levels. For example, if four programs have a medium priority, the system might give preference to one of the four programs and allow it to have more system resources than the other three programs. Because there are usually more normal priority programs being processed by the system, this section describes how the system automatically adjusts the normal priorities.

Two dispatching status types, waiting and ready, have previously been described. One type of waiting status that is significant for swapping purposes is the **long wait**. This status type applies to certain operations that tend to last for a relatively long time. This means that the system usually has enough time to swap the program out to disk before it will again be ready. Waiting for a resource is an example of a long wait. The most common is the wait for display station input.

When a program is ready to be swapped in, pages of main storage might have to be swapped out. Programs that are in a long wait are the most eligible to be swapped out. If no programs are in a long wait, another program might be swapped out, usually a batch program. (Batch programs are recognized by the system as those programs that have exceeded a system-defined time limit without entering a long wait.) The system performs additional checks to ensure that nonproductive swapping does not occur.

When a program is swapped out, its priority is effectively increased or maintained at its current level. When a program runs for a particular length of time without entering or attempting to enter a long wait, its priority is reduced. The effect of this is that batch programs move to the bottom of the normal priority programs while interactive programs stay at the top. When interactive programs are swapped out, they have preference over batch programs to be swapped in when the user is finished typing in data.

Job Scheduling Guidelines

The following are some ideas you should consider:

- When two or more batch programs are run at the same time, they:

  - Might take longer to run than if they were run separately (one after the other).

  - Have a tendency to adversely affect response time of interactive programs.

  One way to keep batch programs from adversely affecting interactive programs is to put the batch programs on the job queue.

- Interactive programs that do a lot of calculating or do a lot of disk reading and writing might have poor or inconsistent response times. The system might treat these interactive programs the same as batch programs. Therefore, you may not want other batch programs running on the system when these types of interactive programs are running. Also, you may want to assign medium priority to these types of interactive programs.

- The specification of high and medium processing priorities might cause normal priority interactive programs to process slower.

- Use processing priorities sparingly; however, if you must use them:

  - Use low priority for batch programs (these programs might be swapped more often).

  - Use medium priority for your important interactive programs that perform lots of calculations or disk read and write operations; for example, order entry applications.

  - Use high priority for extremely important programs or for efficiently coded interactive programs. You should not assign high priority to more than one batch job; if you do, your interactive jobs may slow down considerably.

- Consider the control storage requirements for the jobs you are running. See "Control Storage Considerations" later in this section.

Control Storage Considerations

Depending on the size of your control storage relocatable area and the functions you run there, all functions may not be able to run concurrently. For example, if you have a 5360 System Unit stage 1 (see Figure 17-2), and if you run BASIC and FORTRAN programs at the same time, you cannot also run the data compression, extended trace, SMF communications, or 1255 MCR functions.

The following is a list of functions that run in the relocatable control storage area and the size of the relocatable storage each function requires:

Function or Programs	**5360 System Unit**	**5362 System Unit**	**5364 System Unit**
BASIC	3.75K	3.75K	3.75K
Data Communications	1.75K	1.75K	1.75K
Data Compression	1.75K	1.75K	1.75K
Disk Cache	1.25K	1.25K	1.25K
Diskette Functions	2.0K	2.5K	2.5K
Extended Trace	0.75K	0.75K	0.75K
FORTRAN	3.75K	3.75K	3.75K
Local Area Network	2.0K	2.0K	2.0K
SMF Communications	1.25K	1.25K	1.25K
1255 MICR	1.25K	See Note 1	See Note 1
6157 1/4-inch Tape Drive	0 See Note 2	0 See Note 2	0 See Note 2

Notes:

1. *1255 MICR is not allowed on the 5362 and 5364 System Units.*
2. *6157 1/4-inch tape drive can run without using any control store relocatable space on the 5360 and 5362 System Units.*

Figure 17-1. Relocatable Area Sizes and Rules

Data communications and diskette functions are always guaranteed space to run. If enough space is not available for the other functions, an error message is issued. To recover from the error, you must either wait for some programs to finish or cancel some programs. You can either cancel the program requesting the storage (option 3), or cancel some other programs that are using one of the other functions. This will free up the space for the requested program. The following relocatable area size and rules will help you understand and schedule which functions can run concurrently on your system.

System Unit	Control Storage Relocatable Size
5360 stage 1	3 3.75K segments
5360 stage 2 or stage 2.1	5 3.75K segments
5360 stage 3	5 3.75K segments
5362 with work station controller	5 3.75K segments
5362 without work station controller	3 3.75K segments
5364	2 3.75K segments and 1 1.5K segments

Figure 17-2. Control Store Relocatable Sizes

Note: *To determine the stage of your system, look on the label on the inside of the control panel cover. In the top center of the label, immediately to the right of the words* ***Processor Check****, will be the stage of your system: stage 2, stage 2.1, or stage 3. If there is no indication and the system unit is not a 5360 Model 1D, you have a stage 1 system.*

Control Store Relocatable Area Usage Rules

Following are the rules used by the system to use the control store relocatable area:

1. Data communications and diskette function code are always guaranteed space to run.

2. Code in the relocatable area cannot cross over the segments.

3. Space to accommodate the entire size of the function (see Figure 17-1) must be available for a function to run.

		Data	Extended	SMF	1255	Disk
BASIC	FORTRAN	Compression	Trace	Communications	MICR	Cache
X		X	X			X
	X	X	X			X
X				X	X	X
	X			X	X	X
X			X	X		X
	X		X	X		X
X			X		X	X
	X		X		X	X
		X	X	X	X	X
X	X					
X			X	X	X	
	X		X	X	X	

Figure 17-3. Partial List of Programs That Can Run Concurrently on a 5360 System Unit Stage 1 Control Storage (See Note)

All the programs listed above can run concurrently on the 5360 System Unit with a stage 2 or 2.1 control storage processor.

Note: To determine the stage of your system, look on the label on the inside of the control panel cover. In the top center of the label, immediately to the right of the words ***Processor Check****, will be the stage of your system: stage 2, stage 2.1, or stage 3. If there is no indication and the system unit is not a 5360 Model 1D, you have a stage 1 system.*

BASIC	FORTRAN	Data Compression	Extended Trace	SMF Communications	Disk Cache
X			X	X	X
	X		X	X	X
	X	X			X
		X	X		
		X	X	X	
X		X	X		
	X	X	X		

Figure 17-4. Partial List of Programs That Can Run Concurrently on a 5362 System Unit without a Work Station Expansion Feature

Note that BASIC and FORTRAN programs are mutually exclusive.

All the programs listed above can run concurrently on a 5362 System Unit with a work station expansion feature.

BASIC	FORTRAN	Data Compression	Extended Trace	SMF Communications	Disk Cache
X					X
	X				X
			X	X	X
		X	X		X
X			X		
	X		X		
X				X	
	X			X	
		X	X	X	

Figure 17-5. Programs That Can Run Concurrently on a 5364 System Unit

Note that data compression, BASIC, and FORTRAN are mutually exclusive.

Job-Related Information Contained in the History File

The system records information about each job run by the system in the history file. The information is logged when the job ends, and includes the following:

- Starting time
- Ending time
- Amount of time used to run the job
- Date the job was run
- User ID of the operator running the job
- Display station ID of where the job was run
- Name of procedure that started the job
- Whether the job was:
  - A single-requester-terminal program
  - A multiple-requester-terminal program
  - Run from the job queue
  - A nonrequester-terminal program

The HISTORY procedure in the *System Reference* manual has more information about the information logged to the history file.

Evoking Other Jobs

You can use the EVOKE OCL statement to start a new job from within a job that is running. For example, you have a procedure with one or more job steps similar to the job shown below:

```
*  Start order entry program
// LOAD ORDENT
// RUN
*  Start order summary listing program
// LOAD PRTLIST
// RUN
```

The first program, ORDENT, uses the display station to enter new orders into the system. The orders are stored in a disk file. The second program, PRTLIST, reads the disk file and prints the information on an order form. Because the PRTLIST program does not require a display station, it could be used to do other work while the PRTLIST program is running. However, the display station will remain attached to the procedure while the PRTLIST job step is running, thus preventing the display station from doing other work.

If you create two procedures and use the EVOKE OCL statement to start the second procedure, the user's display station is free to do additional work **while** the PRTLIST program is running. For example:

```
*  Start order entry program
// LOAD ORDENT
// RUN
*  Start order summary listing procedure
// EVOKE PRTLIST
```

Procedure PRTLIST

```
*  Start order summary listing program
// LOAD PRTLIST
// RUN
```

The *System Reference* manual describes the EVOKE OCL statement.

Submitting Jobs to be Run Later

You can have jobs start at a later time by using either:

- The job queue
- The WAIT OCL statement

Job Queue

The job queue is described earlier in this chapter. You can use the JOBQ control command or the JOBQ OCL statement to place a job on the job queue. The *System Reference*, *Using Your Display Station*, and *Operating Your System* manuals describe these.

The JOBQ OCL statement allows you to place a job on the job queue from within a job that is already running.

The HOLD and RELEASE commands allow you to hold and release jobs on the job queue.

The START and STOP commands allow you to start and stop the processing of jobs from the job queue, or to start and stop the processing of job queue priority levels. The START command also allows you to immediately start a job on the job queue.

WAIT OCL Statement

The WAIT OCL statement causes a job to wait until a certain time of day, or until a certain period of time has passed. Once a WAIT OCL statement is processed, the job does not resume processing until the specified condition is met. The *System Reference* manual describes the WAIT OCL statement.

When the WAIT OCL statement is processed, any system resources allocated to a waiting job are treated as if they are owned by a never-ending program. See Chapter 16, "Programs" for more information about never-ending programs.

If you use the WAIT OCL statement, you should consider the following items:

- The ?TIME? substitution expression, along with the WAIT OCL statement, can be used to check to see if the job has waited until a specific time. Then processing can begin based upon how long the job has waited.

- The STATUS USERS control command can be used to see if the job is waiting.

- A job can be canceled even if it is waiting.

- If you place a job containing a WAIT OCL statement on the job queue, that job may prevent other jobs on the queue from processing. This might delay processing considerably, depending upon the length of time specified by the job using the WAIT OCL statement.

This example shows how to make a procedure wait until 4 p.m. (a time of 160000). If the time is already greater than 4 p.m., the WAIT statement is not processed. For example, if the procedure were run at 5 p.m., the procedure PROC1 would be run immediately. If the ?TIME? test were not performed, procedure PROC1 would not be run until 4 p.m. of the **next day**.

```
// IFF ?TIME?>160000  WAIT TIME-160000
// EVOKE PROC1
```

Changing the Position of a Job in the Job Queue

Once a job has been placed in the job queue, you can place a job in a higher or lower priority level by using the CHANGE JOBQ command. The processing priority of the job is not changed. The *System Reference*, *Using Your Display Station*, and *Operating Your System* manuals have more information about the CHANGE control command.

Submitting Jobs by Security Classification

You can specify that certain security jobs be run by a user who has a particular security classification. The following is a list of the various security levels, listed from the highest to the lowest level of security, that can be assigned to users:

Master security officer (M)
Security officer (S)
System operator (O)
Subconsole operator (C)
Display station operator (D)

When you submit a job, you can use the SECURITY conditional expression to determine whether an operator has the required security level.

The following example uses the SECURITY conditional expression to determine whether the user who submitted the job has a security classification of system operator or higher. The SECURITY-O expression tests for the system operator security classification. If the user does not have system operator classification or higher, the IFF statement cancels the job.

```
// IFF SECURITY-O CANCEL
// LOAD PROGRAM
// RUN
```

You can also use the SECURITY conditional expression to determine whether password security is active.

The *System Reference* manual has more information about the SECURITY conditional expression,

Preventing Jobs from Being Canceled or Interrupted

Preventing Canceled Jobs

If you specify CANCEL-NO on the ATTR OCL statement in a job, any subsequent use of the Attn key at the display station causes options 2 and 3 (cancel job) of the Inquiry display not to be shown. This prevents users from canceling jobs from the display stations. A job may be canceled at any time by a user using the CANCEL control command at the system console or system service display station.

Preventing Interrupted Jobs

If you specify INQUIRY-NO on the ATTR OCL statement in a job, any subsequent use of the Attn key at the display station causes option 1 (request command display) of the Inquiry display not to be shown. This prevents another job from being started at the display station.

The *System Reference* manual has more information about the ATTR OCL statement.

If you are running MRT programs, the ATTR OCL statement should be placed in the MRT procedure.

Preventing Informational Messages from Displaying

Most IBM-supplied procedures display informational messages at the display station from which they are run. When you have a job composed of programs and IBM-supplied procedures (such as DELETE) that display informational messages, the messages can be very confusing to a user who has no idea of what processing steps are being performed by the system.

Also, messages sent to remote display stations cause the system to store the display on disk, show the informational message, and then show the previous display again. This can have an adverse effect on performance for remote display stations.

The INFOMSG control command or OCL statement allows you to select whether informational messages from procedures should be displayed. The *System Reference* manual has more information.

Running Jobs During Initial Program Load (IPL)

The system allows you to have jobs run as you are starting up the system by using the #STRTUP1 and #STRTUP2 procedures. The *System Reference* manual has more information about the #STRTUP1 and #STRTUP2 procedures.

#STRTUP1 Procedure

The #STRTUP1 procedure is called by the IPL system function and runs **during** IPL after the system operator has signed on but before other users are allowed to sign on. The #STRTUP1 procedure can be used to perform job processing functions you want done when no other jobs are being run and when these functions require a dedicated system. For example, you may want to condense certain libraries or reorganize disk space before running any jobs.

You create the #STRTUP1 procedure member and place procedures, such as CONDENSE or COMPRESS, within the member. You can store the #STRTUP1 procedure either in the system library or the system operator's sign-on library.

You cannot start any other jobs until the #STRTUP1 procedure ends nor can any parameters be passed to the #STRTUP1 procedure. Because the #STRTUP1 procedure runs before IPL is complete, certain functions such as print spooling and job queue initialization have not yet been done; therefore, do not place a never-ending program within the #STRTUP1 procedure member or a program that uses the job queue or print spooling.

#STRTUP2 Procedure

The #STRTUP2 procedure is similar to the #STRTUP1 procedure except:

- #STRTUP2 begins running only when IPL is completed.

- Jobs can be submitted to the system to be run while #STRTUP2 is running. For example, jobs can be submitted to the job queue while #STRTUP2 is running.

You can store the #STRTUP2 procedure either in the system library or the system operator's sign-on library.

Running Jobs Without Operators

You can have jobs running on the system while no operators are present, such as overnight. When you schedule jobs to run without operator supervision, you should consider the following:

- Using programs that have been tested and are working correctly. The programs should not require display stations.

- Using the automatic response facility to respond to any error messages that occur. Automatic response is discussed in Chapter 14, "Messages and Message Members."

- Using the HOLD-YES parameter of the PRINTER OCL statement if you are doing any printing. The HOLD-YES parameter causes one copy of the output to be held on the spool file. This allows you to reprint that spool file entry without having to rerun the job that created the printed output.

- Stopping the spool file writers. This prevents any output from printing; however, your programs can run, and the output will be stored in the spool file.

- If you do print the output from programs, be sure your printers have enough forms or paper to complete the jobs. Also, ensure the forms are aligned properly.

End-of-Day Processing

You can use the WAIT and POWER OFF OCL statements together to automatically start end-of-day processing tasks and then power off the system. For example, you may want to do a nightly save of your master files:

```
* Nightly Save Procedure
*
* Wait until 6 p.m. before beginning save
* and power-off sequence.
// WAIT TIME-180000
*
* Allocate the diskette drive
// ALLOCATE UNIT-I1,CONTINUE-YES
*
* Save the master files on diskette
SAVE CUSTMAST,,,VOL001,M1
SAVE ITEMMAST,,,VOL001,M1
SAVE ACCTMAST,,,VOL001,M1
SAVE SHIPMAST,,,VOL001,M1
*
* Start power-off sequence
// TAG PWRLOOP
// POWER OFF
*
* If power-off is not successful,
* wait 1 minute and try it again.
// WAIT INTERVAL-000100
// GOTO PWRLOOP
```

Because the system ignores the POWER statement when it cannot be safely processed (for example, when another job is running), a GOTO and TAG loop is needed in case the POWER statement was not successful. The WAIT OCL statement is included to prevent the system from constantly trying the POWER statement, which would needlessly increase the amount of work the system is doing.

Note that this procedure would require a diskette magazine in slot M1. The magazine would have to be in place before 6 p.m.

Chapter 18. Procedures

This chapter describes what procedures are and how you can create your own procedures.

Procedure Concepts

A **procedure** is a collection of statements that causes one or more programs to be run. Procedures are used to start jobs running on the system.

Many procedures are supplied as part of the SSP, the Utilities Program Product, and the programming languages. For example, the SSP procedures allow you to create data files, create libraries, and copy data files. The Utilities Program Product procedures allow you to create and change library members. The procedures supplied with the languages (such as RPGC and COBOLC) allow you to compile and run the programs you code. These procedures are described in the *System Reference* manual, the utilities manuals, and the programming language manuals.

By using procedures, you can avoid the repetition of entering several statements each time a job must be run. For example, operation control language (OCL) statements can be included in a procedure. The collection of statements is stored in a library member called a **procedure member**.

What a Procedure Can Contain

Procedures can contain the following types of statements. These are described in the *System Reference* manual.

- **OCL statements**, which are used to load and run programs. OCL statements also indicate how the SSP is to run the program and how the SSP is to use the input and output devices that the program may require. Examples of OCL statements are LOAD, FILE, and RUN.

- **Procedure control expressions**, which control how the procedure is processed based on certain conditions.

- **Procedure commands**, which cause other procedures to be run. Examples of these procedure commands are COPYDATA and SAVE. You can also use procedure commands to run your own procedures.

- **Utility control statements** for SSP utility programs, which pass information to SSP utility programs.

Procedures cannot contain any control commands.

Advantages of Using Procedures

Using procedures offers several advantages in running your jobs:

- You can run several job steps by entering one procedure command. This saves the repeated entering of OCL statements each time a job needs to be run.

- If you code the procedure command in a menu, you can start your jobs from the menu by entering an option number.

- You can prompt for and pass variables to your jobs. This information is in the form of **parameters**.

- If you code a procedure so that it uses parameters, you can check that the proper values were entered for the parameters or make decisions based on the values entered.

- You can also code a procedure so that it passes data to any one program called by the procedure. You can also change the user-programmable status indicator switches and the local data area.

Procedure Parameters

You can define parameters in your procedures. Parameters allow information and variables to be passed to the procedure. A procedure can have a maximum of 64 parameters, and each parameter can have up to 128 characters.

Parameters passed to procedures are called **positional parameters**. Parameters are separated by commas (,). Whenever a parameter appears in a procedure command, it must appear in the same position in relation to other parameters in the procedure command. That is, each parameter is assigned a place, such as the first parameter or the second parameter. If a parameter is omitted, a comma must still be used to indicate the *position* of the omitted parameter. In the following example, the second parameter is omitted:

```
PROCA PARM1,,PARM3
```

The first and third parameters are separated by two commas, which indicate that the second parameter is omitted.

To define parameters in your own procedures, you use substitution expressions. The *System Reference* manual has more information about using parameters with your procedures.

The following example shows how two parameters are entered and substituted in a procedure. Procedure PROCA contains the following statements:

```
// LOAD PROGRAM1
// FILE NAME-INPUT,LABEL-?1?
// FILE NAME-OUTPUT,LABEL-?2?
// RUN
```

When you enter the following procedure command to start PROCA:

```
PROCA FILE1,FILE2
```

the first parameter (FILE1) and the second parameter (FILE2), are substituted for the expressions ?1? and ?2?, respectively:

```
// LOAD PROGRAM1
// FILE NAME-INPUT,LABEL-FILE1
// FILE NAME-OUTPUT,LABEL-FILE2
// RUN
```

The substitution is performed when the procedure is processed by the initiator. The initiator is described in Chapter 17, "Jobs and Job Processing."

