IBM System/3
Disk Concepts and Planning Guide

GC21-7571-0

IBM System/3
Disk Concepts and Planning Guide

Preface

This manual discusses the disk concepts and planning information you must know to design computer applications for the IBM System/3 Model 6 and Model 10 Disk Systems. The book is intended for programmers who design applications for their company.

The parts of this manual are intended to be read in a specific sequence. For information on this sequence, see *How to Use This Publication* which follows.

The reader should be familiar with either the *IBM System/3 Disk System Introduction,* GC21-7510, or the *IBM System/3 Model 6 Introduction,* GA21-9122, depending on the system he has.

After completing this manual, the reader should be able to use the various reference manuals to write basic programs. For additional information on processing disk files using RPG II, see the *IBM System/3 RPG II Disk File Processing Programmer's Guide,* GC21-7566.

First Edition (September 1971)

This new publication obsoletes GC21-7503-0 (*IBM System/3 Disk System Concepts and Programming Programmer's Guide*). Those topics in GC21-7503 that are not replaced by this new publication are now covered in GC21-7566 (*IBM System/3 RPG II Disk File Processing Programmer's Guide*). This publication also replaces some of the topics previously discussed in GC21-7511-0 (*IBM System/3 Disk System RPG II and System Additional Topics Programmer's Guide*). Topics in this publication apply to the IBM System/3 Model 10 Disk System and the IBM System/3 Model 6.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, refer to the latest IBM System/3 Newsletter, Order Number GN20-2228, for the editions that are applicable and current.

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

A Reader's Comment Form is at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901.

Contents

PART I. BASIC DISK CONCEPTS 1

CHAPTER 1. INTRODUCTION 3

CHAPTER 2. SEQUENTIAL FILES 5
Creating a Sequential File 5
Processing a Sequential File 6
Maintaining a Sequential File 6
Adding Records 6
Tagging Records for Deletion 6
Updating Records 7
Reorganizing a File 7

CHAPTER 3. INDEXED FILES 9
Creating an Indexed File 9
Processing an Indexed File 10
Sequential Processing 10
Random Processing 11
Maintaining an Indexed File 12
Adding Records 12
Tagging Records for Deletion 13
Updating Records 13
Reorganizing a File 13

CHAPTER 4. DIRECT FILES 15
Relative Record Number 16
Deriving the Relative Record Number 16
Synonym Records 19
Creating a Direct File 21
Creating a Direct File without Synonyms 21
Creating a Direct File with Synonyms 21
Processing a Direct File 22
Consecutive Processing 22
Random Processing by Relative Record Number . . . 22
Random Processing by ADDROUT File 23
Maintaining a Direct File 24
Adding Records 24
Tagging Records for Deletion 24
Updating Records 24

CHAPTER 5. RECORD ADDRESS FILES 25
Files Containing Relative Record Numbers (ADDROUT FILES) 25
Creating an ADDROUT File 25
Processing by an ADDROUT File 26
Files Containing Record Key Limits 27
Creating a File with Record Key Limits 27
Processing Sequentially within Limits 27

PART II. ADVANCED DISK CONCEPTS 29

CHAPTER 6. INQUIRY PROGRAM 31
Requesting Inquiry in an Interrupt Environment 31
Functions of the Request Key 32
Classifying Programs for Inquiry 33
Inquiry in an Interrupt Environment 34
Planning Inquiry Programs 35

CHAPTER 7. DUAL PROGRAMMING FEATURE . . . 37
Advantages of Running Programs in a DPF Environment . . 38
Main Storage 38
Processing Time 38
Input/Output Devices 39
Considerations for Operating under DPF 39
Considerations When Running System/3 Programs in a DPF Environment 42
Inquiry 42
Disk Sort, Alternate Track Assignment, and Disk Initialization 42
Executing RPG II Object Programs in a DPF Environment . 43
Loading Programs in a DPF Environment 43

PART III. FILE PLANNING 45

CHAPTER 8. CHOOSING A FILE ORGANIZATION . . 47
Use of the File 47
Volatility of the File 50
Activity of the File 51
Size of the File 51
Multivolume Files 51
Sorting a File 52