Using Procedures with Menus

Your application users generally don't need to know the procedure's name or parameters because you will provide menus for them. The user selects an option from a menu, and the system runs the appropriate procedure (the procedure you told the system to run when you created the menu). The procedure then loads and runs the program to do the task that the application user specified.

Calling a Procedure from Another Procedure

One procedure can call another procedure. A procedure called by another procedure is a **nested procedure**. Nesting is generally helpful when the same procedure is called several times in a job. The procedure can be entered and stored only once and then called as often as necessary.

A good example of using a procedure to call another procedure is deleting a file your program creates. Your procedure could simply include the DELETE procedure command. For example:

```
* Delete work file FILE1, if it exists
// IF DATAF1-FILE1  DELETE FILE1,F1
* Run program
// LOAD PROGRAM1
// FILE NAME-FILE1,RECORDS-250
// RUN
```

Considerations for Multiple-Requester-Terminal Procedures

To understand multiple-requester-terminal (MRT) procedures, you should first be familiar with MRT programs. A MRT program allows several requesting display stations to be attached to one copy of a program at a time. For more information about MRT programs, see Chapter 16, "Programs."

A MRT program can be started only by a MRT procedure. You specify a MRT procedure by selecting an option at the end of the source entry utility (SEU) that will identify the MRT procedure, or by specifying the MRT parameter of the $MAINT utility.

When you start a MRT procedure, the system first checks whether the MRT procedure is already running. If the procedure **is not running**, the system loads and starts the MRT program.

If the procedure **is running** and the number of display stations using the program is less than the maximum, the requesting display station is attached to the program. If the procedure is running and the maximum number of display stations is using the MRT program, the requesting display station waits for the MRT to release one of the other display stations.

The system processes the statements in the MRT procedure from only the first requesting display station that runs the MRT procedure. MRT procedure statement processing is done only one time for a MRT program. After the first requesting display station starts the MRT program running, all other requesting display stations are attached directly to the MRT program until the MRT program ends.

If other display stations are waiting to use the MRT program when it releases one of the requesting display stations, one of the waiting display station is attached to the program.

The following are additional facts about MRT procedures:

- Only one LOAD and RUN OCL statement pair is allowed in a MRT procedure. Also, any statements that follow the RUN OCL statement are ignored. Therefore, you may need to call the MRT procedure from another procedure if the MRT procedure is a job step within a job.

- A MRT procedure can be **called by** another procedure, but a MRT procedure cannot call another procedure.

- When a MRT procedure is started by another procedure, a new job (in effect) is started on the system. Therefore, OCL statements, such as REGION and ATTR, within the MRT procedure are processed as if they were at the beginning of a job.

- Once the MRT program is running, other display stations that request to use the MRT program (by entering the name of the MRT procedure) are attached directly to the MRT program. Therefore, the OCL statements in a MRT procedure are not processed for other requesting display stations.

- The INCLUDE OCL statement or the procedure command that starts a MRT procedure can pass data to the MRT **program**. However, no parameters can be passed to the MRT **procedure**. The data to be passed to the program starts with the first nonblank character following the procedure name and ends with the last nonblank character in the statement. The system passes the data to the program on the first input operation for the first requesting display station.

- Any DATE, FORMS, or MEMBER OCL statement that has been used in a previous job step has no effect on a job step that runs a MRT program. Instead, the MRT program uses values specified during system configuration or IPL.

- Any PRINTER or SYSLIST OCL statement that has been used in a previous job step has no effect on a job step that runs a MRT program. Instead, the MRT program uses the configured system printer for both program and system list output.

Designing Procedures

This section introduces several topics you should consider when you are creating procedures.

Naming a Procedure

You should assign meaningful names to your procedures. This makes them easier to remember and, therefore, easier to use. For example, all your procedures that have to do with accounting could begin with ACC or ACCT. You could then have ACCTPAY for your accounts payable application and ACCTREC for your accounts receivable application.

Procedure Performance Tips and Coding Techniques

This section describes some of the techniques you can use to help improve the performance of your procedures. The *System Reference* manual has more information.

- Use GOTO and TAG statements rather than several redundant IF expressions. Use one IF expression and a GOTO expression to reduce the time needed to evaluate several IF expressions. The statements skipped by the GOTO and TAG expressions are not processed.

- Use ELSE statements if you have more than one IF expression and only one of the expressions can be true. All ELSE statements are skipped after a true IF or a false IFF expression.

- Combine IF expressions when possible. The remainder of a statement is not processed after a false condition.

- Avoid using the informational message (// *) statement to display prompting messages (such as: ENTER MEMBER NAME or ENTER LIBRARY NAME). Use the PROMPT OCL statement and a display format instead. The advantages are:

  - More information can be displayed.

  - The prompt display can have help text.

  - Fewer disk operations are required.

  - For remote display stations, fewer data transmissions are made. The // * statement must save the current display contents, show the message, and show the display again after the procedure ends. The PROMPT statement shows the display format without having to save the current display contents.

  - For MRT procedures, informational messages are displayed at the system console.

- After you have tested your procedures, stop the logging of the OCL statements to the history file. You may only need to have the OCL statements logged when you are creating and testing your procedure.

- If you have many comments in your procedure, you should put a RETURN statement at the end of the procedure and put your comments after the RETURN. This way the system processes the RETURN statement and your comments are not processed (thus, saving the amount of time the system would otherwise use to read the comments).

- Use your own libraries for your applications; that is, run procedures and programs from a library other than the system library (#LIBRARY). The system library has a very large directory and, therefore, more time is needed to search for a library member in the system library than for the same member in one of your libraries.

  Also, the SSP always searches the current library first, and if the member is not found, then it searches the system library.

- Use IF conditional expressions to avoid having the system operator respond to an informational message when a procedure is sent to the job queue or when the procedure is started by the EVOKE OCL statement or an SSP-ICF evoke operation.

Programming Guidelines for Procedures

This section describes the procedures to use to create, change, and list procedures. It also shows some techniques you can use in your procedures.

Creating or Changing Procedures

You enter procedures into a library using the development support utiltiy (DSU) or the source entry utility (SEU). SEU is described in detail in the *SEU Guide*. DSU is described in detail in the *DSU Guide*.

You can also use the $MAINT utility to create and copy procedures into a library. The $MAINT utility is described in the *System Reference* manual.

Note that DSU and SEU allows you to change lines in a procedure and then store those changes. The $MAINT utility allows you to only **create** procedure members; if you want to **change** a line in the procedure, you must reenter the entire procedure.

SEU, DSU, and the $MAINT utility allow you to specify the following about procedures:

- It is a MRT procedure (the default is a non-MRT procedure).
- The procedure is to have parameters (this is the default for non-MRT procedures).
- The procedure is to pass data to a program (MRT procedures always have this characteristic).

Listing Procedures

You list a procedure by using the LISTLIBR procedure. The LISTLIBR procedure is described in the *System Reference* manual. DSU can also list a procedure.

Controlling How a Procedure Runs

You control how a procedure runs by using **procedure control expressions**. These expressions allow you to do the following and make decisions based on the results:

- Create variables in procedures by using substitution expressions, including:
  - Value entered for a parameter
  - A return code set by system programs called by your procedure
  - Current or session library
  - Date and time
  - Information in the local data area
  - Session printer
  - System list device
  - Requesting display station's ID
- Test a value and process a statement using the IF statement. For example, you can test:
  - Value of a parameter for equal to or greater than
  - Whether a procedure is currently running
  - Whether a specified amount of disk space is available
  - Whether a file or library is on disk
  - Whether a partial file or library is on disk
  - Whether a file is on diskette
  - Whether a job is being run from the job queue or is evoked
  - Whether the maximum number of display stations is using a MRT program
  - Whether a library member exists in a library
  - Security classification of an operator
  - Volume ID of a diskette

- Display messages to the application users using the // * (informational message) statement or to the system operator using the // ** (system console message) statement.

- End an entire procedure using the CANCEL statement or end a procedure level using the RETURN statement.

- Use the EVALUATE statement to:

  - Assign values to parameters

  - Add, subtract, multiply, and divide numbers

  - Determine the value of substitution expressions

  - Set the job return code

- Branch to statements in a procedure using the GOTO and TAG statements.

- Temporarily stop the running of a procedure and display a message using the PAUSE statement.

- Start another procedure or restart the same procedure using the RESET statement.

The procedure control expressions are described in the *System Reference* manual.

Debugging Procedures

Several statements can help you to debug your procedures.

DEBUG OCL Statement: This statement allows you to trace the logic flow of your procedures. It shows each level of substitution expression evaluation. The output is listed on the system list device. It also allows you to temporarily stop the running of a procedure between job steps.

LOG OCL Statement: This statement allows you to have the statements that are processed in your procedures logged to the history file. This allows you to display or print the history file to determine the statements run.

HISTORY Procedure: This procedure lists, displays, or copies the contents of the history file. It allows you to select information to be displayed, listed, or copied for the statements that have been logged to the history file. You can select:

- All entries
- Name of the procedure
- Time the procedure ran
- Display station ID
- Operator's user ID

More information about these statements is in the *System Reference* manual.

History File and Procedure Processing

The system automatically logs, to the history file, each statement processed in a procedure unless:

- At the end of the source entry utility (SEU) on the end-of-job display, you specified that the statements were not to be logged.
- You use the LOG OCL statement to turn off statement logging. The LOG OCL statement controls whether the statements in a procedure are logged to the history file. The LOG OCL statement is used to override the logging indicator you specified for the procedure member when you created it using SEU.

  The LOG statement affects only the logging of OCL statements to the history file. Other items such as messages, halts, and job information are not affected by using the LOG OCL statement.
- The procedure is an IBM-supplied procedure.

Note: All IBM procedures have a default of NOLOG. You can log IBM supplied procedures to the history file the same way normal user procedures are logged.

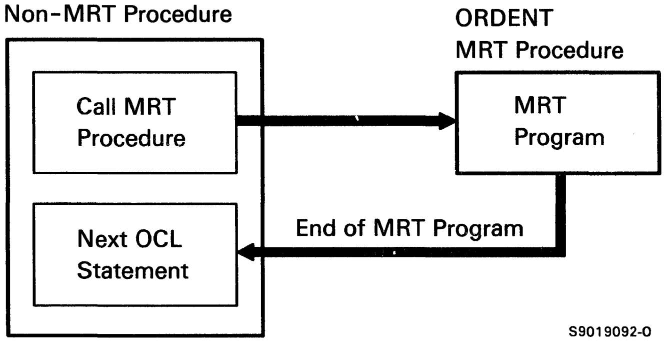
Logging requires that every OCL statement processed by the system be written to the history file. This can increase the number of disk write operations and affect performance. You should only log OCL to history file when you first create your procedure members and are testing them to see if they are working correctly.

Calling Multiple-Requester-Terminal Procedures

By having a non-MRT procedure call a MRT procedure, you can have the non-MRT procedure check whether the maximum number of requesters is already attached to a MRT program. For example, the following non-MRT procedure uses the IF procedure control expression and the MRTMAX conditional expression:

```
* Test for the maximum number
// IF MRTMAX-ORDENT GOTO TOOMANY
*
* Number of requesters is less than the maximum,
* call ORDENT (which is a MRT procedure)
ORDENT
// RETURN          (End this procedure)
*
// TAG TOOMANY     (Maximum requesters already using ORDENT)
// * 'Too many people are running order entry.'
// * 'Canceling the procedure, try again later.'
// PAUSE
// RETURN          (End this procedure)
```

This diagram shows how the procedure ORDENT would be called:



Chapter 19. Error Prevention, Detection, and Recovery

This chapter will aid you in preventing, detecting, and recovering from system, programming, and user errors and failures. As you read this chapter, keep two things in mind: anticipate the unexpected, and take precautions to prevent the unexpected.

Types of Errors and Failures

The following is a description of some of the most common system errors or failures. Although in most cases the probability of one of these occurring is relatively low, you should be aware that at least one of them might happen, and you should be prepared to correct and recover from the error or failure.

System Failure

A system failure might result in a damaged file or library. To recover from such a failure, you must restore a copy of the file or library and reapply all changes to the file or library since the last recorded save operation. The probability of a system failure is relatively low; if you keep up-to-date copies of your files and libraries, recovery should be relatively easy.

Power Failures

Power failures are unpredictable and are usually not preventable. The key to recovering from a power failure is to find out which programs and files were in use when the power failure occurred. You then have to find the last successful checkpoint, the last point at which data in the files correctly and accurately reflected the corresponding data in other files. (Checkpoint code keeps track of the last record in the file that was in some way modified or added.) All changes made to the files after the last successful checkpoint should be eliminated, and all changes from that checkpoint reapplied.

Equipment Failures

An equipment failure could be a modem failure, a problem with a communications line, a problem with external disks, or a display station problem. Equipment failures are unpredictable and are usually not preventable. Often an error message is displayed when the system detects an equipment failure; you can either cancel the job or continue the job. If you continue the job, a code may be returned to your program indicating the error type. Your program can then check the status of the data you were processing or perform the necessary corrections.

Programming Errors

There are two types of programming errors. The first is one you will find within hours of it happening, such as inaccurate information found in a printout, or a program that unsuccessfully attempts to use a display station. This type of error can be treated in much the same way as a power failure (the program producing the error should also, of course, be debugged), and should be relatively easy to correct. The second type of programming error is one that can only be detected after a long period of time, such as inaccurate monthly or yearly summaries. This kind of error might be unrecoverable. Prevention is the key: before you put such a program into use, test it extensively.

System Operator Errors

System operator errors are much more frequent than programming errors. If the system operator inadvertently cancels an important job, for example, data in your files might not correctly and accurately reflect the corresponding data in other files. It is important to detect such a situation as soon as possible and to correct affected files. As with a power failure, you look for the last successful checkpoint and reapply any transactions made since that checkpoint. One important point: the more often you perform checkpoints, the fewer transactions you will have to reapply.

Operator Errors

Operator errors are common. Examples include inadvertently powering off a display station, external disks, or entering inaccurate input. Because users of an application are usually less knowledgeable about the workings of the system, they might be unaware of an error when it occurs, and probably are unfamiliar with the consequences of the error. Again, prevention is important: restrict the application user to only running options from an assigned menu, and should the user inadvertently stray into a potential problem situation, allow the program to detect the problem and stop the user from going any further. You or the program (not the operator) should handle the recovery. Recovery from user errors should be treated in the same way as recovery from system operator errors.

Error Prevention

The following pages describe a few of the things you can do to prevent errors or failures.

Using the Automatic Response Facility

You can have the system automatically respond to displayed messages, by using the RESPONSE and NOHALT procedures. When a message has an automatic response, the message is not displayed; instead, the response is immediately used. This allows you to define a specific response that is to be used automatically, rather than having an operator enter a response.

See "Providing Automatic Responses for Messages" on page 14-4 for more information.

Preventing Job Cancellation

One of the major causes of system operator or user error is the unscheduled cancellation of jobs. Obviously, you do not want to take away the system operator's ability to use the CANCEL command; there is definitely a need for the system operator to cancel jobs in an emergency. However, you should consider preventing a user from canceling jobs via the Attn key.

Using the CANCEL-NO parameter of the ATTR OCL statement, you can prevent the Inquiry cancel options (2 and 3) from being displayed. See "Preventing Jobs from Being Canceled or Interrupted" on page 17-34 for more information.

Program Testing and Debugging

Although it might be considered an obvious statement, thorough program testing and debugging is the soundest way of minimizing programming errors.

As you test your programs and procedures, use the LOG OCL statement or the LOG procedure to ensure that all OCL statements in your procedures are recorded to the history file. Logging OCL statements to the history file allows you to trace the activity of your procedures and programs.

The DEBUG OCL statement allows you to follow the logic of your procedures as you run them. Using the DEBUG OCL statement, you can specify whether procedure control expressions and OCL statements in your procedures should be listed as they are evaluated, and whether the procedures should stop after each job step. Like the LOG statement or procedure, the DEBUG statement enables you to evaluate your procedures and programs as you test them.

The *System Reference* manual has more information about the LOG OCL statement and the LOG procedure, and about the DEBUG OCL statement.

Many of the programming languages allow you to perform debugging operations during compile time. Some of the programming languages allow you to override debugging operations, thus preventing the debugging information from being added to your source code; however, the debug option is a good way of detecting program errors before they occur. For more information about the problem determination and program debugging capabilities of the programming language that you use, see the appropriate language manual.

Extending the Size of a Library

When you are copying a member to a library and the member is too large to fit in the library, the system automatically creates an additional area on disk into which this member can be placed. The additional area is called a **library extent**. The system also displays a message informing the user when an extent is about to be created. See "Library Extension" on page 9-11 for more information.

You can prevent a library from having an extent by doing frequent condenses of the library, when you are developing an application for example. The CONDENSE procedure in the *System Reference* manual has information about condensing libraries.

Note: The system library (#LIBRARY) is not allowed to have an extent.

Extending the Size of a File

An extendable file is a disk file for which the system automatically attempts to allocate more space each time the file becomes full. Specifying an extendable file prevents your program from ending abnormally when there is no room in the file to add additional records.

You can specify a file as an extendable file by either of the following methods:

- FILE OCL statement. The EXTEND parameter specifies the number of blocks or records to extend the file.

- BLDFILE procedure. The number of blocks or records to extend the file is a parameter.

If you specify an extension value when the file is created, the extension value becomes an attribute of the file. In that case, the file is extended, if required, by any program using the file. You can also specify an extension value when your program uses an existing file. This allows you to override any existing extension values or to specify a new extension value for a file. For example, if a file was not specified as extendable when it was originally created, you can use the FILE OCL statement to make the file extendable while your program is using the file.

If the file is being shared and the various programs use different EXTEND values on the FILE OCL statement, the system uses the extension value of the program that caused the file to become full. When the file is extended, all programs sharing the file take advantage of the extra space, even if the EXTEND parameter was not specified on the FILE OCL statement.

The *System Reference* manual has more information about the FILE OCL statement and the BLDFILE procedure.

Extending the Size of a Folder

If you are copying a member to a folder and the member is too large to fit in the folder, the system automatically creates an additional area on disk into which the member can be placed. The additional area is called a folder extent. See "Folder Layout" on page 10-3 for more information.

Using the WAIT and FILE OCL Statements

The WAIT OCL statement causes a job to wait until a certain time of day, or until a certain period of time has passed. Once a WAIT OCL statement is processed, the job will not continue processing until a specified condition is met.

If a program needs to use a file or resource and that file or resource is being used by another program or procedure, an error might occur. The WAIT parameter of the FILE OCL statement can prevent other programs from waiting for the resources owned by another job.

The *System Reference* manual has more information about the WAIT and FILE OCL statements.

Allocating the Diskette or Tape Drive to a Job

You can use the ALLOCATE OCL statement to dedicate the diskette or a tape drive to a job. For example, you normally do not retain control of the drives between SAVE procedures. That is, after FILE1 is saved but before FILE2 is saved, another procedure on the system could use the drive. This would make your SAVE FILE2 procedure wait until that other procedure was done.

To avoid allocating the drive longer than necessary, you should use the DEALLOC OCL statement to deallocate the drive. For example, your procedure could save three files and then run another type of job that did not use the drive. You could use the DEALLOC OCL statement to allow other jobs on the system to use the drive.

The *System Reference* manual has more information about these statements.

Error Detection in Programs and Procedures

As you plan and code your applications, you naturally attempt to create programs that are free of errors. Nonetheless, errors might occur in the system facilities or the input data that your program uses.

You should anticipate these problems, and put code in your programs to handle them. If such code is not in your programs, not only could output data and files contain errors, but you might not even be aware of the errors.

Special error detection subroutines are provided by some of the programming languages, such as RPG II. If the programming language that you use does not provide error detection capabilities, you must include error-handling code in the program.

The action taken by your error-handling code can vary from correcting the situation to stopping the program. In any event, a message should alert your users to the situation. You can define the message, severity levels, and various procedures the display station operator should do when an error occurs.

When an error is detected, the program must determine what action to perform. The following is a list of recovery actions you should consider:

- Trying the display station operation again. For example, if your program cannot read a display format, the program can keep trying to read the display format.
- Saving the data used by the program and ending the program. If you do this, you must have a method of using the saved data and restarting the program.
- Ending the program without saving the data, or ending the program and discarding previously saved data. The program must be reloaded, and the data must be reentered.
- Releasing the display station from a MRT program.

Each programming language supplies ways to detect errors as they occur. The following describes a few of the error detection capabilities of each of the programming languages. For more information about the error detection capabilities and the error return codes for each programming language, see the appropriate language manual.

Error Detection in Assembler

After each $WSIO macroinstruction is run, you can test the return code in the display station DTF for a successful completion. If the display operation results in an error, then a user-coded error recovery routine for the specific return code can be run.

Error Detection in BASIC

The IOERR parameter of the OPEN, READ, WRITE, and CLOSE statements can be used to specify a line number that the program should branch to when a display station operation completes with an error. The ERR intrinsic function can be used to determine which type of error has occurred, and user-coded operations can be run for that return code.

Error Detection in COBOL

The EXCEPTION/ERROR declarative can be used to specify how the program is to handle input or output errors.

Error Detection in FORTRAN IV

The DISPLY subprogram starts the error detection capabilities of FORTRAN. The RTCODE parameter of the DISPLY subprogram contains the return code for each read or write operation to a display station. The return code can be checked by the program. You can also use the ERR parameter of the READ statement to determine which errors have occurred. The ERR parameter also contains the statement number that the program should branch to if an error occurs.

Error Detection in RPG II

The WORKSTN file information data structure (INFDS) contains status information that the program can check to determine what type of error has occurred. Using the information in the INFDS data structure, the program can then determine which error conditions can be processed by the INSFR subroutine.

Error Detection for MRT Programs

When a permanent I/O error is detected at a work station which is a requester of a MRT program, the system releases the work station and allows the MRT program to continue processing for the other MRT requesters. Because the work station has been released, the program is not canceled and other requesters of the program do not have to wait for a system message response. A return code of hex 24 is placed in the MRT program's DTF after the work station is released, indicating that the terminal has been released by the operator. This return code may be returned for both input and output operations.

User-Coded Error Detection Routines

You can also create your own detection routines for errors unique to your installation. The design and purpose of such coding would depend on your particular application.

Checking Return Codes in Procedures

Using the ?CD? substitution expression, your procedures can check a return code that indicates abnormal or problem situations such as disk failures. Procedure logic flow can be based on the substituted return code, and the procedure can branch to recovery routines if a particular return code is found. The *System Reference* manual has an explanation of the use of the ?CD? substitution expression and the return codes that you can use to control program logic.

Backup and Recovery

The goal of system backup plans and procedures should be to minimize the impact of any breakdown in the system and to recover from this breakdown as quickly and economically as possible.

Most backup procedures used for batch systems also apply to interactive systems. However, there are additional backup considerations in interactive systems. In interactive applications, the entire business can become dependent upon the system; therefore, the system must have reliable backup and recovery procedures.

Equipment Backup

Mutual backup plans between users with similar equipment have long been used for batch-oriented systems. However, this type of arrangement is generally not practical in an interactive environment. Even if an identically configured system could be located, the delays and difficulty of establishing the necessary data links would preclude relying on this approach to equipment backup.

Equipment backup plans can vary according to the business requirements and the cost of implementing those plans. Equipment backup can range from having a spare display station to completely duplicating the central system. The actual level of equipment backup provided must be determined by weighing the cost against the possible business loss if the backup is not there when it is needed.

Data Backup and Recovery

Because data (libraries, files, and folders) can be damaged or destroyed by incorrect modification, a system failure, operator errors, or a natural disaster such as a fire, keeping backup copies of vital information is recommended. Backup procedures typically involve copying the vital information kept on the system and then storing the copy in a safe location. For example, files and libraries can be copied to diskettes or tape, dated, labeled, and stored in a fireproof safe or at another location.

Loss of data could be disastrous to most businesses using data processing. Thus, a standard, well-documented, backup procedure should be established and used regularly. Typically, master files and all files related to the master files are saved at the same time. New data such as batches of transaction records can be copied to disk, diskette, or tape after they have been entered and edited. These saved transactions can be used during recovery procedures to make the master files current.

The level of data protection can vary greatly without a significant difference in cost implementation. Developing a backup plan and testing that plan can provide a potential return far greater than the time and effort involved. Developing a backup plan includes analyzing the effect of a system failure on each step of an application, defining the appropriate recovery procedures, and testing those procedures. It is important that the procedures be tested. A crisis situation is not the time to find the shortcomings of a backup plan.

For backup and recovery purposes, data can be divided into different categories. These are:

- Historical data
- Master files
- Data processed but not distributed
- Data logged but not processed
- Data received but not logged

Historical Data

Historical data or archive data is not used in day-to-day processing and is saved on some media (such as diskettes, tape, or microfilm) in a secure location, preferably off site. Any test of the backup plans should include trying to reconstruct current files from these historical files. The test should verify such considerations as:

- Are the files *always* available?
- Is the storage media compatible with the present hardware?
- Is the media protected against modification or deterioration?
- Are copies of the programs that process the data protected in the off-site location also? Is the program documentation available?
- How current are master files that are reconstructed directly from these historical files? What would be the cost of making the reconstructed files more current?
- Are operating procedures and run books available?
- Is the stored data the right data?

Master Files

Master files are those files used in day-to-day operations. Many users keep backup copies of all master files on site and update them on a daily basis. In this way, no more than one day's processing can be lost, and the transactions for that day are logged in the transaction log file. The transaction log file is retained until the daily update run has been completed.successfully.

Data Processed but Not Distributed

If a system failure occurs during the day's processing, then data that has been processed but not distributed must be considered. Master files have been updated, but the output of the application is stored within the system and is not recoverable. The transactions cannot simply be rerun because the master files would reflect double activity. A method that can be used is to have the application programs that process the transaction file flag the header record of all orders processed. The recovery plan would be designed to begin processing at the last order printed or distributed to a display station and to process all the following transaction records, but not actually update master files or memo fields on flagged orders.

Data Logged but Not Processed

When the recovery program has processed all the flagged orders, another category of data must be recovered: the records in the transaction file that have been received from the display stations but have not been processed. If a system failure occurs, the operators must not reenter these records. The operators must be notified as quickly as possible not to enter more data because a recovery is in process. The recovery program must scan the transaction file and identify for the operator the last record correctly entered in the transaction file. If transactions are linked together by display station within the transaction file, the logic of the recovery program can be much more straightforward.

Data Received but Not Logged

Data that has been received but not logged must be reentered by the operators. This data was in main storage at the time of the power or system failure and is now lost.

Loss of Transaction File Data

In many instances, the transaction file is the key factor in a successful recovery from a system failure. Transaction file data that is lost or unusable will have to be reentered. Transaction file data can be made less critical if master files are backed up on a daily basis. If a failure occurs in this case, a maximum of one day's input has to be reentered. If the master files are backed up twice a day, then a maximum of one-half day's input has to be reentered. Even orders that have been processed and printed have to be reentered so that the master files can be correctly updated. The printing of the output can be bypassed for the duplicates to prevent wasting forms.

Whichever method is used, the time required to reenter records should be balanced against the time required to back up files.

Printing the Configuration Member

The configuration member identifies and describes the machines, devices, and programs used by your system. If the configuration member is somehow destroyed during a system failure, you will have to reconstruct the configuration member. If you have a system with a large number of devices and programs, it would be particularly difficult for you to reconstruct the configuration member from memory. Therefore, you should have a printout of your configuration safely stored away so that you can reconfigure your system.

Using the main menu of the CNFIGSSP procedure, you can select an option to print your configuration member. The *Changing Your System Configuration* manual has more information about using the CNFIGSSP procedure to print out the configuration member.

IPL File-Rebuild Function

The system provides IPL file-rebuild as a recovery function that makes sure that a correct VTOC entry exists for every file, library, and folder. If the VTOC entry contains an error, the IPL file-rebuild function tries to correct the entry.

In addition, the IPL file-rebuild function does the following for data files:

- Displays the names of all files that are in error.

- If the VTOC entry for the file cannot be corrected, asks whether the file should be deleted.

- Deletes the VTOC entries of files that are deleted by the IPL file-rebuild function.

- Removes dump files whenever the current date exceeds the date of the task dump files and a request is made to remove them by using the IPL overrides.

The system checks each file to see if the data and the end-of-data pointer in the file are correct. If the data is not correct, the system attempts to reconstruct the end-of-data pointer for each file.

For indexed files, the system attempts to create the index if either the end-of-data pointer changed or the index has been flagged as defective by the system.

After the file-rebuild function is completed, the folder-rebuild function checks the folders in the VTOC and rebuilds any folders that were not closed before the IPL began. Folder-rebuild takes place only after a file rebuild; if you override the file-rebuild function at IPL, you automatically override the folder-rebuild function as well.

The files, libraries, and folders in the VTOC cannot be used until the rebuild function is completed.

Backup and Recovery Methods

Recovery is a series of steps that you follow or procedures that you run to restore data on the system. Recovery procedures can require removing all or some master files, restoring backed up master files, and running those procedures that updated files to post transactions in the order that they were originally processed.

Programs and procedures can be designed to restore and recover all files, inform the operators about the last items correctly processed, and allow operations to continue from that point. This effort might involve using additional fields in records and using additional calculations in programs. Also, new files, programs, and procedures might be needed, particularly for recovery in an interactive environment. The planning and programming effort might not seem costly in light of the potential results of inadequate backup and recovery procedures. Typically, businesses that are most dependent on their data processing system require the shortest recovery times and thus should develop the most elaborate backup and recovery procedures. Regardless of their complexity, backup and recovery procedures should be well-documented so that all operators use them correctly.

The following information describes three methods of backup and recovery. The first method requires the least design and programming effort, but probably requires the longest recovery time because transaction batches are not saved. The second method requires more planning and programming, but reduces the amount of recovery time required because reentering the transactions is not necessary. The third method requires the most planning and programming but provides the quickest way to recover data because the operator's involvement is minimized.

Method 1

This method requires the operator to periodically save master files and files that the application updates in order to establish a point from which to recover (restart) the application. For example, at the end of each day after all transactions have been posted, the operator might run a procedure that contains SAVE procedures to back up all master files and their related files on diskette or tape.

Operators should keep a log of the work they do on the system. This manually kept log must be accurate if it is to be relied upon during recovery. One method of keeping a log is to use the following sample run sheet.

RUN SHEET

Display Station ID____________________ Date____________________ Page_______

Menu, Job, or Command Name	Operator's Initials	Start Time	Stop Time	Comments, Halts, or Messages

Another method of obtaining a log of work done on the system is to print the history file.

This recovery method consists of the operator (1) deleting files from disk, (2) restoring the backup copies from diskettes or tapes to establish a point from which to recover, and (3) reprocessing all transactions that have been entered since the last backup was done.

All of the work done since the last backup must be redone. Because they are not saved, transaction batches must be reentered. This method might be adequate for a business that processes low volumes of data and that frequently backs up its data.

Method 2

This method requires the operator to (1) periodically save the master files and their related files and (2) save batches of transactions at logical breakpoints in the application. For example, at the end of each day after all transactions have been posted, the operator runs a procedure that contains SAVE procedures to back up master files and their related files on diskettes or tapes. As part of the transaction-posting procedure run during normal processing, a batch of transactions is saved on diskette or tape and deleted from disk. The operator labels the diskettes or tapes that contain the transactions to document the sequence in which the batches have been saved. Also, the operator lists the names of the procedures in the order that they were run.

The recovery method requires the operator to (1) delete the files from disk, (2) restore the backup copies from diskettes or tapes to establish a point from which to recover, and (3) reprocess the application's procedures in their original order using the saved copies of the transaction batches. The operator uses the information labeled on the diskettes or tapes to ensure that the transaction batches are restored in the correct order.

This method eliminates the entering of transactions that was required in method 1.

Method 3

This method requires code to be included in an application's procedures to do the following:

- Periodically back up the master files and their related files.
- Automatically back up batches of transactions on disk, diskette, or tape at logical breakpoints in the application.
- Assign names and sequence numbers to these batches of transactions.
- Keep a history of all procedures that the operator runs after the previous backup.
- Provide a common recovery procedure.

The recovery method requires the operator to run the common recovery procedure, which lists the control file and restores the files. The operator uses the history to rerun the application's procedures in their original order. The common recovery procedure could prompt the operator to insert the proper backup diskettes or tapes in the correct sequence.

This recovery method uses a program-generated control log, which is more accurate than a manually kept log. Because unnecessary procedures such as reprinting statements or reports could be skipped during recovery, this method provides the quickest recovery of the three methods. This method is similar to the backup and recovery procedures used in some IBM licensed application programs.

The following example shows one method of restoring a particular file from a diskette or tape:

Step	Procedure	Action
1	CATALOG	Determine the file to be restored.
2	RESTORE	Restore the file from backup diskettes or tapes.
3	CATALOG	Verify that file has been restored.

Service Aids Procedures

The following pages describe the system's service aid procedures. You will not normally have to run these procedures; however, you may be asked to run some of these procedures to do problem determination and correction. The *System Problem Determination* manual has information about problem determination. The *System Reference* manual has more information about these service aid procedures.

SERVICE Procedure

The SERVICE procedure displays a menu that allows you to do the various tasks you may be asked to do for problem determination and correction. These tasks include:

- Running the service aid procedures for determining program-related problems (such as DUMP, APAR, DFA, and TRACE)
- Running the service aid procedures for determining hardware-related problems (such as ERAP)
- Copying and applying program temporary fixes (PTFs) to the licensed programs
- Adding entries to or listing the system service log
- Running the PATCH procedure to display or change disk or diskette sectors

APAR Procedure

The APAR procedure collects diagnostic information that helps software service people to correct programming problems that might occur in the system.

DFA Procedure

The DFA (dump file analysis) procedure retrieves selected information from a dump file, formats the information, and either prints or displays it. The DFA procedure can also be used to format dump files copied to diskette by the APAR procedure.

DUMP Procedure

The DUMP procedure prints or displays dump files from disk, diskette, or tape created by the APAR procedure.

ERAP Procedure

The ERAP procedure displays or prints data that was logged for the devices on the system.

PATCH Procedure

The PATCH procedure displays selected disk or diskette sectors and allows you to modify the data in those sectors. You can modify disk or diskette information by replacing data displayed with new data.

PTF Procedure

The PTF procedure allows you to apply program temporary fixes (PTFs) to a specified library.

SETDUMP Procedure

The SETDUMP procedure allows you to debug programs running in main storage at predetermined breakpoints or addresses without having to stop the main storage processor.

TRACE Procedure

The TRACE procedure allows you to keep a history of events that occur in the system.

Remote Operation/Support Facility

Remote Operation/Support Facility (ROSF) is the capability provided with the SSP that allows an operator at a remote display station (and an optional printer) to enter many system commands to provide you with operational, programming, and technical support. In addition, your remote support group can use ROSF to perform problem determination.

The *Remote Operation/Support Facility Guide*, SC21-9060, has more information.

Chapter 20. Ideographic Data Concepts and Considerations

This chapter describes ideographic data and identifies things to consider when you are using ideographic characters.

Ideographic characters are available only with the ideographic version of the SSP.

Special display stations and printers are required to display, enter, and print ideographic characters.

Ideographic Data Concepts

The ideographic version of the SSP allows users of the system in some countries to work in their native language. The common written languages of some countries consist of thousands of symbols, each of which may represent a single word or concept.

Because these languages include thousands of characters, a single byte cannot uniquely represent each character in the system. The ideographic version of the SSP allows each of the characters to be represented by 2 bytes of data. Characters that are defined by 2 bytes of data are called **ideographic** characters. The usual 1-byte characters are called alphanumeric or alphanumeric/Katakana characters.

Each ideographic character is defined by a matrix pattern of dots. The size of the matrix depends on which device is used to display or print the ideographic character. The matrixes are either 24 by 24 dots or 18 by 18 dots, depending on the device used to display or print the characters. See the operator's guide for your display station or printer for information about how characters are displayed or printed.

Some versions of the SSP support ideographic display stations only. This saves the disk space by not retaining the nonideographic version of the display formats.

Shift-out and Shift-in Characters

All ideographic data streams must be preceded by a shift-out (SO) character. The data must be followed by a shift-in (SI) character. SO tells the system to shift out of normal alphanumeric/Katakana mode; SI tells the system to shift into normal alphanumeric/Katakana mode. The SO and SI characters serve as a signal to the system that any data between them should be interpreted as a string of 2-byte characters. The shift characters' hexadecimal equivalents are:

Shift Character	Hex Value
SO	0E
SI	0F

The ideographic display stations provide a special keying sequence to allow the operator to enter the shift characters. Another key allows them to be displayed. See the operator's guide for your display station for information about how to enter ideographic characters.

Ideographic Character Sets

The ideographic characters available on the system are divided into the following categories:

- The **basic** ideographic character set contains 3,707 characters. These 3,707 characters are stored in the display stations and printers. No special SSP support is necessary to process the basic set of characters. The basic character set includes:
  - Commonly used Kanji characters
  - Special symbols (such as circles and arrows)
  - Alphanumeric characters
  - Katakana characters
  - Jamo
  - Hanjuel
  - Hiragana (a cursive form of Katakana) characters
  - Greek characters
  - Russian characters
  - Roman numerals
- The **extended** ideographic character set contains 3,483 less commonly used ideographic characters. The matrix patterns that define these characters are stored in IBM-supplied system files. Because different ideographic devices require different sized character matrixes, the system has two extended character files:

  #EXT1818 contains the IBM extended character definitions for the 18x18 dot matrix characters and is available for Japan only.

  #EXT2424 contains the IBM extended character definitions for the 24x24 dot matrix characters and is available for Japan and the SEAR countries.
- The **user-defined** ideographic character set can contain up to 4,370 additional characters. You can use the character generator utility (CGU) to define these characters as required. CGU stores the characters you create in #EXT1818 and #EXT2424 along with the characters provided by IBM.

  Note: The character generator utility is available for Japan only.

  See "Creating Ideographic Characters" on page 20-9 for more information.

All the characters stored in the extended character files, both IBM extended and user-defined, are called EXTN characters. The files are often called the EXTN files. The SSP support necessary to process the characters in these EXTN files is described in "Processing Extended Characters" on page 20-8.

Note: *Extended and user-defined ideographic character sets are valid for Japan, Korea, Taiwan, and PRC. They are used primarily for the 5227 printer which cannot store the entire character set at the device and must access #EXT2424 for the definition of the user-defined and extended characters.*

Hex Representation of Ideographic Characters

Each ideographic character is represented by a 2-byte hexadecimal code. The first byte of an ideographic character is called the **ward number** and points to a particular code table. The second byte is called the **location number** and provides the location of the character in the code table. Each byte can contain values from hex 41 to hex FE. The only exception is an ideographic blank which is represented as hex 4040.

The following table shows the allowed ward and location numbers for each of the three Japan-only character sets:

Japan- Only Character Set	Ward Number First Byte (in hex)	Location Number Second Byte (in hex)
IBM Basic	41 through 55	41 through FE
IBM Extended	56 through 68	41 through FE
User-defined	69 through 7F	41 through FE
Reserved	80 through FE	- -

The following tables show the ward and location numbers for the 5227 printer support character set. Note that the ROS character set is not equivalent with the Basic character set.