CHAPTER 9. PLANNING DISK FILES 55
Designing a Record 55
Determining Field Size 55
Providing for a Delete Code 57
Providing Extra Space 57
Naming Fields 58
Documenting Record Layout 58
Record Length 58
Block Length 59
Shared Input/Output Area 60
Determining Size and Location of a Disk File 60
Determining Number of Records in a File 61
Calculating Record Space 61
Calculating Index Space 62
Deciding Where to Locate the File 64
Data File Security 67

CHAPTER 10. STORING PROGRAMS AND PROCEDURES ON DISK 69
Advantages of Storing Programs and Procedures on Disk . . 69
Increasing System Efficiency 69
Storing Programs and Their Data Files on Removable Disks 70
Locations of Libraries on Disk 71
Source Libraries 71
Physical Characteristics of the Source Library 72
Object Libraries 73
Physical Characteristics of the Object Library 74
Storing Programs and Procedures into Libraries 75
Library Maintenance Program 75
RPG II Control Card 76
COMPILE OCL Statement 76

APPENDIX. IBM 5445 DISK STORAGE DRIVE 77
General Description 77
File Planning 79
Calculating Record Space 79
Calculating Index Space 79

INDEX 81

How to Use This Publication

This publication has three parts and an Appendix:

- *Part I–Basic Disk Concepts.* This part discusses the basic characteristics of the IBM 5444 Disk Storage Drive and describes these basic file organizations:

  1. Sequential files
  2. Indexed files
  3. Direct files
  4. Record address files

- *Part II–Advanced Disk Concepts.* This part discusses the following topics:

  1. Inquiry
  2. Dual programming feature

- *Part III–File Planning.* This part discusses the considerations for selecting a particular file organization, how to plan the files to be created, and how to store programs and procedures on disk.

- *Appendix.* The Appendix describes the IBM 5445 Disk Storage Drive and the additional planning information necessary for using this device.

The first part of the manual is for those users who need a basic knowledge of how to use disk files. The second part is only for those users who plan to use inquiry or dual programming in their processing applications. The third part can be read after the reader thoroughly understands the basic concepts discussed in Part I. The appendix should be read if an IBM 5445 is attached to the user's system.

PART I
BASIC DISK CONCEPTS

Chapter 1. Introduction

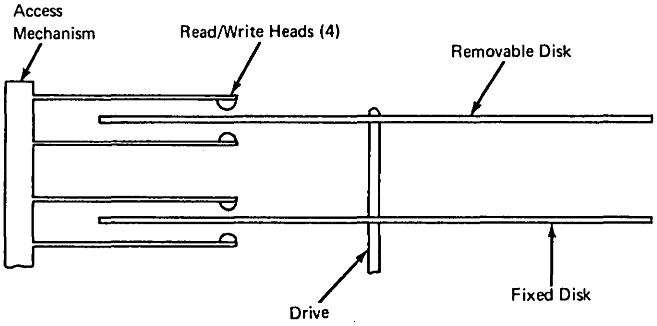
Both the IBM System/3 Model 6 and Model 10 Disk Systems use the IBM 5444 Disk Storage Drive to store information such as master, customer, and inventory files as well as programs used on the system. The major advantages of storing information on disk instead of cards are:

- Large storage capacity. A disk can hold the same amount of data as 25,600 96-column cards. Also, a disk pack is more convenient to handle than large numbers of cards.

- Faster processing rate. A card file must be processed in its entirety even if all the cards are not needed. A disk file, on the other hand, can be processed randomly; that is, only the records needed are accessed and processed.

The IBM 5444 Disk Storage Drive consists of one drive, two disks, and an access mechanism (Figure 1). The lower disk is mounted permanently on the drive. The upper disk is removable and can be replaced with other disks. Each disk, whether it is fixed or removable, is called a volume.