5227-001 Japan 4K ROS Character Set	Ward Number First Byte (in hex)	Location Number Second Byte (in hex)
Basic=ROS	41 through 55	41 through FE
Extended	56 through 68	41 through 7F
User-defined area	69 through 7F	41 through FE

Note: *The above table is also for devices other than the 5227 printer.*

5227-002 **Korea 4K ROS** **Character Set**	**Ward Number** **First Byte** **(in hex)**	**Location Number** **Second Byte** **(in hex)**
Special symbol/ Hiragana/ Katakana/ Greek/ Roman = ROS	41 through 46	41 through FE
Hanja (Chinese) = ROS part of total Hanja set	50 through 52	41 through FE
Hanja (Chinese) part of total Hanja set (not in ROS)	53 through 67	41 through FE
Hanjuel and Jamo (scattered) = ROS	84 through D3	41 through FE
User-defined area (not in ROS)	D4 through DD	41 through FE

5227-003 **Taiwan 4K ROS** **Character Set**	**Ward Number** **First Byte** **(in hex)**	**Location Number** **Second Byte** **(in hex)**
Non-Chinese = ROS	41 through 44	41 through FE
Special Symbol = ROS	45 through 46	41 through FE
Chinese character first set = ROS part of first set	4C through 5C	41 through FE
Chinese character first set (not in ROS)	5D through 68	41 through FE
Chinese character first set (not in ROS)	69 through 91	41 through FE
User-defined area (not in ROS)	D0 through DD	41 through FE

5227-005 PRC 8K ROS Character Set	**Ward Number First Byte (in hex)**	**Location Number Second Byte (in hex)**
Non-Chinese = ROS	41 through 46	41 through FE
Special Symbol = ROS	48 through 5C	94 through 89
Chinese level-2 = ROS part of first set	5C through 6C	94 through 94
Reserved (not in ROS)	5D through 68	41 through FE
User-defined area	76 through 7F	41 through FE
Reserved	80 through FE	41 through FE

Ideographic Number

Each ideographic character is also assigned a 2- to 5-digit ideographic number.

Ideographic display stations have an alternative entry mode which allows the operator to enter any ideographic character using its number. If a character is not represented on the keyboard or if the operator is using an alphanumeric/Katakana keyboard, the character can **only** be entered by using the alternative entry mode.

Ordinarily, users are not concerned with the hexadecimal representations of the characters. The ideographic number associated with a character can be calculated from the 2-byte hexadecimal code:

1. Subtract hex 4000 from the hexadecimal code.

2. Convert the number to decimal.

Dual Language Support

The system allows you to create and use ideographic and alphanumeric/Katakana versions of messages and display formats. System messages and display formats are also available in ideographic and alphanumeric/Katakana versions. This allows you to direct alphanumeric/Katakana messages and display formats to nonideographic display stations. With some versions of the SSP, there is not an alphanumeric/Katakana version of the display formats. Nonideographic display stations should not be attached if some versions of the SSP are loaded.

Ideographic Sessions

The system provides an option for each operator to specify that system messages and formats which are directed to that operator should be displayed.

At an ideographic display station, the Sign On display contains an extra prompt that indicates whether an ideographic session is to be started. System messages and display formats are directed to that display station throughout the session if an ideographic session is started.

The extra prompt is not displayed for the ideographic display station attached to the Kanji version of the SSP. A display station will automatically be placed in an ideographic session for the Kanji version of the SSP.

Ideographic sessions are not possible if the display station is not capable of displaying ideographic characters. All system messages and display formats directed to a nonideographic display station consist of Katakana and other alphanumeric data. These are ordinary 1-byte characters.

Programming Considerations

User-written programs can show display formats that contain ideographic data at any display station. The formats are correctly displayed at an ideographic display station, regardless of the session type. But at a nonideographic display station, the 2-byte data is interpreted as 1-byte data and the result is meaningless.

You can prevent this from happening by using the following procedure control expression:

```
IF DSPLY-IGC
```

This allows you to determine whether the procedure is being run from a display station in an ideographic session. The *System Reference* manual has more information.

Processing Extended Characters

Extended ideographic (EXTN) characters are stored in two system files called #EXT1818 and #EXT2424. For an EXTN character to be printed, the character's matrix pattern must be read from the EXTN file that contains the correct matrix size for the device on which it to be displayed or printed, and then transmitted to the device.

The system provides a special program, called the EXTN task, to handle the processing of EXTN characters. This program is always active when the ideographic version of the SSP is installed and ideographic devices are configured. The EXTN task maintains the EXTN characters in the display station or printer by transmitting the proper matrix pattern whenever that character is required. See the appropriate operator's guide for information about how many extended ideographic characters can be stored in the display stations or printers.

Work station data management automatically handles EXTN character processing for input operations from display stations. When an operator types an ideographic number, the EXTN task replaces the ideographic number with the hexadecimal representation of the character. If an EXTN character is encountered, the EXTN task ensures that the correct ideographic character is shown at the display station, by using the EXTN character files.

For output operations, you can turn EXTN processing on and off using the SYSLIST, PRINTER, or WORKSTN OCL statements.

The *System Reference* manual has more information about these OCL statements.

Note: For SEAR DBCS countries, the EXTN processing is only performed for printers (primarily 5227). EXTN processing is not used for display stations because they are full-font capable.

Displayed Output

If EXTN processing is specified, work station data management scans the output data for EXTN characters. When an EXTN character is found, work station data management works with the EXTN task to ensure the correct ideographic character is displayed. If EXTN processing is not specified, work station data management does not scan the data, and any EXTN characters encountered are displayed as the default ideographic character.

Printed Output

If EXTN processing is specified, printer data management scans the output data for EXTN characters. When an EXTN character is found, printer data management works with the EXTN task to ensure that the correct ideographic character is printed. If EXTN processing is not specified, printer data management does not scan the data, and any EXTN characters are printed as the default ideographic character.

Programming Guidelines for Ideographic Data

This section describes how you can:

- Create ideographic characters
- Create ideographic message members
- Create ideographic display formats
- Sort ideographic data
- Manipulate bytes of ideographic data

Creating Ideographic Characters

The character generator utility (CGU) is an interactive utility program that allows you to define and maintain ideographic characters on your system.

CGU allows you to define just the 24x24 character, just the 18x18 character, or both the 18x18 and 24x24 characters. Characters are defined by entering the dot pattern in an 18x18 or 24x24 matrix work area, depending on which size character is being defined. CGU stores the character's matrix pattern in the appropriate extended ideographic character set file.

Note: The character generator utility (CGU) is available for Japan only. Other countries use a PC-based character generator utility.

The *Character Generator Utility Guide* has more information about CGU.

Creating Ideographic Message Members

The CREATE procedure allows you to build message members which contain two portions.

- The first portion contains messages which are used for nonideographic sessions.
- The second portion contains messages which are used for ideographic sessions.

Note that the same message identification code is used in each portion of the message member, so that two versions of each message are available. For more information about message members, see Chapter 14, "Messages and Message Members."

Creating Ideographic Display Formats

You can create ideographic displays by using S- and D-specifications and the FORMAT procedure, or by using the screen design aid (SDA) utility. For more information about display formats, see Chapter 13, "Displays."

Sorting Ideographic Characters

The ideographic sort program allows you to select records based on ideographic fields and to sort the records based on ideographic control fields. The ideographic sort program is available for Japan only.

The *Ideographic Sort Guide* has information about sorting ideographic data.

Manipulating Bytes of Ideographic Data

Manipulating ideographic data requires special care to preserve the proper relationship between the SO and SI pairs. Problems can occur when you improperly truncate, concatenate, or separate ideographic character strings. An unmatched SO character can cause alphanumeric data to be incorrectly interpreted as ideographic data. A missing SO character will cause ideographic data to be interpreted as alphanumeric data.

The information in this section describes possible situations that may occur when you are truncating, concatenating, or separating ideographic data.

Truncating Ideographic Data

The following are ways of handling truncated records:

- If the last byte of the remaining data is an SO character, delete the SO character.

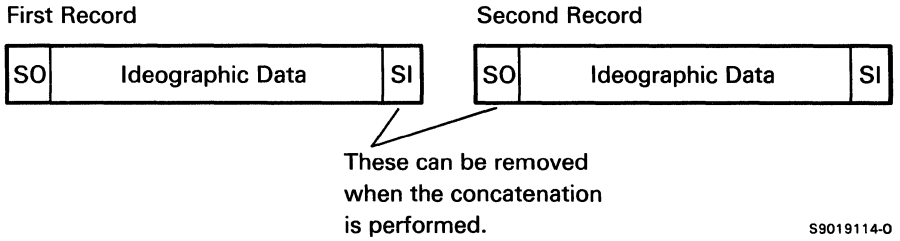
- If the last byte of the remaining data is the ward number of an ideographic character, replace the last byte with an SI character.

- If the last byte is the location number of an ideographic character, delete the last byte and replace the next to last byte with an SI character.

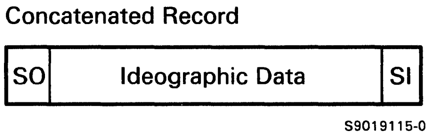
Concatenating Ideographic Data

When ideographic data is concatenated, you should be aware of the following situations:

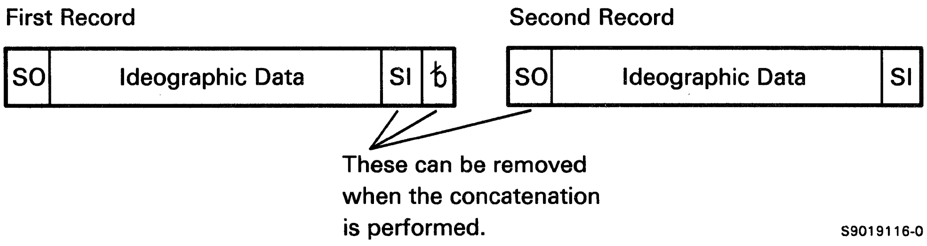
1. Ideographic data fills the first record, and the last position contains an SI character.



If the **second** record begins with an SO character, the SI character at the end of the first record and the SO at the beginning of the second can be deleted when the records are combined. If necessary, add blanks to the end of the concatenated record.

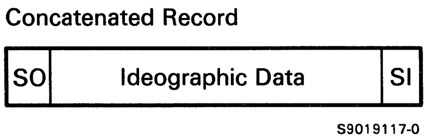


2. Ideographic data fills the first record except for the last byte which is a blank (ƀ in the diagram).



These can be removed when the concatenation is performed.

If the second record begins with an SO character, delete the SI character and blank at the end of the first record and the SO character at the beginning of the second record. If necessary, add blanks to the end of the concatenated record.



Note: Constant ideographic data specified on D specifications for display formats is concatenated differently for certain situations. Appendix C of the ***Creating Displays*** *manual has information on concatenating constant ideographic data for $SFGR display formats.*

Separating Ideographic Data

When you separate an ideographic data record, extra SO and SI characters may need to be inserted so that neither of the individual records contain an unmatched SO or SI.

Chapter 21. Summary of Design Considerations

This chapter summarizes the major topics discussed in this book.

Application Design Steps

Step and Page	Application Users	Application Designer
"Step 1. Definition" on page 2-5.	Explain current methods. Provide complete examples of input and output for current methods. Explain sources and destinations of information. Describe desired methods for existing and additional application functions.	Identifies what users do. Defines input and output. Identifies what the application should do. Defines application functions. Makes preliminary schedule. Reviews work with users. Documents the definition.
"Step 2. General Design" on page 2-12.	Help define the application functions. Review the design.	Specifies the input, processing, and output for the application in general.
"Step 3. Detailed Design" on page 2-14.	Review and agree upon displays and reports.	Designs printed output, files, and displays.
"Step 4. Program Design" on page 2-18.	Help define the sequence and frequency of functions. Identify deadlines and constraints.	Designs programs and then codes them.
"Step 5. Testing" on page 2-20.	Help define test cases, test the application, and check the results.	Codes test cases. Ensures that each program and all application functions work.
"Step 6. Conversion and Installation" on page 2-23.	Help convert information.	Trains application users. Ensures all existing information is converted to a form usable by the new application. Loads the application components onto the system.
"Step 7. Operation" on page 2-24.	Use the application.	Turns the application and documentation over to the users.

Printed Output

Design Considerations	For More Information, See
Consider printing information that: • Is not needed immediately or often • Is not needed by several people • Does not change frequently • Has large volumes of output (several pages of information) • Must be sent to several places	"Printed Output Considerations" on page 3-13.
Use a printer spacing chart to design your output. Consider: • Leaving enough space on the edges of the report so you can bind it • Separating each field on the report by at least one space • Grouping similar items • Numbering all pages of the report • Providing meaningful headings for data on the report	"Designing Your Printed Reports" on page 3-14.

Disk Storage

Design Considerations	For More Information, See
For systems with more than one disk drive, try to balance how the system uses the disks. This will minimize the time the system uses to search for and read or write data.	"Placing User Files and User Libraries on Disk" on page 4-15.
Collect the free spaces on disk to make larger areas of disk space available.	"Reorganizing Disk Space" on page 4-21.

Diskette Storage

Design Considerations	For More Information, See
When you have several job steps that copy information on diskette, you should allocate the diskette drive to your job so that you can retain control of the diskette drive during that job.	"Allocating the Diskette Drive to a Job" on page 5-14.
When a job contains several SAVE, SAVELIBR, or SAVEFLDR procedures or several RESTORE, RESTLIBR, or RESTFLDR procedures, you should allocate the diskette drive to your job so that you can read or create a sequential series of diskette files.	"Creating a Sequential Set of Files on Diskette" on page 5-15.

Tape Storage

Design Considerations	For More Information, See
When you have several job steps that copy information on tape, you should allocate the tape drive to your job so that you can retain control of the tape drive during that job.	"Allocating the Tape Drive to a Job" on page 6-12.
When a job contains several SAVE, SAVELIBR, or SAVEFLDR procedures or several RESTORE, RESTLIBR, or RESTFLDR procedures, you should allocate the tape drive to your job and use the LEAVE parameter so that you can create a sequential series of tape files.	"Creating a Sequential Set of Files on Tape" on page 6-13.

Records

Design Considerations	For More Information, See
When you are designing records that contain numeric fields, select the most appropriate format: zoned decimal, packed decimal, binary, or floating point. Select a format that can be used by the processing program (for example, do not select floating point for a file to be used by Query/36).	"Numeric Fields" on page 7-3.
Document record layouts so that your programs are easier to create and maintain.	"Documenting Record Layout" on page 7-11.

Files

Design Considerations	For More Information, See
When you create a file, you should decide which file organization is most appropriate to use (sequential, indexed, or direct). You must also decide which access method is most appropriate to use (consecutive, sequential by key, random by key, or random by relative record number).	"File Organizations" on page 8-13, "Accessing Files" on page 8-32, and "Choosing a File Organization" on page 8-43.
Blocking records and index entries is useful when your programs are likely to process several records at a time. By specifying a large blocking factor, you might improve the performance of your programs.	"Blocking Records and Index Entries" on page 8-60.
Using file sharing, you can allow several programs to access the same file at the same time.	"Sharing Files" on page 8-73.
You can secure your files to prevent unauthorized use or changes.	"Securing Files" on page 8-12.

Libraries

Design Considerations	For More Information, See
As you develop applications, you may find it easier to have a development library that contains the items you are working on and a production library that contains the items you have tested. Using multiple libraries can help you organize and control your work. For example, you can use any of the following approaches: • Create a separate library for each application. • Allow each user to have a separate library. • Create a library for each shift of users. • Create separate libraries for programs, procedures, messages, and display formats.	"Uses of Libraries" on page 9-5.
You can secure your libraries to prevent unauthorized use or changes.	"Securing Libraries" on page 9-13.

Folders

Design Considerations	For More Information, See
You can organize your document folders according to the kinds of documents contained in them.	"Uses of Folders" on page 10-2
You can secure your folders to prevent unauthorized use or changes.	"Securing Folders" on page 10-6.

Subdirectories

Design Considerations	For More Information, See
You can organize your documents in a folder according to the types of documents and group them in subdirectories.	"Using Subdirectories" on page 11-2
You can secure your subdirectories to prevent unauthorized use or changes to the members.	"Securing a Path to a Subdirectory" on page 11-10.

Menus

Design Considerations	For More Information, See
When you are designing menus, remember to: • Use free-format menus • Use both uppercase and lowercase letters • Make the menu title and menu options meaningful • Put options in logical order Also, chain menus together. For example, have a general or main menu from which users can call several more detailed menus.	"Designing Menus" on page 12-7.
Creating help text for menus can provide helpful information for your users, should they need it.	"Creating and Displaying Help Text for Menus" on page 12-12.

Displays

Design Considerations	For More Information, See
When you are designing display formats, remember to: • Choose the appropriate form for your display. • Identify the displays. • Provide meaningful headings. • Provide one idea for each display. • Respond to operator input.	"Display Design Considerations" on page 13-11.
Creating help text for displays can provide helpful information for your users, should they need it.	"Creating Help Text for Your Displays" on page 13-27.
If you use remote display stations: • Reduce the amount of data transmitted. • Increase the line speed. • Send only the minimum amount of data required. • Use an erase input fields operation to remove the contents of an input field rather than displaying the entire format again.	"Designing Displays for Remote Display Stations" on page 13-19.
If you have a color display station, take advantage of the colors you can control using the display format.	"Using Color to Highlight Data" on page 13-16.

Messages and Message Members

Design Considerations	For More Information, See
When you design your application, you can group messages by: • Program type • Displayed or printed messages	"Designing Message Members" on page 14-4.
You can have the system automatically respond to displayed messages.	"Providing Automatic Responses for Messages" on page 14-4.

Programs

Design Considerations	For More Information, See
Interactive programs use display stations to display and allow data to be entered. Batch programs have little or no operator interaction, and can be used to process groups of information that have accumulated over a period of time.	"Interactive Programs" on page 16-2.
You can design your programs several ways: • One large program for each user • One large program shared by all users • Several small programs with each user running a separate copy of the program • Several small programs with all users sharing one copy of the program	"Application Structure" on page 16-11.

Jobs and Job Processing

Design Considerations	For More Information, See
Using processing priorities allows you to control which jobs may run faster than other jobs on the system.	"Processing Priorities" on page 17-12.
Using the job queue allows you to have batch jobs processed sequentially. You can also use the job queue to process your batch jobs at a later time (for example, second shift). You can use job queue priority to change the order of the jobs in the queue.	"Job Queue" on page 17-14.

Procedures

Design Considerations	For More Information, See
Use procedures to: • Run several programs by entering one procedure command • Prompt for parameters for jobs • Check entered parameters for errors	"Advantages of Using Procedures" on page 18-2.
Use parameters to pass information and variables to the procedure. A procedure can have a maximum of 64 parameters, and each parameter can have up to 128 characters.	"Procedure Parameters" on page 18-3.

Appendix A. Access Algorithms for Direct Files

This appendix describes some access algorithms you can use with direct files. These methods may help you design your direct files more efficiently.

A key to designing and implementing a direct file is defining an access algorithm that satisfies the processing requirements for the file while preserving the advantages of direct files.

In the simplest case, relative record numbers are assigned sequentially. The first record placed in the file has relative record number 1, the second record has relative record number 2, and so on.

In another simple case, a control field in each record is used as its relative record number. For example, loan number 3456 could be used without change as relative record number 3456. A control field should be used directly as a relative record only if there is not a large number of unused values within the range of values for the control field. If there are many unused values and, therefore, unused record positions, an algorithm should be defined to reduce the size of the file.

Choosing an Access Algorithm

An **access algorithm** is whatever method is used to determine the position to be occupied by each record. The algorithm can be simple or complex. In either case, the algorithm must yield a positive, whole number as a relative record number.

One method is to use a formula as an algorithm to determine the record number. For example, if loan numbers start with 1001, then loan number 3456 could be relative record number 2456 (3456 minus 1000). The formula can be as complex as you need to make it. Refer to "Examples of Access Algorithms" on page A-4 for more information and examples.

Another method is to use a control field that contains alphameric data. An algorithm would then convert the alphameric data to a relative record number. Refer to "Handling Synonym Records" on page A-3 for an example of using a customer name as the control field.