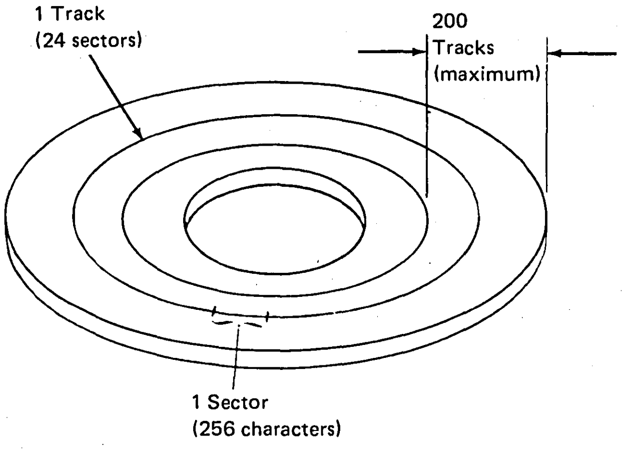
The access mechanism contains four read/write heads, one for each surface of the two disks. This mechanism moves back and forth across the disk surfaces to position the heads to read or write data. When the access mechanism is in any one position, all four heads are positioned in the same relative location on the four disk surfaces.



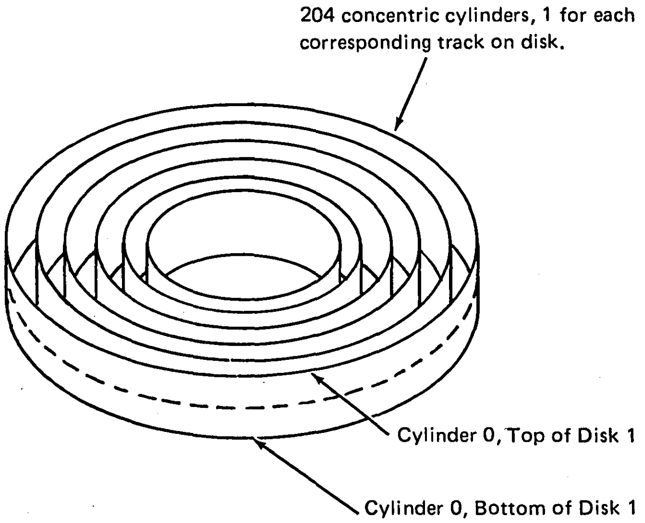
ART: 52695

Figure 1. IBM 5444 Disk Storage Drive

Each surface of each disk contains 100 or 200 tracks depending on which model of the disk storage drive you have. Tracks are divided into 24 equal parts called sectors; each sector of a track has its own unique address. Each sector can contain 256 characters of data.



Corresponding tracks from both surfaces of one disk form a cylinder. These two corresponding tracks can be accessed in a single position of the read/write heads.


204 concentric cylinders, 1 for each
corresponding track on disk.
Cylinder 0, Top of Disk 1
Cylinder 0, Bottom of Disk 1

The IBM 5444 Disk Storage Drive is available in the following configurations:

Configuration	Number of Drives	Number of Disks	Number of Cylinders	Storage Capacity
1	1	2	100/disk	2,457,600 bytes
2	1	2	200/disk	4,915,200 bytes
3	2	3	200/disk	7,372,800 bytes
4	2	4	200/disk	9,830,400 bytes

Chapter 2. Sequential Files

A disk file can be organized and processed like a card file. Such a disk file is called a sequential file. The sequence of the file can be determined by control fields, such as an employee number or a customer number, or the records may be in no particular sequence. Consecutive processing means that the records are processed one after another in the order they occur.

An example of a sequential file is an employee master file arranged in employee number order and containing information about each employee. When this file is used for processing, such as payroll checks, the records are processed consecutively. The lowest employee number is processed first and so on until the last record, the highest employee number, is processed.

A sequential file may span multiple disk volumes. (A volume refers to one disk. A multivolume file is a file that is contained on more than one disk.) A multivolume file, however, affects the processing of your file. For information on processing considerations when using multivolume sequential files, see the discussion on multivolume files in Chapter 8.