The choice of an access algorithm and, ultimately, the decision about whether to use a direct file is usually based on how well synonym records can be handled. A **synonym record** is a record in a direct file whose control field yields the same relative record number as another control field. The first record with a given relative record number is called the **home record**. If the handling of synonyms requires a significant number of additional disk accesses, one of the important advantages of the direct file is lost. Also, because the access algorithm and the synonym code must reside in each program that uses a direct file, a risk is involved: if the algorithm and synonym handling are revised, you might need to rebuild files and modify all the programs that use those files.

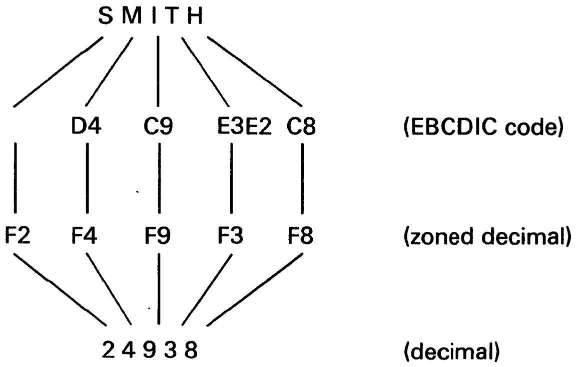
Handling Synonym Records

Synonyms can be handled in many ways. Two of the common ways are:

- Place synonyms in a separate part of the file.
- Place synonyms in the next available blank location.

In these two methods, the record must contain a pointer to the synonym record. If two or more synonyms exist, the first synonym contains a pointer to the second synonym, and so on.

For example, assume that the control field for a file is the first five characters of the customer's name. The file contains space for 40,000 records and allowance for three synonyms for each home record. The customer's name is converted to a decimal value as follows:



S9019093-0

The decimal value is then divided by 9999:

```
  24938 / 9999 = 2.4940
```

Ignoring the whole number of the quotient, you would calculate the location as follows:

```
  (4940 x 4) + 1 = 19761
```

Because many customers may have the same name, such as Smith, the program may have to read records 19761, 19762, 19763, and 19764 to find the correct Smith. If extra synonyms are required, the third synonym could point to the next available space in the file (possibly an unused synonym location for the next home record). Another possibility, to reduce the number of synonyms, is to accept six or more characters from the customer name.

Examples of Access Algorithms

The following examples illustrate three approaches to designing access algorithms for direct files.

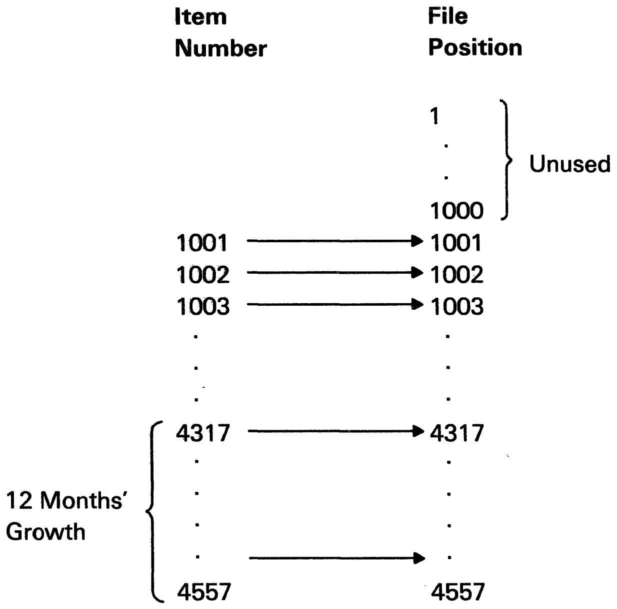
Example 1

In this example, the major goals are to build a file in which:

- The records can be accessed with an average of one disk access.
- The disk space used for the file should contain little unused space.
- The file should easily accommodate new records.

Defining the Algorithm: In this example, an indexed item file is to be converted to a direct file for an interactive order entry application. The key field is a five-digit item number; four digits are assigned by the user, and the fifth digit is a check digit. The four digits start with 1001, and the user merely assigns the next sequential number to new items. Deleted item numbers are not reused until item 9999 has been taken. Approximately 20 new items are added per month, and four items are deleted. Currently, the highest number is 4317, but the file contains only 2,812 items.

The algorithm could be stated this way: the direct file position for each record is equal to the four-digit item number. Assume that the new record is a few bytes larger than the old record and that the file also accommodates 12 months of growth before reorganization. The algorithm requires a file containing 4,557 record positions. The items are related to direct file positions as follows:



S9019100-0

This approach, while yielding no synonyms, uses only two-thirds of the record positions, and most of the unused space is at the beginning of the file.

The algorithm can be revised to state: the direct file position for each record is equal to the four-digit item number minus 1000. The file now requires 3,557 positions with the following relationships:

Item Number	File Position
1001 →	1
1002 →	2
1003 →	3
.	.
.	.
.	.
4317 →	3317
.	.
.	.
.	.
4557 →	3557

S9019101-0

This approach, also yielding no synonyms, uses 85 percent of the record positions. The unused portions are embedded randomly within the file where items have been deleted. Although each record requires only one disk access, the file size still is 15 percent larger than the data portion of the indexed file it is to replace.

Now assume that the algorithm is further revised to state: the direct file position for each record shall be found by subtracting 1000 from the four-digit item number, multiplying the difference by 0.85, and half-adjusting the result. The file then occupies 3,023 positions with the following relationships:

Item Number		File Position
1001	→	1
1002	→	2
1003	→	3
.		.
.		.
.		.
4317	→	2819
.		.
.		.
.		.
4557	→	3023

S9019102-0

This approach uses 99 percent of the record positions, and the file size is only 1 percent larger than an indexed file. It has, however, introduced the possibility of synonym records. For example, if item 1004 exists, it is assigned to direct file record position number 3 (same as item 1003). Similarly, items 4316 and 4317 conflict, as do items 4556 and 4557. Thus, the refinement of the algorithm to meet the second major goal, minimum file space, may now affect the first goal, minimum disk accesses, because synonym records take a minimum of two accesses.

Handling Synonyms: The method that is used to handle synonyms must accomplish two goals: minimum accesses and minimum file space. The first step is to define (program) the manner in which a record will be placed in an alternative position when its home location is filled.

Further analysis of the item file in this example might offer some suggestions for synonym handling. Note that, in this example, a synonym occurs about once in seven records.

The previous algorithm caused the following mapping (asterisks identify synonyms):

Item Number	File Position	Item Number	File Position	Item Number	File Position
1001	1	1009	8	1017	14*
1002	2	1010	9*	1018	15
1003	3*	1011	9*	1019	16
1004	3*	1012	10	1020	17
1005	4	1013	11	1021	18
1006	5	1014	12	1022	19
1007	6	1015	13	1023	20*
1008	7	1016	14*	1024	20*

S9019103-0

Approximately one in seven item numbers is unused because of deleted items; the file is only 86 percent full. Thus, you might expect to find an unused position in the direct file about as frequently as the synonyms occur.

Assume that the method of handling synonyms can be stated as follows: a synonym record is placed in the next higher numbered position that is unused. Because the file uses only 85 percent of the range of numbers, 15 percent of the numbers are not used because they are deleted. However, the deleted numbers are randomly distributed throughout the range of numbers. Thus, some positions will be available in the file for synonym records. About every seventh number will be a synonym. Assume that of the first 40 item numbers, items 1007, 1008, 1015, 1017, 1020, and 1039 are among those deleted numbers.

Item Number	File Position	Item Number	File Position	Item Number	File Position
1001	1	1016	14	1030	26
1002	2	1018	15	1031	**
1003	3	1019	16	1032	27
1004	6	1021	18	1033	28
1005	4	1022	19	1034	29
1006	5	1023	20	1035	30
1009	8	1024	33	1036	31
1010	9	1025	21	1037	**
1011	13	1026	22	1038	32
1012	10	1027	23	1040	34
1013	11	1028	24		
1014	12	1029	25		

S9019104-0

Note the following:

- Item 1031 will be placed after position 34.
- Item 1037 occupies a higher numbered position than item 1031.
- File positions 7 and 17 are unused.
- After accessing a record, the program has to verify that the record is the one that was requested. If it is not, the program must access a synonym.
- No more than two items have the same relative record number. Thus, most records require no more than two disk accesses.

Note: This example assumes that records are loaded into locations before synonym records are loaded in a second run. This example also assumes that few records are added. If records are added after the synonyms are loaded, the locations for the added records may be occupied by synonyms. Thus, the added record becomes a **pseudo synonym**. *If many records are added, most have to be handled as synonyms. In this situation, the technique described here may be less useful because performance tends to be degraded as records are added.*

In this synonym-handling technique, the average synonym should be close to the first position searched. Thus, a second access is necessary approximately 15 percent of the time, and this access should find the record not too distant from the original location.

At this point, the file should be loaded, and the synonyms added in a second run. As the synonyms are added in the next available higher numbered position, a synonym pointer in the record has to be updated to point to the synonym record position.

Example 2

Assume that a customer master file contains three types of records (A, B, and C) for three types of customers. These records are in an indexed file, which has keys. Type A records have customer numbers from 10000 to 49999; type B records are numbered from 60000 to 79999; and type C records from 90000 to 99999. Each type of record is arranged alphabetically by customer name.

The file was first loaded with approximately 500 alphabetized type C records, followed by 1000 alphabetized type B records, and finally about 3000 alphabetized type A records.

Records were added at the end of the file in the following manner: first, the added record type is determined (A, B, or C); then it is assigned an unused customer number that corresponds to the alphabetic sequence of the customer name according to a listing of the file. When first loaded, the contents of the file were as follows:

Record Number	Customer Number	
0001	90000	Type C (alphabetical by customer name)
0002	90020	
0003	90040	
.	.	
.	.	
.	.	
0467	60020	Type B (alphabetical by customer name)
0468	60040	
0479	60060	
.	.	
.	.	
.	.	
1592	10000	Type A (alphabetical by customer name)
1593	10013	
1594	10026	
1595	10039	

S9019105-0

The file originally contained 4,725 records. Space was allowed for 6,000 records. Now, 18 months later, the file contains 5,638 records.

An analysis of the file indicates the following:

- The file is being expanded at the rate of about 12 percent per year and should probably be planned for about 6,600 records to meet one year's requirements.

- Eight percent of customer numbers 10000 through 50000 are used, and 5 percent of the other numbers are used.

- Synonym records should be kept as close as possible to the expected location.

- The best file design solution might be more than one file and more than one type of file organization.

- If all the customer numbers will be in one file, an algorithm must take into account the necessity of loading type C customers at the front of the file, followed by types B and A.

- The ratio of A to B to C types is about 6 to 2 to 1.

A trial algorithm might try to accomplish the following mapping:

Customer Number	Type	File Record Number
90000 through 99999	C	0001 through 0733 (1/9 x 6600 = 733)
60000 through 79999	B	0734 through 2200 (2/9 x 6600 = 1467)
10000 through 49999	A	2201 through 6600 (6/9 x 6600 = 4400)

In order to accomplish the mapping, the algorithm must:

- Convert customer numbers 90000 through 99999 into a set of relative record numbers from 1 through 733.
- Convert customer numbers 60000 through 79999 into a set of relative record numbers from 734 through 2200.
- Convert customer numbers 10000 through 49999 into a set of relative record numbers from 2201 through 6600.

One method of doing these conversions is as follows:

- If the customer number is greater than 89999, subtract 89999 from it, multiply the difference by 0.0733 (the ratio of 733 positions to 10000 numbers), and use the half-adjusted product as the record position.
- If the customer number is less than 50000, subtract 9999 from it, multiply the difference by 0.11 (the ratio of 4400 record positions to 40000 record numbers), add the half-adjusted product to 2200, and use the sum as the record position.
- For all other customer names (60000 to 79999), subtract 59999 from the number, multiply the difference by 0.0733 (the ratio of 1,467 record positions to 20,000 numbers), add the half-adjusted product to 733, and use the sum as the record position.

The synonym-handling technique might be the same as in "Example 1" on page A-4. You should test the synonym-handling technique by loading the file. Then the technique's effectiveness can be measured by another program that attempts to retrieve all records and counts the number of accesses necessary. The results of the second program indicate whether modifications are necessary or desirable. To further test the synonym-handling technique, run a sample program in an interactive environment to see whether response time at the display stations is acceptable.

Example 3

The following techniques are randomizing techniques. These techniques use part of the data to determine the record position. Regardless of which randomizing technique you use, document the concept and approach in each program that uses the technique.

Some master files might have altogether different uses and for that reason use different techniques. Consider a rate file in a telephone revenue accounting application; the file would have one record for every *from-to* location in the United States. A call made from number (123) 555-1234 to (456) 555-4567 would require the retrieval of a rate record from the master file that would have a key of 123555456555. How can such a number be converted to a relative record position on a direct file?

You could develop an algorithm that multiplies the numbers 123555 and 456555 and then uses the second, fourth, sixth, eighth, and tenth digits of the product as the relative record position. This technique might yield a random distribution across a file for approximately 100,000 records. Another approach would be to use an algorithm that takes the second, fourth, sixth, eighth, and tenth digits from the 12-digit key. Thus, the first algorithm might locate the rate record in relative position 20632 (123555 x 456555 = 22109653025); the second algorithm might place the same record in position 25555.

Some records, for a given billing location, would be far more active than the majority of the records. These very active records might be placed in a separate file, which may or may not be direct.

Glossary

#LIBRARY. The library, provided with the system, that contains the System Support Program Product. See *system library.*

abnormal termination. A system failure or operator action that causes a job to end unsuccessfully.

access level. The level of authority an operator has in order to use a secured file, library, folder, or folder member.

access method. The way that records in files are referred to by the system. The reference can be consecutive (records are referred to one after another in the order in which they appear in the file), or it can be random (the individual records can be referred to in any order).

accumulate. To collect. For example, to accumulate the values in a field.

accumulating. The process of totaling the values in a particular field as records are being processed.

acquire. To assign a display station or session to a program.

acquired session. A session that has been started by a System/36 program using an acquire operation, or in BASIC, using an OPEN statement.

adapter. See *communications adapter.*

address. (1) A name, label, or number that identifies a location in storage, a device in a network, or any other data source. (2) In PS/36, an 8-byte code required for sign-on and the distribution of mail.

address output file. Either a record address file or a limits file.

address switches. Switches that you set to represent the address of a work station.

addressing. (1) In data communications, the way that the sending or control station selects the station to which it is sending data. (2) A means of identifying storage locations.

addrout file. See *address output file.*

advanced program-to-program communications (APPC). Communications support that allows System/36 to communicate with other systems having the same support. APPC is the way that System/36 puts the IBM SNA LU-6.2 protocol into effect.

alarm. An audible signal at a display station or printer that is used to get the operator's attention.

alert. A record sent to another system to communicate a problem or an impending problem. On System/36, the problem management portion of the Communications and Systems Management feature used to generate and send alerts.

align. To bring into or be in line with another or with others. For example, to align numbers on the decimal point.

allocate. To assign a resource, such as a disk file or a diskette file, to perform a specific task.

alphabetic character. Any one of the letters A through Z (uppercase and lowercase). Some program products extend the alphabet to include the special characters #, $, and @. A character that is one of the 26 uppercase characters of the alphabet, or a space. Any one of the uppercase letters A through Z, or the special character $.

alphameric. Consisting of letters, numbers, and often other symbols, such as punctuation marks and mathematical symbols.

alphanumeric. See *alphameric.*

alternative index. An index that is built after a physical file is created and that provides a different order for reading or writing records in the file. Contrast with *primary index.*

alternative system console. A command display station that can be designated as the system console.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

application. (1) A particular business task, such as inventory control or accounts receivable. (2) A group of related programs that apply to a particular business area, such as the Inventory Control or the Accounts Receivable application.

application program. A program used to perform an application or part of an application.

archive. To copy a folder member onto tape or diskette.

archived member. A folder member that has been saved on disk, diskette or tape.

argument. An expression that is passed to a function or subroutine for evaluation. A parameter passed between a calling program and a subprogram or statement function.

ascending key sequence. The arrangement of data in order from the lowest value of the key field to the highest value of the key field. Contrast with *descending key sequence.*

ascending sequence. The arrangement of data in order from the lowest value to the highest value, according to the rules for comparing data. Contrast with *descending sequence.*

ASCII. See *American National Standard Code for Information Interchange (ASCII).*

assembler. A program that converts assembler language statements to machine instructions.

assembler language. A symbolic programming language in which the set of instructions includes the instructions of the machine and whose data structures correspond directly to the storage and registers of the machine.

assign/free area. An area of main storage that contains control information for all system activity and for each job that is active.

assignment. The process of giving values to variables.

asynchronous transmission. In data communications, a method of transmission in which the bits included in a character or block of characters occur during a specific time interval. However, the start of each character or block of characters can occur at any time during this interval. Contrast with *synchronous transmission.*

attribute. A characteristic. For example, an attribute for a displayed field could be blinking. A characteristic of a graph that you can change.

audit trail. Information that allows the history of things such as a customer account or item record to be traced. The more recent information can be stored in the computer.

authority. The right to communicate with or use a resource.

authorization list. A list of user identifications and access levels that is used to secure folders and folder members.

authorize. To allow a user to communicate with or use a resource.

auto report. An RPG option that simplifies the defining of formats for printed reports and that allows the previously written statements to be included in new programs.

automatic response severity level. The value that indicates whether messages should be automatically responded to by the System Support Program Product.

back up. To copy information, usually onto diskette or tape, for safekeeping.

backup copy. A copy, usually of a file, library member, or folder, that is kept in case the original is unintentionally changed or destroyed.

badge security. A System Support Program Product option that helps prevent the unauthorized use of a display station by checking the data from a magnetic stripe on a badge before allowing an operator to sign on.

base number. The part of a self-check field from which the check digit is calculated.

basic data exchange. A file format for exchanging data on diskettes between systems or devices.

basic ideographic character set. A character set defined by IBM that contains 3226 Kanji and 481 additional characters. The additional characters include Katakana, Hiragana, the alphabet (A through Z and a through z), numbers (0 through 9), Roman numerals (I through X), Greek, Cyrillic, and special symbols. Contrast with *extended ideographic character set*; see also *ideographic character set.*

batch. Pertaining to activity involving little or no operator action. Contrast with *interactive.*

batch compilation. A method of compiling programs without the continual attention of an operator.

batch processing. A processing method in which a program or programs process records with little or no operator action. Contrast with *interactive processing.*

beginning of tape. A reflective marking near the beginning of a tape reel that indicates where the system can begin recording data.

BGU/36. See *Business Graphics Utilities/36 (BGU/36).*

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Involving a choice of two conditions, such as on-off or yes-no.

binary operator. A symbol representing an operation to be performed on two data items, arrays, or expressions. The four types of binary operators are numeric, character, logical, and relational.

bit. Either of the binary digits 0 or 1. See also *byte.*

block. (1) A group of records that is recorded or processed as a unit. Same as *physical record.* (2) Ten sectors (2560 bytes) of disk storage. (3) In data communications, a group of records that is recorded, processed, or sent as a unit. A sequential group of statements (defined using line commands) that is treated as a unit. A sequential string of text (defined using cursor movement keys or line commands) that is treated as a unit.

Boolean data. A category of data items that are limited to a value of one or zero.

bps. Bits per second.

branching. Performing a statement other than the next one in sequence.

buffer. (1) A temporary storage unit, especially one that accepts information at one rate and delivers it at another rate. (2) An area of storage, temporarily reserved for performing input or output, into which data is read or from which data is written.

Business Graphics Utilities/36 (BGU/36). A program product that can be used to design, display, print, and plot graphics.

byte. The amount of storage required to represent one character; a byte is 8 bits.