CREATING A SEQUENTIAL FILE

When you create a file, you are writing the records onto a disk for the first time. The records in a sequential file are placed on the disk consecutively; that is, they are written on the disk in the order in which they are read. Both tracks in one cylinder are filled first, then both tracks in the next cylinder, and so on until the whole file is placed on the disk.

Figure 2 shows an example of this process. In this example, each record is 128 positions long. Since each track can contain 6144 bytes of data, 48 records can be written on each track; 96 records can be written on each cylinder. The number on the tracks in the figure correspond to the number and position of each record.

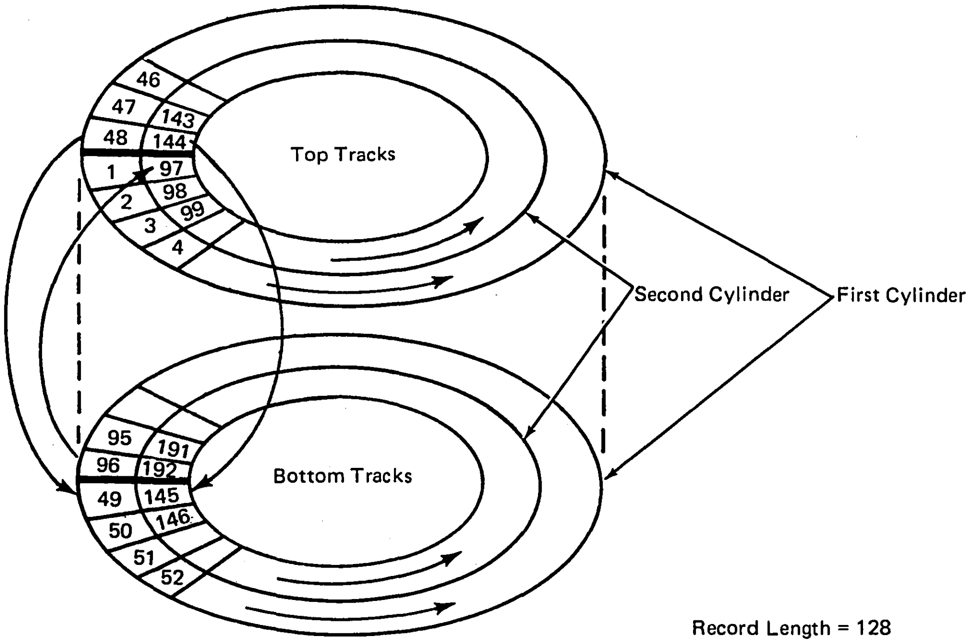


Figure 2. Writing Records on a Disk

PROCESSING A SEQUENTIAL FILE

Sequential files can be processed consecutively or randomly by relative record number. Normally the file is processed consecutively because a sequential file is usually used when all the records in the file are to be processed.

Sometimes, however, you may want to process only certain records in the file. Consecutive processing can be time consuming in this case because all the records must be processed or at least read. It would be faster to process the records randomly by a number related to the position of the records in the file. This number is called a relative record number. If your sequential file is in order by control fields and there are no missing or duplicate records, the contents of the control fields can be used as relative record numbers. For more information on this type of processing, see *Random Processing by Relative Record Number* in Chapter 4.

MAINTAINING A SEQUENTIAL FILE

Once you create a file, you must maintain it. File maintenance means performing those functions that keep a file current for daily processing needs. Four file maintenance functions apply to sequential files:

1. Adding records

2. Tagging records for deletion

3. Updating records

4. Reorganizing a file

Adding Records

Records can be added to a file after the file has been created. When records are added to a sequential file, they are written at the end of the file. Thus, the file is extended by the added records.

Sometimes, however, the new records must be merged between the records in the file. This may be necessary in order to keep the file in a particular order when the new records are not higher in sequence. In order to merge the new records in the proper sequence, you must sort the new records with the original file to create a new file containing the merged records. Another technique would be to merge the new records into the proper place in the original file during a copy to a new file.