C. Celsius.

C & SM. See *Communications and Systems Management (C & SM).*

cache. A fixed user area of main storage that contains recently accessed disk data.

cache page. The smallest amount of contiguous disk data that can be held in a cache.

calendar. A list or schedule of appointments, reminders, and programs.

calendar default. A default value assigned to a calendar user. The calendar defaults are the calendar itself, calendar view, and display format.

call. (1) To activate a program or procedure at its entry point. Compare with *load.* (2) In data communications, the action necessary in making a connection between two stations on a switched line.

cancel. To end a task before it is completed.

CGU. See *character generator utility (CGU).*

chain. (1) A group of logically linked records. (2) In SNA, a group of logically linked records that are transferred over a communications line.

chained file. An input, output, or update disk file from which records can be read randomly.

change authority. The right to create, add, change, and remove files, libraries, and folders.

change bar. A character used to indicate any document line that is changed.

change management. The part of the Communications and Systems Management feature that allows a host system operator to send (via DSX) programming changes and new programs to System/36, and to start procedures on System/36.

channel. A path along which data passes.

character. A letter, digit, or other symbol.

character expression. A character constant, a simple character variable, a scalar reference to a character array, a character-valued function reference, or a sequence of the above separated by the concatenation operator (&) and parentheses.

character generator utility (CGU). A program that is used to create, maintain, and display ideographic characters.

character key. A keyboard key that allows the user to enter the character shown on the key. Compare with *command key* and *function key.*

character string. A sequence of consecutive characters. A sequence of characters that form a COBOL word, a literal, a PICTURE character string, or a comment entry.

character variable. The name of a character data item whose value is assigned and/or changed while the program is running.

characters per inch (CPI). The number of characters printed within an inch horizontally across a page.

chart. A display screen, printed page, or plotted page that contains multiple graphs.

chart utility. The part of BGU/36 that allows you to design, display, print, and plot charts and maintain chart members.

check. (1) An error condition. (2) To look for a condition.

child. Pertaining to a secured resource, either a file or library, that uses the user list of a parent resource. A child resource can have only one parent resource. Contrast with *parent.*

close. To end the processing of a file.

COBOL (common business-oriented language). A high-level programming language, similar to English, that is used primarily for commercial data processing.

code. (1) Instructions for the computer. (2) To write instructions for the computer. Same as *program.* (3) A representation of a condition, such as an error code.

column. (1) A character position within a print line or on a display. The positions are numbered from 1, by 1, starting at the leftmost character position and extending to the rightmost position. (2) A group of data that is aligned vertically (usually with tabs) within a list or table. A list of values in a report. Each field in the report is a single column.

command. A request to the system to perform an operation or a procedure.

command display. A display that allows an operator to display and send messages, and use control commands and procedure commands to start and control jobs. Contrast with *standby display.* See also *console display* and *subconsole display.*

command display station. A display station from which an operator can start and control jobs. A command display station can become an alternative system console, can be designated as a subconsole, and can also be used as a data display station. See also *alternative system console*, *data display station*, and *subconsole.*

command file. In the MSRJE utility, a disk file, procedure member, or source member that can contain MSRJE utility control statements and records to be transmitted to the host system. Contrast with *data file.*

command key. A keyboard key that is used to request specific programmed actions. Compare with *character key* and *function key.*

command processor. The part of the System Support Program Product that processes control commands and that passes procedure commands and operation control language statements to the initiator.

comment. Words or statements in a program or on a display that serve as documentation rather than as instructions, choices, or prompts. A note in the Identification Division or Procedure Division of a COBOL source program. A comment is ignored by the compiler. As an IBM extension, comments may be included at any point in a COBOL source program.

comment line. A source program line that is not translated by the compiler. The comment line can be used to document the program. A special form of the comment line can be used to cause page ejection before the comment line is printed.

communications. See *data communications.*

communications adapter. A hardware feature that enables a computer or device to become a part of a data communications network.

Communications and Systems Management (C & SM). A feature of the System Support Program Product that contains the remote management support (also referred to as DHCF), the change management support (referred to as DSNX), and the problem management support (referred to as alerts).

communications file. A file that describes an advanced program-to-program communications (APPC) subsystem session between a System/36 program and a remote device, another program, or another system.

communications file definition. The format in the communications file that contains the APPC subsystem session description.

communications line. The line over which data communications takes place; for example, a telephone line.

communications security. A System Support Program Product option that allows the identity of a remote location to be verified before that location can run programs on your system.

compile. To translate a program written in a high-level programming language into a machine language program.

compiler. A program that compiles.

complex condition. A condition in which one or more logical operators (AND, OR or NOT) act upon one or more conditions. Complex conditions include negated simple conditions, combined conditions, and negated combined conditions. See *conditional expression* and *simple condition.*

compress. (1) To move files, libraries, or folders together on disk to create one continuous area of unused space. (2) To replace repetitive characters in a file or folder with control characters so that the file or folder takes up less space when saved on diskette.

compression. In data communications, a technique for removing strings of duplicate characters and for removing trailing blanks before transmitting data.

computer graphics. The use of a computer to produce pictorial representations of relationships, such as charts, and two- or three-dimensional images, by means of dots, lines, curves, and so forth.

concatenate. (1) To link together. (2) To join two character strings.

concept. An idea generalized from particular instances.

condense. To move library members together in a library to create one continuous area of unused space in the library.

condition. An expression in a program or procedure that can be evaluated to a value of either true or false when the program or procedure is running. An expression in a program for which a truth value can be determined at run time. Conditions include the simple conditions (relation condition, class condition, condition-name condition, switch-status condition, sign condition) and the complex conditions (negated simple conditions, combined conditions, negated combined conditions).

conditional expression. A logical statement that describes the relationship (such as greater than or equal) of two items. A simple condition or a complex condition specified in an IF, a PERFORM, or a SEARCH statement. See *complex condition* and *simple condition.*

conditioning. The use of indicators to control when calculations or output operations are done.

configuration. The group of machines, devices, and programs that make up a data processing system. See also *system configuration.*

consecutive processing. The processing of records in the order in which they exist in a file. Same as *sequential processing.* See also *random processing.*

console display. A display that can be requested only at the system console. From a console display an operator can display, send, and reply to messages and use all control commands.

constant. A data item with a value that does not change. Contrast with *variable.*

constant field. A field that is defined by a display format to contain a value that does not change.

contiguous. Being in actual contact.

continuation line. A line of a source statement into which characters are entered when the source statement cannot be contained on the previous line or lines.

control block. A storage area used by a program to hold control information.

control command. A command used by an operator to control the system or a work station. A control command does not run a procedure and cannot be used in a procedure.

control field. A field that identifies a record's relationship to other records (such as a part number in an inventory record). In RPG, control fields are compared from record to record to determine when certain operations are to be performed. In sort, control fields determine the order of records in the sorted file.

control panel. A panel that contains lights and keys used to observe and operate the status of the operations within the system.

control storage. Storage in the computer that contains the programs used to control input and output operations and the use of main storage. Contrast with *main storage.*

control storage initial program load. The loading of control storage programs from disk or diskette to control storage.

control storage processor. The hardware that performs control storage instructions to handle data transfer and main storage, and input/output assignments.

controlled cancel. The system action that ends the job step being run and saves any new data already created. The job that is running can continue with the next job step.

controller. Circuitry or a device used to coordinate and control the operation of one or more devices.

counter. A register or storage location used to accumulate the number of occurrences of an event.

CPI. See *characters per inch (CPI).*

creation date. The program date at the time a file is created. See also *program date*, *session date*, and *system date.*

current library. The first library searched for any required members. The current library can be specified during sign-on or while running programs and procedures.

current record. The record that is currently available to the program.

cursor. A movable symbol on a display, used to indicate to the operator where to type the next character.

cylinder. All disk or diskette tracks that can be read or written without moving the disk drive or diskette drive read/write mechanism.

data communications. The transmission of data between computers and/or remote devices (usually over a long distance).

data definition. Information that describes the contents and characteristics of a field, format (record), or file. A data definition can include such things as field names, lengths, and data types. See also *field definition*, *file definition*, and *format definition.*

data dictionary. A folder that contains field, format, and file definitions.

data display station. A display station from which an operator can only enter data. A data display station is acquired and controlled by a program. Contrast with *command display station.*

data entry facility. A function of Query/36 that allows a user to add, change, and mark records to be deleted in a file. The file must be linked to a file definition created with IDDU.

data file. In the MSRJE utility, a disk file, procedure member, or source member that can contain only records to be transmitted to the host system. Contrast with *command file.*

data file utility (DFU). The part of the Utilities Program Product that is used to create, maintain, display, and print disk files.

data link. The equipment and rules (protocols) used for sending and receiving data.

data management. See *disk data management.*

data stream. All information (data and control information) transmitted over a data link.

data type. A category that identifies the mathematical qualities and internal representation of data.

DDM. See *Distributed Data Management (DDM).*

deactivate. To make ineffective. For example, to deactivate security.

debug. To detect, locate, and remove errors from a program.

decimal. (1) Pertaining to a system of numbers to the base ten; decimal digits range from 0 through 9. (2) A proper fraction in which the denominator is a power of 10.

default. See *default value.*

default printer. A printer that accepts all the printed output from a display station that is assigned to it.

default value. A value stored in the system that is used when no other value is specified.

define-the-file (DTF). A control block containing information that is passed between data management routines and users of the data management routines.

delete. To remove. For example, to delete a file.

delete character. A character that identifies a record to be removed from a file.

delete-capable file. A file from which records can be logically removed without compressing the file.

delimiter. A character or sequence of characters that marks the beginning or end of a unit of data. A character or a sequence of consecutive characters that marks the end of a unit of data and is not a part of that unit of data.

descending key sequence. The arrangement of data in order from the highest value of the key field to the lowest value of the key field. Contrast with *ascending key sequence.*

descending sequence. The arrangement of data in order from the highest value to the lowest value, according to the rules for comparing data. Contrast with *ascending sequence.*

detail record. A record that contains the daily activities or transactions of a business. For example, the items on a customer order are typically stored in detail records. Contrast with *header record.*

detail report. A report that contains all the information produced by a query. Contrast with *summary report.*

Development Support Utility (DSU). A program product that can be used to create, edit, remove, view, or print procedure members and source members.

device passthru. See *Display Station Pass-Through (DSPT).*

DFU. See *data file utility (DFU).*

diagnostic. Pertaining to the detection and isolation of an error.

diagnostic diskette. A diskette that contains tests to check that the system is operating properly.

diagnostic program. A computer program that recognizes, locates, and explains either a fault in equipment or a mistake in a computer program.

digit. Any of the numerals from 0 through 9.

direct file. A disk file in which records are referenced by the relative record number. Contrast with *indexed file* and *sequential file.*

directory. See *network resource directory (NRD).* A file containing such information as a name, address, and telephone number for each user of PS/36. Indirect users and individuals or organizations that do not use PS/36 may also be listed in the directory.

disable. In interactive communications, to end a subsystem and free the area of main storage used by that subsystem. Contrast with *enable.*

disk. A storage device made of one or more flat, circular plates with magnetic surfaces on which information can be stored.

disk data management. The System Support Program Product support that processes a request to read or write data.

disk drive. The mechanism used to read and write information on disk.

disk file. A set of related records on disk that is treated as a unit. See also *record file* and *stream file.*

diskette. A thin, flexible magnetic plate that is permanently sealed in a protective cover. It can be used to store information copied from the disk or to exchange information with other computers.

diskette drive. The mechanism used to read and write information on diskettes.

diskette magazine drive. A diskette drive that holds up to two magazines plus three individual diskettes.

diskette 1. A diskette that contains information on only one side.

diskette 2D. A diskette that contains information on both sides, and with two times the amount of information stored in the same space as a diskette 1. Therefore, a diskette 2D holds approximately four times the amount of information as a diskette 1.

display. (1) A visual presentation of information on a display screen. (2) To show information on the display screen.

display format. Data that defines (or describes) a display.

display layout sheet. A form used to plan the location of data on the display.

display screen. The part of the display station on which information is displayed.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator can see the information sent to or the information received from the system.

Display Station Pass-Through (DSPT). A communications feature that allows a user to sign on to one System/36 from another System/36 and access that remote system's resources.

DisplayWrite/36 (DW/36). A program product that creates, revises, views, and prints documents that are produced in an office environment.

disposition. In file processing, the process of specifying whether a file is new, old, or shared, and how the file is to be shared.

Distributed Data Management (DDM). A feature of the System Support Program Product that allows an application program to work on files that reside on a remote system.

Distributed Systems Executive (DSX). A program product available for IBM host systems (System/370, 43XX, and 30XX) that allows the host system to get, send, and remove files, programs, formats, and procedures in a network of computers.

Distributed Systems Node Executive (DSNX). Another name for the change management support offered by the Communications and Systems Management feature. This support processes changes sent by a DSX host system.

distribution list. A list of users to receive a particular piece of mail. This list can be within a group.

distribution request. A request to send a document or documents to an individual or to a distribution list.

document. One or more lines of text that can be named and stored as a member in a folder.

document description. Data that describes the characteristics of a document. The description can include document type, subject, author, and date created.

document folder. A folder that is used to store documents.

document format. The selected arrangement of text for a specific document.

document ID. An eight-character name that PS/36 automatically assigns to each document it sends or receives.

dot matrix. (1) In computer graphics, a two-dimensional pattern of dots used for constructing a display image. (2) In word processing, a pattern of dots used to form characters.

down load. To transmit a font over a communications line to a 6670 printer.

DSNX. See *Distributed Systems Node Executive (DSNX).*

DSU. See *Development Support Utility (DSU).*

DSX. See *Distributed Systems Executive (DSX).*

DTF. See *define-the-file (DTF).*

dump. (1) To copy the contents of all or part of storage, usually to an output device. (2) Data that has been dumped.

dump file. A file that contains the data areas used by a program that failed.

DW/36. See *DisplayWrite/36 (DW/36).*

dynamic access. An access mode in which records can be read from or written to a file in a nonsequential order (see *random access*) and read from a file in a sequential order (see *sequential access*) during the scope of the same OPEN statement.

E-format. Floating-point format, consisting of a number in scientific notation.

EBCDIC. See *extended binary-coded decimal interchange code (EBCDIC).*

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC set.

edit. (1) To modify the form or format of data; for example, to insert or remove characters such as for dates or decimal points. (2) To check the accuracy of information that has been entered, and to indicate if an error is found. (3) To make changes to a document by adding, changing, or removing text.

edit code. A number or letter indicating that editing should be done according to a defined pattern.

EDIT display. (DSU) The display used to make changes to a member by adding, changing, or removing statements.

Edit display. (DW/36) The display used to make changes to a document by adding, changing, or removing text.

embedded blanks. Blanks that are surrounded by any other characters.

emulation. Imitation; for example, the imitation of a computer or device.

enable. In interactive communications, to load and start a subsystem. Contrast with *disable.*

end of extent. The end of the area on a disk or diskette reserved for a file.

end of tape. A reflective marking near the end of a tape reel that indicates where the system must stop recording data.

enhance. To make greater; to improve.

enter. To type in information from a keyboard and press the Enter key in order to send the information to the computer.

enter/update mode. The mode that is used to enter new statements into a source or procedure member, or to change statements that already exist in a source or procedure member.

entry. Any descriptive set of consecutive clauses ended by a period and written in the Identification, Environment, or Data Division of a COBOL source program.

evoke. To start a program or procedure so that it can communicate with your program.

exchange file. A file format for exchanging data on diskette or tape between systems or devices that support that medium. See also *basic data exchange.*

expiration date. The date after which a diskette file is no longer protected from being automatically erased by the system.

exponent. A number, indicating to which power another number (the base) is to be raised.

exponent (of an E-format number). An integer constant specifying the power of ten by which the base (mantissa) of the decimal floating-point number is to be multiplied.

exponentiation. The operation in which a value is raised to a power.

expression. A representation of a value. For example, variables and constants appearing alone or in combination with operators.

extendable disk file. A file that the system can increase in size whenever more space is needed.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

extended character file. An area on disk that contains the extended ideographic character set.

extended ideographic character set. An ideographic character set, residing in auxiliary storage, that contains 3483 IBM-supplied ideographic characters and up to 4370 user-defined ideographic characters. Contrast with *basic ideographic character set*; see also *ideographic character set.*

extent. A continuous space on disk or diskette that is occupied by, or reserved for, a particular file, library, or folder.

external indicators. Indicators that can be set by another program before a program is run or changed while a program is running. The external indicators are U1 through U8.

extract. To obtain. For example, to extract information from a file.

factor. A field name, constant, literal, subroutine name, label, display name, or file name used in an operation.

feature. A programming or hardware option, usually available at an extra cost. For example, Communications is a feature of the System Support Program Product.

field. One or more characters of related information (such as a name or an amount).

field definition. Information that describes the characteristics of data in a field. A field definition is contained in a data dictionary.

file. A set of related records treated as a unit.

file definition. (1) In RPG, file description and input specifications that describe the records and fields in a file. (2) In IDDU, information that describes the contents and characteristics of a file. A file definition is contained in a data dictionary.

file name. The name used by a program to identify a file. See also *label.*

file organization. The permanent file structure established at the time a file is created.

first-level message. A message that is issued immediately when an error occurs. See also *second-level message.*

fixed-format menu. A menu that is formatted as two 12-item columns. Compare with *free-format menu.*

fixed-point constant. A numeric constant consisting of an optional sign followed by one or more digits and a decimal point.

fixed-point format. The form used to express a fixed-point constant.

floating-point constant. (1) A numeric constant consisting of an optional sign followed by one or more digits and a decimal point, which may be at the end. (2) A numeric constant with an optional sign followed by the letter D or E, followed by a one- to three-digit integer constant. For example, 3E-02, which is 3 times 10 to the -2 power or 0.03.

floating-point format. The form used to express a floating-point constant.

folder. A named area on disk that contains documents, profiles, mail, or data definitions. Compare with *library.*

folder directory. An area, in a folder, that contains information about each member in the folder; for example, the member name and the location.

folder member. A named collection of statements in a folder. A document is an example of a folder member.

font. An assortment of characters of a given size and style; for example, 10 point Courier.

footer. Text that appears at the bottom of every page of a document. For example, a page number could be a footer. Compare with *header.*

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, files, or documents. (2) To arrange such things as characters, fields, and lines. (3) In BASIC, a representation of the correct form of a command or statement. (4) In IDDU, a group of related fields, such as a record, in a file.

format definition. Information that describes the contents and characteristics of data within a group of related fields, such as a record in a file. A format definition is contained in a data dictionary.

format member. A load member that contains display formats generated from S and D specifications in a program. A member that contains all the graph information defined for a graph (for example: graph type, headings, and axis).

formatted diskette. A diskette on which control information has been written but which may or may not contain any data.

formatted message. A two-line display in which the first line (format line) provides information about the message, and the second line (message text line) contains the message itself.

FORTRAN (formula translation). A high-level programming language used primarily for scientific, engineering, and mathematical applications.

free-form format. The SEU display format designed for entering and updating statements, such as OCL statements and utility control statements, that do not have a constant format. The format line and language prompt designed for entering and updating statements, such as OCL and utility control statements, that do not have a constant format.

free-format menu. A menu for which the programmer defines the format of lines 3 through 20. Contrast with *fixed-format menu.*

full-procedural file. A disk file that can be processed both randomly and sequentially.

full-screen editor. A program that allows you to edit an entire screen of data or text at a time.

function key. A keyboard key that requests an action but does not display or print a character. The cursor movement and Help keys are examples of function keys. Compare with *command key* and *character key.*

function subprogram. A user-written subprogram defined by FORTRAN statements, the first of which is a FUNCTION statement. See also *statement function* and *subroutine.*

Gaiji. A character in the extended ideographic character set.

general user. A person, such as an office principal (manager or professional), secretary, or clerk, who is enrolled in and who can sign on to and use PS/36 directly. Contrast with *indirect user.*

generation. For some remote systems, the translation of configuration information into machine language.

graph. Displayed, printed, or plotted output that compares two or more sets of variable data. The types of graphs are bar, line, pie, surface, and text.

graph utility. The part of BGU/36 that allows you to design, display, print, and plot graphs, produce a graph object file, and maintain format and data members.

graphic. (1) A picture. (2) See *computer graphics.*

group. A list of names that are known together by a single name.

group resource record. A record in the resource security file that secures a group of files and/or libraries.

hard copy. A printed copy.

hardware. The equipment, as opposed to the programming, of a system.

header. Text that appears at the top of the printed pages of a document. For example, the subject of the document could be a header. Compare with *footer.*

header label. A special set of records on a diskette or tape that describes the contents of the diskette or tape.

header record. A record that contains information, such as customer name and customer address, that is common to following detail records. Contrast with *detail record.*

Help key. A function key that, when pressed, displays online information or some part of the system help support.

help text. The part of the system help support that offers additional information about displays and messages.

hex. See *hexadecimal.*

hexadecimal. Pertaining to a system of numbers to the base sixteen; hexadecimal digits range from 0 (zero) through 9 (nine) and A (ten) through F (fifteen).

history file. A file that contains a log of system actions and operator responses.

I/O. See *input/output (I/O).*

ID. Identification.

IDDU. See *interactive data definition utility (IDDU).*

identifier. (1) A sequence of bits or characters that identifies a program, device, or system to another program, device, or system. (2) In COBOL, a data name that is unique or is made unique by the correct combination of qualifiers, subscripts, or indexes. (3) In PS/36, a name that identifies the type of member in a group. The identifier can be a calendar, a user ID, or another group.

ideographic. Pertaining to 2-byte characters consisting of pictograms, symbolic characters, and other types of symbols.

ideographic character set. The combination of the basic and extended ideographic character sets; see also *basic ideographic character set* and *extended ideographic character set.*

ideographic session. A display station operating session during which ideographic data is used for system communication with the operator.

ideographic sort utility. A program that sorts ideographic data.

ideographic SSP. A version of the System Support Program Product that includes formats for help displays in both Katakana 1-byte and 2-byte ideographic characters. Compare with *Kanji-preferred SSP.*

ideographic support. The hardware and programming elements that allow processing of ideographic data.

IF expressions. Expressions within a procedure that are used to test for a condition.

include. To add statements from one library member to another library member.

independent work station. A work station that can operate independently of a host system, but which can also communicate with a host system. An example of an independent work station is a Displaywriter.

independent work station user. A person who uses the Electronic Document Distribution licensed program to communicate with PS/36.

index. (1) A table containing the key value and location of each record in an indexed file. (2) A computer storage position or register, the contents of which identify a particular element in a set of elements.

index key. The field within a record that identifies that record in an indexed file.

indexed file. A file in which the key and the position of each record are recorded in a separate portion of the file called the index. Contrast with *direct file* and *sequential file.*

indicator. An internal switch that communicates a condition between parts of a program or procedure.

indirect user. A person enrolled as a PS/36 user who is authorized to receive and print mail but has no mail log. Contrast with *general user.*

informational message. A message that provides information to the operator, but does not require a response.

initial program load (IPL). The process of loading the system programs and preparing the system to run jobs.

initialize. To prepare for use. For example, to initialize a diskette.

initiator. The part of the System Support Program Product that reads and processes operation control language statements from the system input device.

input. Data to be processed.

input stream. The sequence of operation control statements and data given to the system from an input device.

input/output (I/O). Pertaining to either input or output, or both.

inquiry. (1) A request for information in storage. (2) A request that puts a display station into inquiry mode. (3) In data communications, a request for information from another system.

inquiry mode. A mode during which the job currently running from a display station is interrupted so that other work can be done. The operator puts the display station in inquiry mode by pressing the Attn key.

inquiry program. (1) A program that allows an operator to get information from a disk file. (2) A program that runs while the system is in inquiry mode.

instruction. A statement that specifies an operation to be performed by the computer and the locations in storage of all data involved in that operation.

integer. A positive or negative whole number; that is, an optional sign followed by a number that does not contain a decimal point. A numeric data item or literal that does not include any character positions to the right of the decimal point. When the term integer appears in formats, integer must be an unsigned numeric literal and must be nonzero unless the rules for that format explicitly state otherwise.

interactive. Pertaining to activity involving requests and replies as, for example, between an operator and a program or between two programs. Contrast with *batch.*

Interactive Communications Feature (SSP-ICF). A feature of the System Support Program Product that allows a program to interactively communicate with another program or system.

interactive data definition utility (IDDU). The part of the System Support Program Product used to define the characteristics of data and the contents of files.

interactive processing. A processing method in which each operator action causes a response from the program or the system. Contrast with *batch processing*.

interrupt. (1) To temporarily stop a process. (2) In data communications, to take an action at a receiving station that causes the sending station to end a transmission.

invite. To ask for input data from either a display station or an SSP-ICF session.

IPL. See *initial program load (IPL).*

ISO. International Standards Organization.

job. (1) A unit of work to be done by a system. (2) One or more related procedures or programs grouped into a procedure.

job file. A disk file that exists until the job that uses it ends.

job queue. A list of jobs waiting to be processed by the system.

job region. The main storage space reserved by the System Support Program Product for use by a job.

job step. A unit of work represented by a single program or a procedure that contains a single program. A job consists of one or more job steps.

job stream. One or more library source members or procedure members saved on diskette or tape.

justify. To adjust text to be even with the top, bottom, left, or right margin.

K-byte. 1024 bytes.

Kanji. (1) The ideographic character set used by the Japanese to represent their native language. (2) A single character in the ideographic character set.

Kanji-preferred SSP. A version of the System Support Program Product that includes formats for help displays in Kanji 2-byte ideographic characters. Compare with *ideographic SSP.*

Katakana. A native Japanese character set that is used primarily to write foreign words phonetically.

key. One or more characters used to identify the record and establish the record's order within an indexed file. One or more characters used to identify the record and establish the record's order within an indexed file or a direct (relative) file.

keyboard. A group of numeric keys, alphabetic keys, and function keys used for entering information at a display station and into the system.

label. (1) The name in the disk or diskette volume table of contents or on a tape that identifies a file. See also *file name.* (2) The name that identifies a statement. The name that identifies a BASIC program line.

left-adjust. To place or move an entry in a field so that the leftmost character of the field is in the leftmost position. Contrast with *right-adjust.*

library. (1) A named area on disk that can contain programs and related information (not files). A library consists of different sections, called library members. Compare with *folder.* (2) The set of publications for a system.

library control sector. In a library directory, the first sector, which contains a record of the used and available space in the library.

library directory. An area, in a library, that contains information about each member in the library; for example, the member name and the location.

library member. A named collection of records or statements in a library. The types of library members are *load member*, *procedure member*, *source member*, and *subroutine member.*

library member subtype. A specific classification of a library member type. For example, a source member can be identified as a COBOL source member or a DFU source member.

library name. A user-defined word that names a library.

licensed application program. A set of licensed programs used to perform a particular data processing task, such as a distribution management application or a construction management application.

licensed program. An IBM-written program that performs functions related to processing user data.

lightness. The characteristic that allows colors to be ranked on a scale from light to dark.

limits file. A file that contains upper and lower values of the record keys that can be used to read from an indexed file.

lines per inch (LPI). The number of characters printed within an inch vertically down the page.

link-editing. To combine, by the overlay linkage editor, a number of load members and/or subroutine members into one program.

linkage. The coding that passes control and parameters between two routines.

linkage editor. See *overlay linkage editor.*

literal. A symbol or a quantity in a source program that is itself data, rather than a reference to data.

load. (1) To move data or programs into storage. (2) To place a diskette into a diskette drive or a diskette magazine into a diskette magazine drive. (3) To insert paper into a printer. (4) To mount a tape or insert a tape cartridge into a tape drive.

load member. A library member that contains information in machine language, a form that the system can use directly. Contrast with *source member.*

load module. A program in a form that can be loaded into main storage and run. The load module is the output of the overlay linkage editor.

local. Pertaining to a device, file, or system that is accessed directly from your system, without the use of a communications line. Contrast with *remote.*

local data area. A 512-byte area on disk that can be used to pass information between jobs and job steps during a session. A separate local data area exists for each command display station.

log. (1) To record; for example, to log all messages on the system printer. (2) See *mail log.*

logical expression. An expression consisting of logical operators and/or relational operators that can be evaluated to a value of either true or false.

logical operator. A word or symbol that defines the logical connection between conditions or that makes opposite a condition. A reserved word that defines the logical connection between conditions or negates a condition: OR (logical connective-either or both), AND (logical connective-both), and NOT (logical negation). A set of operators used in logical expressions. The operators are: .NOT. (logical negation), .AND. (logical conjunction), and .OR. (logical union).

loop. A sequence of instructions that is performed repeatedly until an ending condition is reached.

LPI. See *lines per inch (LPI).*

M-byte. See *megabyte.*

machine instruction. An instruction of the machine language that can be performed by the computer.

machine language. A language that can be used directly by a computer without intermediate processing.

macro. See *macroinstruction.*

macroinstruction. A single instruction that represents a set of instructions.

magazine. A container that holds up to 10 diskettes.

magnetic tape. See *tape.*

magnetic tape unit. A device for reading or writing data from or on magnetic tape.

mail. Any correspondence (online or hard copy) that is sent between users.

mail folder. A folder used to store documents sent and received as mail.

mail list. A selected part of an entire mail log.

mail log. A record of all the mail sent or received by a user.

main storage. The part of the processing unit where programs are run. Contrast with *control storage.*

main storage processor. Hardware that performs the machine language instructions in main storage.

mainline module. A mainline routine after it has been compiled.

mainline routine. The first subroutine encountered when link-editing.

mandatory entry field. A field in which an operator must enter at least one character.

mandatory fill field. A field for which an operator must enter nothing or must fill in completely.

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters.

master configuration record. Information, stored on disk, that describes system devices, programming, and characteristics.

master file. A collection of permanent information, such as a customer address file.

master security officer. A person who is designated to control all of the security tasks that are provided with the System Support Program Product. A master security officer can, for example, deactivate password, badge, or resource security, or add, change, or remove security information about any system operator. Contrast with *security officer.*

match fields. When processing more than one file with RPG, fields that are compared to determine whether operations should be done.

matrix. An array arranged in rows and columns.

megabyte. One million bytes.

member. See *library member.*

memo slip. A short message that can be sent with a document to give instructions to the addressee.

menu. A displayed list of items from which an operator can make a selection.

menu security. A System Support Program Product option that restricts an operator to selecting items from a particular menu.

merge. To combine two or more ordered files into one similarly ordered file. To perform a data/text merge. To break down the groups contained within another group to their individual members.

message. (1) Information sent to one or more users or display stations from a program or another user. A message can be either displayed or printed. (2) An indication of the condition of the system sent by the system. (3) For IMS/IRSS, a unit of data sent over the communications line.

message identification. A field in the display or printout of a message that directs the user to the description of the message in a message guide or a reference manual. This field consists of up to four alphabetic characters, followed by a dash, followed by the message identification code.

message identification code (MIC). A four-digit number that identifies a record in a message member. This number can be part of the message identification.

message member. A library member that defines the text of each message and its associated message identification code.

MIC. See *message identification code (MIC).*

mm. Millimeter.

mode. A method of operation. For an example, see *enter/update mode.*

module. (1) One part of a program, which usually performs a specific task (such as disk input/output). (2) See *load module.* (3) See *object module.*

modulus 10/modulus 11 checking. Formulas used to calculate the check digit for a self-check field.

monitor. Programming or hardware that observes, supervises, controls, or verifies the operation of a system.

MRT procedure. See *multiple requester terminal (MRT) procedure.*

MRT program. See *multiple requester terminal (MRT) program.*

MSRJE. See *Multiple Session Remote Job Entry (MSRJE).*

multiple. More than one.

multiple requester terminal (MRT) procedure. A procedure that calls a multiple requester terminal program.

multiple requester terminal (MRT) program. A program that can process requests from more than one display station or SSP-ICF session at the same time using a single copy of the program. Contrast with *single requester terminal (SRT) program.*

Multiple Session Remote Job Entry (MSRJE). A feature of the System Support Program Product that allows one or more remote job entry sessions to operate on a host system (such as a System/370, or a 30XX or 43XX processor) at the same time.

multiprogramming. The processing of two or more programs at the same time.

multivolume file. A diskette file that occupies more than one diskette.

name. A word that defines a COBOL operand. A name is composed of not more than 30 characters.

national characters. The characters #, $, and @.

NEP. See *never-ending program (NEP).*

nest. To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one loop (the nested loop) within another loop (the nesting loop); to nest one subroutine (the nested subroutine) within another subroutine (the nesting subroutine).

nested procedure. A procedure that is called by another procedure. See also *procedure level.*

network. A collection of data processing products connected by communications lines for information exchange between stations.

network resource directory (NRD). An area on disk that lists the files on remote systems that can be accessed using Distributed Data Management (DDM).

never-ending program (NEP). A long-running program that does not share system resources, except for shared files and the spool file.

noncontiguous. Not being in actual contact.

nonlabeled tape. A tape that has no labels. Tape marks are used to indicate the end of the volume and the end of each data file.

nonrequesting terminal program. A program that is not associated with a requesting display station.

nonsequenced display. A display that is not part of a sequence. Contrast with *primary display sequence*, *secondary display*, and *sequenced display.*

nonstandard labeled tape. A tape that has labels but does not follow the IBM standard labeling conventions.

nonswappable storage. The storage containing programs or data that must remain in storage.

nucleus. That portion of main storage that is used by the System Support Program Product.

null character. The character hex 00, used to represent the absence of a displayed or printed character.

null character string. Two consecutive single quotation marks that specify a character constant of no characters.

numeric. Pertaining to any of the digits 0 through 9.

object module. A set of instructions in machine language. The object module is produced by a compiler from a subroutine or source program and can be input to the overlay linkage editor.

object program. A set of instructions in machine-runnable form. The object program is produced by a compiler from a source program.

OCL. See *operation control language (OCL).*

office products. A group of IBM-supplied programs that work together to help an office operate more efficiently. The office products are DisplayWrite/36 (DW/36), Personal Services/36 (PS/36), and Query/36. The interactive data definition utility (IDDU) can be used to define files used by DW/36 and Query/36.

OFFICE/36. The group of office products: DisplayWrite/36 (DW/36), Personal Services/36 (PS/36), and Query/36.

offline. Neither controlled directly by, nor communicating with, the computer, or both. Contrast with *online.*

online. Being controlled directly by, or directly communicating with, the computer, or both. Contrast with *offline.*

online information. Information, read on the display screen, that explains displays, messages, and programs. For some programs, the online information is similar to a printed manual and may contain a table of contents, guide information, practice exercises, help text, a glossary, and an index.

open. To prepare a file for processing.

operand. A quantity of data that is operated on, or the address in a computer instruction of data to be operated on. The object of a verb or an operator; that is, an operand is the data or equipment governed or directed by a verb or operator.

operation. A defined action, such as adding or comparing, performed on one or more data items.

operation control language (OCL). A language used to identify a job and its processing requirements to the System Support Program Product.

operator. (1) A person who operates a device. (2) A symbol that represents an operation to be done.

option. An item (usually numbered) in a list that a user selects to perform a task.

output. The result of processing data.

output stream. Messages and other output data, displayed on output devices by an operating system or a processing program.

overlay. (1) To write over (and therefore destroy) an existing file. (2) A program segment that is loaded into main storage and replaces all or part of a previously loaded program segment.

overlay linkage editor. The part of the System Support Program Product that combines object programs to produce code that can be run and allows the user to determine overlays for programs.

overlay region. A continuous area of main storage in which segments can be loaded independently of other regions.

override. (1) A parameter or value that replaces a previous parameter or value. (2) To replace a parameter or value.

override user ID. A user identification that is used to sign on to the system if the user identification file is destroyed.

owner authority. The right to create, add, change, delete, and rename files, libraries, and folders that he/she owns. The right to add, change, read (view), and delete items in files, libraries, and folders that he/she owns.

packed decimal format. A format in which each byte (except the rightmost byte) within a field represents two numeric digits. The rightmost byte contains one digit and the sign. For example, the decimal value 123 is represented as 0001 0010 0011 1111. Contrast with *zoned decimal format.*

packed key. An index key in packed decimal format.

pad. To fill unused positions in a field with dummy data, usually zeros or blanks.

page. A 2048-byte segment of main storage.

parameter. A value supplied to a procedure or program that either is used as input or controls the actions of the procedure or program. A variable or a literal that is used to pass data values between calling and called programs.

parent. Pertaining to a secured resource, either a file or library, whose user list is shared with one or more other files or libraries. Contrast with *child.*

password. A string of characters that, when entered along with a user ID, allows an operator to sign on to the system.

PC Support/36. A group of programs that can be used to transfer data from a System/36 to an IBM personal computer, to use disk storage on System/36 as IBM personal computer disk storage, and to use printers attached to System/36 as a IBM personal computer printer.

PCE. See *procedure control expression (PCE).*

pending. Waiting, as in an operation is pending.

personal document. A document that may be handled only by its owner or by someone who knows the owners personal document password specified.

personal document password. A string of characters that must be entered to handle a personal document.

Personal Services/36 (PS/36). A program product that can be used to send and receive mail, schedule appointments on calendars, maintain directories of names and addresses, and work with groups of users or calendars.

physical file. A file that contains data records.

physical record. (1) A group of records that is recorded or processed as a unit. Same as *block.* (2) A unit of data that is moved into or out of the computer.

position. The location of a character in a series, as in a record, a displayed message, or a computer printout.

positional parameter. A parameter that must appear in a specified location, relative to other positional parameters.

primary display sequence. The first set of displays coded in a WSU source program.

primary file. The main file from which a program reads records.

primary index. The index that is built when a file is created. Contrast with *alternative index.*

print image. A character set that corresponds to the characters on a print band.

print intercept routine. The spooling routine that causes printer output to be placed in a spool file rather than being printed.

printout. Information from the computer that is produced by a printer.

priority. The relative ranking of items. For example, a job with high priority will be run before one with regular or low priority.

problem determination. The process of identifying why the system is not working. Often this process identifies programs, equipment, data communications facilities, or user errors as the source of the problem.

problem management. The part of the Communications and Systems Management feature that allows System/36 to generate and send alerts to a host system using APPC.

procedure. A set of related operation control language statements (and, possibly, utility control statements and procedure control expressions) that cause a specific program or set of programs to be performed. One or more successive paragraphs or sections within the Procedure Division, which directs the computer to perform some action or series of actions.

procedure command. A command that runs a procedure.

procedure control expression (PCE). A set of statements and expressions that control how a procedure runs.

Procedure Division. One of the four main component parts of a COBOL program. The Procedure Division contains instructions for solving a problem. The Procedure Division may contain imperative statements, conditional statements, paragraphs, procedures and sections.

procedure level. The relative position of a procedure within nested procedures. For example, if procedure A calls procedure B, and procedure B in turn calls procedure C, then procedure C is a third-level procedure.

procedure member. A library member that contains the statements (such as operation control language statements) necessary to perform a program or set of programs.

procedure start request. A message from the remote system asking an SSP-ICF subsystem to start a System/36 procedure.

processing unit. The part of the system unit that performs instructions and contains main storage.

profile. Data that describes the significant features of a user, program, device, or remote location.

program. (1) A sequence of instructions for a computer. See *source program* and *load module.* (2) To write a sequence of instructions for a computer. Same as *code.*

program date. The date associated with a program (job step). See also *creation date*, *session date*, and *system date.*

program generation. The compilation of a WSU program.

program product. A licensed program for which a fee is charged.

program temporary fix (PTF). A temporary solution to or bypass of a defect in a current release of a licensed program.

Programming Request for Price Quotation (PRPQ). A program created especially for a particular group of customers or an application. Documentation for the program is provided only to those customers who order the PRPQ.

prompt. A displayed request for information or operator action.