Tagging Records for Deletion

When a record becomes inactive, you will no longer want to process it with the other records. A record cannot be physically removed from the file during regular processing; therefore, it is necessary to identify or tag the record so it can be bypassed. One way to tag such a record is to put a code, called a delete code, in a particular location in the record. When the file is processed, your program can then check for the delete code; if the code is present, the program can bypass that record.

Updating Records

When you update records in a file, you can add or change some data on the record. For example, in an inventory file you might want to add the quantity of items received to the previous quantity on hand. The record to be updated is read into storage, changed, and written back on the disk in its original location.

Reorganizing a File

When several records in a file have been tagged for deletion, you should physically remove them from the file. This will free disk space. You can remove the inactive records by copying the records to be retained onto another disk area.

Chapter 3. Indexed Files

In some data processing applications you may not want to process your file consecutively. Consecutive processing can be time consuming if you only want to process certain records in the file. It is faster to skip the records not needed in a job and process only the required ones. An indexed file allows this type of processing.

An indexed file is organized into two parts: an index and the records. The index contains an entry for each record in the file. You can go to the index, find the location of the record, go to that location, and find the record you want.

Each entry in the index describes a record in the file: if the index has 2000 entries, the file contains 2000 records. The first part of the entry contains the record's *key field.* The key field contains data that uniquely identifies the record. For example, the customer number may be the key field for a customer master record. The second part of the index entry contains the *disk address* of the record. The disk address represents the location on the disk where the record is stored. The index is arranged in ascending sequence according to the key field.

An indexed file can be a multivolume file. When processing an indexed file, however, you must consider the effect that multivolume files will have on file processing. For information on processing considerations when using multivolume indexed files, see the discussion on multivolume files in Chapter 8.

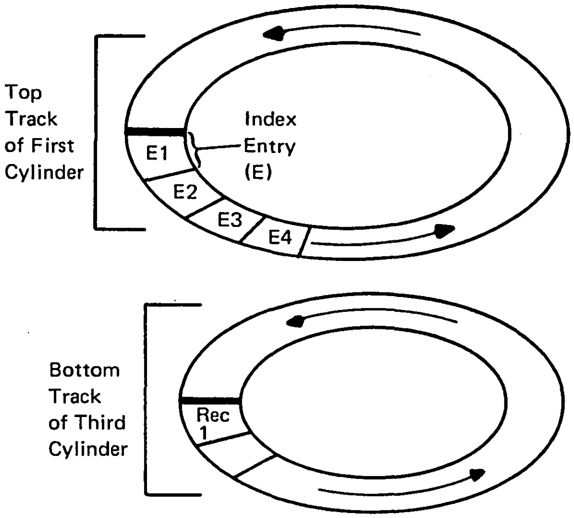
CREATING AN INDEXED FILE

When you create an indexed file, the records in the file can be in an ordered or an unordered sequence. An ordered sequence means the records are arranged in order according to some major control field used as the key field. An unordered sequence means the records are in no particular order.

An inventory file loaded according to frequency of use is an example of an unordered file. The most active items are at the beginning of the file. When the file is used to write customer orders, most of the records needed are located in a small area of the file rather than scattered throughout the entire file. This reduces the total time it takes to process the records because the access mechanism does not have to move back and forth across the whole disk to access the required records.

When an indexed file is created, the index is created as the records are written on disk. If the file is an ordered file, the index is in the correct sequence when the records are written. If the file is an unordered file, the index is automatically sorted into ascending sequence after all the records in the file have been loaded.

The index area precedes the area where records are placed. For example, suppose an index for a certain file uses five tracks. Thus the index entries would be written on the first and second cylinders and the top track of the third cylinder. Records would be written beginning in the first sector of the bottom track of the third cylinder. Both the index area and the record area must start at the beginning of a track.



PROCESSING AN INDEXED FILE

As stated previously, indexed files are not limited to consecutive processing. Indexed files can be processed several ways because the index provides several ways to find records.

Sequential Processing

When an indexed file is processed sequentially, the keys are processed one after another in ascending order. If the records are not in order on this disk, they can be processed in order by using the index. There are two ways to sequentially process an indexed file.