PRPQ. See *Programming Request for Price Quotation (PRPQ).*

PS/36. See *Personal Services/36 (PS/36).*

PTF. See *program temporary fix (PTF).*

query. A request for information from a file based on specific conditions; for example, a request for a list of all customers in a customer master file whose balance is greater than $1000.

Query/36. A program product that produces files and reports based on those files. The files must be linked to file definitions created with IDDU.

queue. A line or list formed by items waiting to be processed.

random access. An access method in which records can be read from, written to, or removed from a file in any order. An access mode in which records can be read from, written to, or removed from a file in any order.

random by key. A processing method for chained files in which record keys identify records to be processed.

random by relative record number. A processing method for chained files in which relative record numbers identify the records to be processed.

random processing. The processing of records in an order other than the order that they exist in a file. See also *consecutive processing* and *sequential processing.*

real number. A number, containing a decimal point, stored in fixed-point or floating-point format.

record. A collection of fields that is treated as a unit.

record address file. An input file that indicates to a program which records are to be read from a disk file, and the order in which these records are to be read from the disk file.

record file. A file on disk in which the data is read and written in records. Contrast with *stream file.*

record ID code. See *record identification code (record ID code).*

record identification code (record ID code). One or more characters that identify a record as belonging to a particular format of a disk file.

record identifying indicator. An indicator that identifies the record just read.

record type. The classification of records in a file.

recovery procedure. (1) An action performed by the operator when an error message appears on the display screen. Usually, this action permits the program to continue or permits the operator to run the next job. (2) The method of returning the system to the point where a major system error occurred and running the recent critical jobs again.

region size. The amount of main storage available for a program to run. See also *job region* and *step region.*

register. A storage area, in a computer, usually intended for some special reason, capable of storing a specified amount of data such as a bit or an address.

relational expression. A logical statement that describes the relationship (such as greater than or equal) of two arithmetic expressions or data items.

relational operator. The reserved words or symbols used to express a relation condition or a relational expression. A reserved word, a relation character, a group of consecutive reserved words, or a group of consecutive reserved words and relation characters used to express a relation condition. Any of the set of operators that express an arithmetic condition that can be either true or false. The operators are: .GT., .GE., .LT., .LE., .EQ., and .NE.. They are defined as greater than, greater than or equal to, less than, less than or equal to, and not equal to, respectively.

relative addressing. A means of addressing instructions and data areas by designating their location in relation to the location counter or to some symbol.

relative file. Same as *direct file.*

relative record number. A number that specifies the location of a record in relation to the beginning of the file.

remote. Pertaining to a device, file, or system that is accessed by your system through a communications line. Contrast with *local.*

remote job entry (RJE). Sending job instructions and possibly data to a remote system requesting it to run a job.

Remote Operation/Support Facility (ROSF). An implementation that allows an operator at a remote support group to use a remote display station (and an optional remote printer) to provide operational and technical assistance.

remotely started session. A session started by an incoming procedure start request from the remote system. Contrast with *acquired session.*

reorganize. To move folder members together at the front of the folder to reduce as much as possible the number of folder extents.

reserved fields. Special fields provided and maintained by WSU that contain such current information as relative record numbers, date and time, and error codes.

reset. To return a device or circuit to a clear state.

resident file. A file that exists on disk until it is specifically deleted or changed to a scratch file.

resource. Any part of the system required by a job or task, including main storage, input and output devices, the processing unit, and files, libraries, and folders.

resource security. A System Support Program Product option that restricts the use of information in files, libraries, folders, and folder members to specified users.

resource security file. A security file that contains information that restricts access to files, libraries, and folders.

restore. Return to an original value or image. For example, to restore a library from diskette.

return code. In data communications, a value generated by the system or subsystem that is returned to a program to indicate the results of an operation issued by that program.

right-adjust. To place or move an entry in a field so that the rightmost character of the field is in the rightmost position. Contrast with *left-adjust.*

ROSF. See *Remote Operation/Support Facility (ROSF).*

routine. A set of statements in a program that causes the system to perform an operation or a series of related operations.

RPG. A programming language specifically designed for writing application programs that meet common business data processing requirements.

run. To cause a program, utility, or other machine function to be performed.

scratch file. A file, usually used as a work file, that exists until the program that uses it ends.

screen design aid (SDA). The part of the Utilities Program Product that helps the user design, create, and maintain displays and menus. Additionally, SDA can generate specifications for RPG and WSU work station programs.

SDA. See *screen design aid (SDA).*

SEAR. South East Asia Region.

search word. Data used to find a match in a table or array.

second-level message. A message that supplies additional information about an error condition when the Help key is pressed for a first-level message. See also *first-level message.*

secondary display sequence. The set of displays that follows the primary display sequence in a WSU source program.

secondary file. Any input file other than the primary file.

sector. (1) An area on a disk track or a diskette track reserved to record information. (2) The smallest amount of information that can be written to or read from a disk or diskette during a single read or write operation.

security. The protection of data, system operations, and devices from accidental or intentional ruin, damage, or exposure. See also *system security.*

security officer. A person who is designated to control many of the system security tasks that are provided with the System Support Program Product. A security officer can, for example, add, change, or remove security information about system console operators, subconsole operators, and display station operators. A security officer cannot, however, deactivate password, badge, or resource security. Contrast with *master security officer.*

segment. A part of a program that can be run without the entire program being in main storage.

self-check field. A field, such as an account number, consisting of a base number and a check digit.

separator page. A printed page used to show the end of output for one job and the start of output for another job.

sequenced display. A display within a sequence. See *nonsequenced display.*

sequential access. An access mode in which records are read from, written to, or removed from a file based on the logical order of the records in the file.

sequential by key. A method of indexed file processing in which records are read or written in the order of the record keys.

sequential file. A file in which records occur in the order in which they were entered. Contrast with *direct file* and *indexed file.*

sequential processing. The processing of records in the order in which they exist in a file. Same as *consecutive processing.* See also *random processing.*

session. (1) The logical connection by which a System/36 program or device can communicate with a program or device at a remote location. (2) The length of time that starts when an operator signs on the system and ends when the operator signs off the system.

session date. The date associated with a session. See also *creation date, program date*, and *system date.*

session library. The library specified, or assigned as a default, when signing on or while running a program.

SEU. See *source entry utility (SEU).*

severity code. A code that indicates how serious a compiling error or an operating error is.

severity level. See *automatic response severity level.*

shared folder facility. A function of PC Support/36 that allows multiple System/36 and personal computer users concurrent access to a folder from a personal computer.

shift-in control character. A character (hex 0F) that indicates the end of a string of ideographic characters. Contrast with *shift-out control character*

shift-out control character. A character (hex 0E) that indicates the start of a string of ideographic characters. Contrast with *shift-in control character*

sign off. To end a session at a display station.

sign on. (Verb) To begin a session at a display station.

sign-on. (Noun) The action an operator uses at a display station in order to begin working at the display station.

simple condition. Any single condition chosen from the set: relation condition; class condition; condition-name condition; switch-status condition; sign condition. See *complex condition* and *conditional expression.*

single requester terminal (SRT) program. A program that can process requests from only one display station or SSP-ICF session from each copy of the program. Contrast with *multiple requester terminal (MRT) program.*

SMF. See *system measurement facility (SMF).*

SNA. See *systems network architecture (SNA).*

SNA Upline Facility (SNUF). The SSP-ICF subsystem that allows System/36 to communicate with CICS/VS and IMS/VS application programs on a host system. Also, using this subsystem, DHCF communicates with HCF and DSNX communicates with DSX.

SNUF. See *SNA Upline Facility (SNUF).*

sort utility. The part of the System Support Program Product used to arrange records (or their relative record numbers) in a sequence determined by data contained in one or more fields within the records.

source. A system, a program within a system, or a device that makes a request to a target. Contrast with *target.*

source entry utility (SEU). The part of the Utilities Program Product used by the operator to enter and update source and procedure members.

source member. A library member that contains information in the form in which it was entered, such as RPG specifications. Contrast with *load member.*

source program. A set of instructions that are written in a programming language and that must be translated to machine language before the program can be run.

source statement. A statement written in a programming language.

special character. A character other than an alphabetic or numeric character. For example; *, +, and % are special characters. A character that is neither numeric nor alphabetic. Special characters in COBOL include the space ( ), and the period (.), as well as the following: + - * / = $, ") (; < >.

specification sheets. Forms on which a program is coded and described.

split key. A key, for an indexed file, defined from more than one field within each record.

spool file segment. A part of the spool file that can hold a print file, or part of a print file.

spool intercept buffer. An area of main storage containing printer data that is being written in the spool file.

spool writer. The part of the System Support Program Product that prints output that has been saved in the spool file.

spooling. The part of the System Support Program Product that saves output on disk for later printing.

SRT program. See *single requester terminal (SRT) program.*

SSP. See *System Support Program Product (SSP).*

SSP-ICF. See *Interactive Communications Feature (SSP-ICF).*

standard label tape. A tape that follows the IBM standard labeling conventions.

standby display. A display that allows an operator to enter data only. When a standby display appears, the display station can be acquired by a program. Contrast with *command display.*

statement. An instruction in a program or procedure. A syntactically valid combination of words and symbols, beginning with a verb, that is written in the Procedure Division.

statement function. A user-written function that is defined and referred to within the same program unit. The user-written function is defined in a statement function definition statement. See also *function subprogram* and *subroutine.*

status. A condition. For example, the status of a printer, a job, or a communications line.

step region. The main storage space reserved by the System Support Program Product for use by a program.

storage index. A table in main storage that contains the address of the lowest key on each track in the file index.

stream file. A file on disk in which data is read and written in consecutive fields. Contrast with *record file.*

subconsole. A display station that controls a printer or printers.

subconsole display. A display that can be requested only from a command display that appears on a subconsole. From a subconsole display an operator can display and send messages, and enter all control commands except those that can be entered only at the system console. See also *console display.*

subdirectory. (S/36) A part of a folder that contains the names, descriptions, member types, and security information for other directories (subdirectories and folder members. Subdirectories are part of the shared folder facility. See also *shared folder facility.*

subdirectory level. Subdirectories within folders or within other subdirectories in a folder are assigned a level based on the number of subdirectories in the path. If a subdirectory is the first one in a folder, it is assigned level 1. If the subdirectory is within another subdirectory, it is assigned level 2, 3, etc. based on how many subdirectories precede it.

subordinate. Occupying a lower class or rank.

subroutine. A group of instructions that can be called by another program or subroutine. A subprogram consisting of FORTRAN statements, the first of which is a SUBROUTINE statement. It optionally returns one or more parameters to the calling program unit. See also *function subprogram* and *statement function.*

subroutine member. A library member that contains information that must be combined with one or more members before being run by the system. A library member that contains a BASIC program in the form in which it appears within the computer.

subsystem. The part of communications that handles the requirements of the remote system, isolating most system-dependent considerations from the application program.

subtype. See *library member subtype.*

summary report. A report that contains only the summary information produced by a query. Contrast with *detail report.*

swapping. The process of temporarily removing an active job from main storage, saving it on disk, and processing another job in the area of main storage formerly occupied by the first job.

synchronous. Occurring in a regular or predictable sequence.

synchronous transmission. In data communications, a method of transmission in which the sending and receiving of characters is controlled by timing signals. Contrast with *asynchronous transmission.*

system. The computer and its associated devices and programs.

system configuration. A process that specifies the machines, devices, and programs that form a particular data processing system.

system console. A display station from which an operator can keep track of and control system operation.

system date. The date assigned by the system operator during the initial program load procedure. See also *creation date*, *program date*, and *session date.*

system dump. A dump of all active programs (and their associated data) recorded after an error stops the system. Contrast with *task dump.*

system help support. The part of the System Support Program Product that uses menus, prompts, and descriptive text to aid an operator.

system library. The library, provided with the system, that contains the System Support Program Product and is named #LIBRARY.

system list device. The device that receives output for most System Support Program Product utility programs and service aids.

system log device. The device or devices designated by the LOG OCL statement to record messages and OCL statements.

system measurement facility (SMF). System Support Program Product routines that, in conjunction with control storage routines, observe system and device activity, observe SSP work area usage, and record this data in a disk file.

system printer. The printer that is used for any printed output that is not specifically directed to another printer.

system program. An IBM-supplied program that is installed on the system. The System Support Program Product (SSP) is an example.

system security. A system function that restricts the use of files, libraries, folders, folder members, and display stations to certain users.

system service display station. A display station that can use all the procedures, programs, and commands needed to service the system.

system services control point (SSCP). A focal point within an SNA network for managing the configuration, coordinating network operator and problem determination requests, and providing directory support and other session services for network users.

System Support Program Product (SSP). A group of licensed programs that manage the running of other programs and the operation of associated devices, such as the display station and printer. The SSP also contains utility programs that perform common tasks, such as copying information from diskette to disk.

system unit. The part of the system that contains the processing unit, the control panel, the disk drive and the disk, and either a diskette drive or a diskette magazine drive.

SYSTEM-CONSOLE. A COBOL function name associated with the operator's keyboard/display.

systems network architecture (SNA). A set of rules for controlling the transfer of information in a data communications network.

TACLA. See *twin asynchronous communications line attachment.*

tape. A thin, flexible magnetic strip on which data can be stored. It can be used to store information copied from the disk.

tape cartridge. A case containing a reel of magnetic tape arranged for insertion into a tape drive.

tape drive. A mechanism used to read and write information on magnetic tapes.

tape mark. A mark on the tape that indicates the beginning or end of a file or tape.

tape reel. A round device on which magnetic tape is wound.

tape volume. A single reel of magnetic tape.

target. A system, a program within a system, or a device that interprets, rejects or satisfies, and replies to requests received from a source. Contrast with *source.*

task. A unit of work (such as a user program) for the main storage processor.

task dump. A dump of a program that failed (and its associated data). Contrast with *system dump.*

task work area. An area on disk containing control information and work areas related to a specific task.

terminal. In data communications, a device, usually equipped with a keyboard and a display device, capable of sending and receiving information over a communications line.

terminator. The part of the System Support Program Product that performs the action necessary to end a job or program.

text. The displayed or printed information of a document.

trace. To record data that provides a history of events that occur in the system. To create a list of the number of lines performed when a BASIC program is run.

track. A circular path on the surface of a disk or diskette on which information is magnetically recorded and from which recorded information is read.

transaction. (1) An item of business. The handling of customer orders and customer billing are examples of transactions. (2) In interactive communications, the communication between the application program and a specific item (usually another application program) at the remote system.

transaction file. A file containing data, such as customer orders, that is usually used only with a master file. A direct file containing control records and data records for each work session.

transient. Pertaining to a System Support Program Product program that does not reside in main storage or to a temporary storage area for such a program.

translation table. A table that provides replacement characters for characters that cannot be printed by the 3262 printer.

transparent text mode. A mode that allows BSC to send and receive messages containing any of the 256 character combinations in hexadecimal, including transmission control characters.

truncate. To shorten a field or statement to a specified length.

twin asynchronous communications line attachment (TACLA). In data communications, a feature that allows a second communications line using asynchronous protocol to be connected to the 5362 System Unit with the single line communications adapter/attachment (SLCA).

unique. The only one.

update authority. The right to add, change, or remove items in a file, library, or folder.

update file. A disk file from which a program reads a record, updates fields in the record, and writes the record back into the location it came from.

user area. The parts of main storage and disk that are available to the user.

user ID. See *user identification (user ID).*

user identification (user ID). A string of characters that identifies a user to the system.

user identification file (user ID file). A file containing information about which operators can use certain system functions, which menu is displayed when an operator signs on to the system, and which library is assigned to an operator when the operator signs on to the system.

user identification record (user ID record). A record in the directory that gives a user's name, address, and telephone number.

user list. A list, containing the user identification and access levels, of all operators who are allowed to use a specified file or library.

user profile. A profile in the user identification file that contains information about someone who is allowed to sign on to the system.

user program status indicator (UPSI) switch. One of a set of eight switches that can be set by and passed between application programs and procedures.

utilities. See *utility program.*

Utilities Program Product. A program product that contains the data file utility (DFU), the source entry utility (SEU), the work station utility (WSU), and the screen design aid (SDA).

utility program. (1) A program provided to perform a task that is required by many of the programs using the system; for example, a program that copies information from diskette to disk. (2) A program of the System Support Program Product that performs a common task.

U1-U8 indicators. See *external indicators.*

valid. (1) Allowed. (2) True, in conforming to an appropriate standard or authority.

variable. A name used to represent a data item whose value can change while the program is running. Contrast with *constant.*

verify. To confirm the correctness of something.

volume label. An area on a standard label tape used to identify the tape volume and its owner. This area is the first 80 bytes and contains VOL1 in the first four positions.

volume table of contents (VTOC). An area on a disk or diskette that describes the location, size, and other characteristics of each file, library, and folder on the disk or diskette.

VTOC. See *volume table of contents (VTOC).*

WACK. See *wait-before-transmitting-acknowledgment character (WACK).*

wait-before-transmitting-acknowledgment character (WACK). In BSC, the transmission control character indicating that the station is temporarily not ready to receive data.

work file. A file that is used for temporary storage of data being processed.

work station. A device that lets people transmit information to or receive information from a computer; for example, a display station or printer.

work station data management. The part of the System Support Program Product that enables a program to present data on a display screen by providing a string of data fields and a format name.

Work Station Expansion feature. A feature that provides six ports for the attachment of work stations to the 5360 and 5362 System Unit.

work station utility (WSU). The part of the Utilities Program Product that helps you to write programs for data entry, editing, and inquiry.

write-enable ring. A device that is installed in a tape reel to permit writing on a tape. If a tape is mounted on a tape drive without the ring in position, writing to the tape cannot occur; the tape is protected.

write-protect plug. A device on a tape cartridge that controls writing on the tape.

WSU. See *work station utility (WSU).*

WSU-generated procedure. The procedure that WSU creates to load and run a WSU program for the first operator who calls the procedure.

X.21. In data communications, a specification of the CCITT that defines the connection of data terminal equipment to an X.21 (public data) network.

X.21 feature. The feature that allows System/36 to be connected to an X.21 network.

X.21 short hold mode. An option specified during system configuration that allows a circuit switched line to be disconnected when the line is not active.

X.25. In data communications, a specification of the CCITT that defines the interface to an X.25 (packet switching) network.

X.25 feature. The feature that allows System/36 to be connected to an X.25 network.

zoned decimal format. A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the rightmost byte; bits 0 through 3 of all other bytes contain 1s (hex F). For example, in zoned decimal format, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Contrast with *packed decimal format.*

zoned decimal item. A numeric data item that is represented internally in zoned decimal format.

zoned field. A field that contains data in the zoned decimal format.

1024-byte format. A format for diskette 2D diskettes with 1024 bytes per sector and 8 sectors per track.

1255 Magnetic Character Reader. A device that reads documents printed with magnetic ink characters.

128-byte format. A format for diskette 1 diskettes with 128 bytes per sector and 26 sectors per track.

256-byte format. A format for diskette 2D diskettes with 256 bytes per sector and 26 sectors per track.

512-byte format. A format for diskette 1 diskettes with 512 bytes per sector and 8 sectors per track.

Index

IBM System/36
Concepts and Programmer's Guide

SC21-9019-5

READER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your IBM-approved remarketer. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

□ If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

□ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): Comment(s):

Please contact your IBM representative or your IBM-approved remarketer to request additional publications.

Name ____________________________

Company or
Organization ____________________________

Address ____________________________

____
City State Zip Code

Phone No. ____________________________
Area Code

No postage necessary if mailed in the U.S.A.

Cut Along Line

Fold and tape. **Please do not staple.**

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS / PERMIT NO. 40 / ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Information Development
Department 245
Rochester, Minnesota, U.S.A. 55901

Fold and tape. **Please do not staple.**

Cut Along Line

IBM®

International Business Machines Corporation

Concepts and Programmer's Guide

File Number
S36-30

Order Number
SC21-9019-5

Printed in U.S.A.

SC21-9019-05