Sequential by Key

One way to sequentially process an indexed file is sequentially by key. Sequential by key means the records are processed in the order of the key fields. This method is used to process all records in a file, regardless of their order.

To illustrate this processing method, note the similarities and differences between File A and File B in Figure 3. Both files contain the same records, and both indexes are in order according to the key field. The difference between the two files is the order of the records. The records in File A are in order according to key field; the records in File B are unordered. All records in either file can be processed in order if you specify the processing as sequential by key.

Sequential within Limits

Another way to sequentially process an indexed file is sequentially within limits. Sequential within limits means processing records in groups.

For example, a wholesale company prepares monthly statements of each cutomer's charges. Each customer is assigned a 5-digit number; the first digit represents the region the customer is in and the remaining four digits represent the customer's number. The company's customers are divided into four regions allowing monthly statements to be sent each week to the customer's in one of the regions. Region 1 customers (10000-19999) are billed the first week of the month, region 2 customers (20000-29999) the second week, and so on. The statements, therefore, are processed sequentially within limits.

For information on processing an indexed file sequentially within limits, see Chapter 5 in this manual.

Random Processing

Indexed files can also be processed randomly. This type of processing, called random by key, permits processing of one particular record without regard to its relation to other records.

When you process a file randomly by key, you specify the key of the record you want. The key is found in the index; the disk address (adjacent to the key) is then used to locate the record so the record can be transferred to storage for processing.

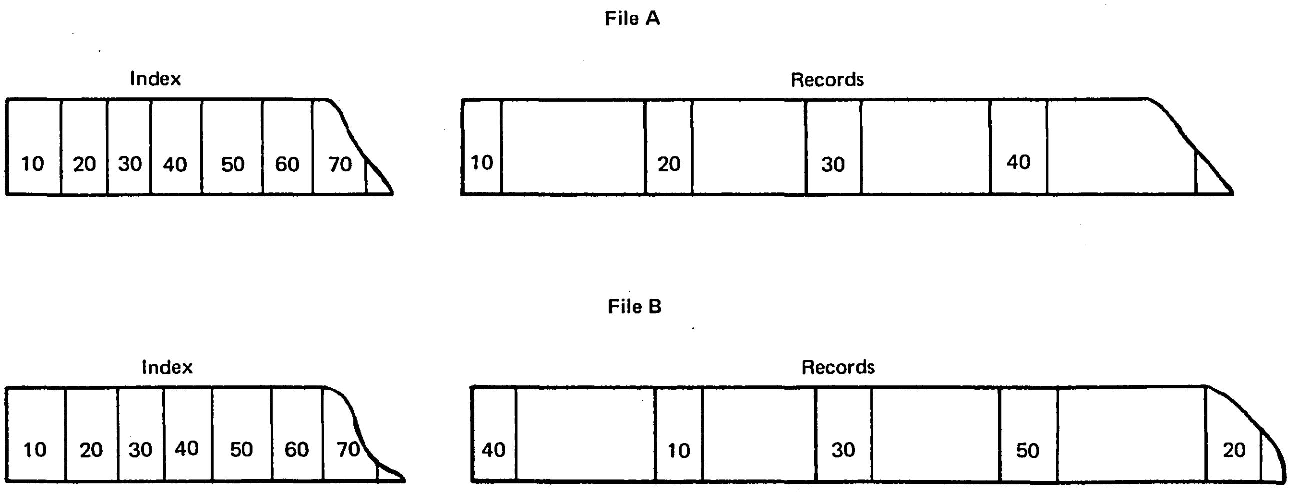


Figure 3. Example of an Ordered and an Unordered File

MAINTAINING AN INDEXED FILE

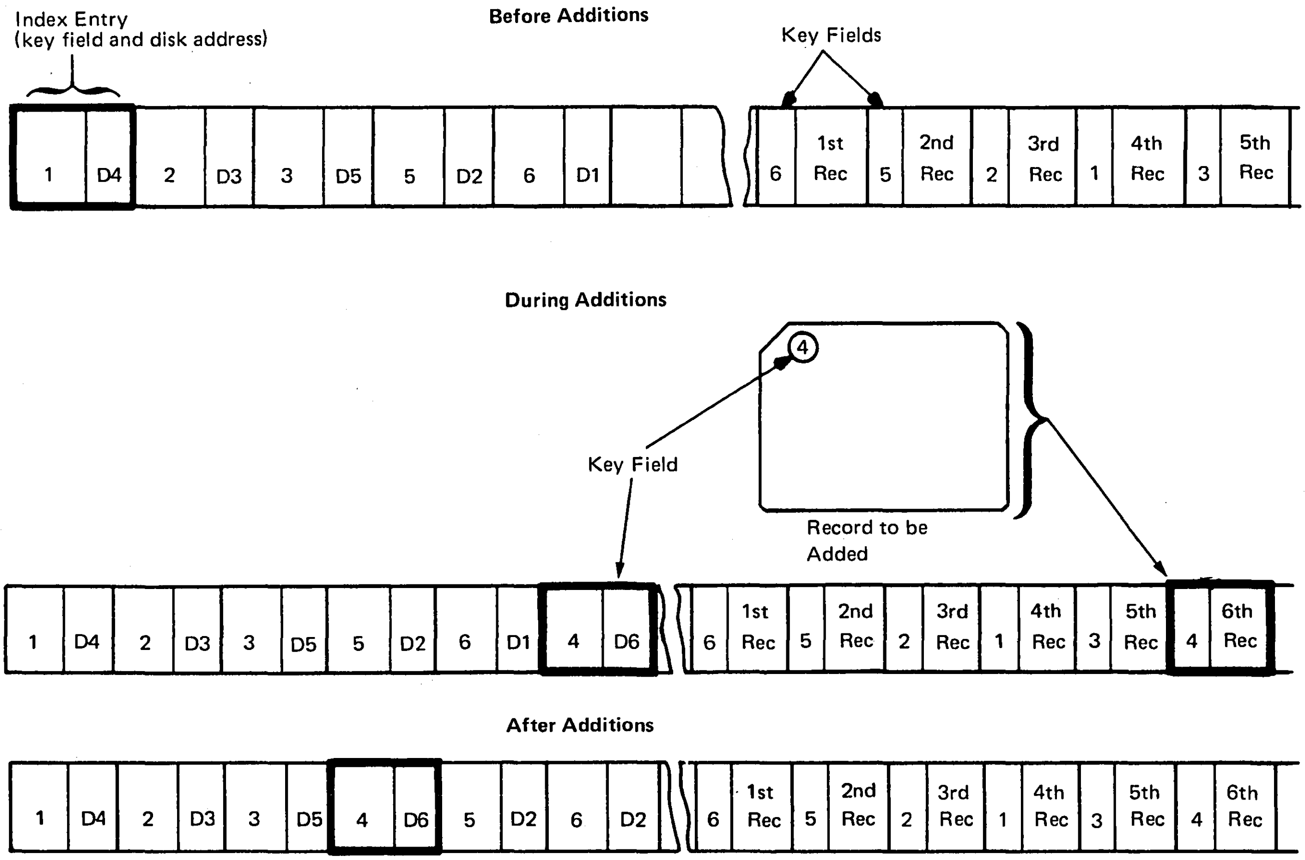
After the file is created, you can use these file maintenance functions to keep the file current for daily processing needs:

1. Adding records
2. Tagging records for deletion
3. Updating records
4. Reorganizing a file

Adding Records

When a record is added to an indexed file, it is written at the end of the records already in the file. Records can be added either sequentially by key or randomly by key. To add records *sequentially* by key, the keys of the records to be added must be higher than the keys of the records presently in the file. When records are added *randomly* by key (the records to be added need not be in any particular sequence), the system checks to ensure that the record is not a duplicate of a record already in the file; if the record is not a duplicate, it will be added to the file.

The index entry for the added record is written at the end of the current entries in the index area. After all the records are added, the index is automatically sorted so the keys of the new records are in ascending sequence.



Before Additions

During Additions

After Additions

Index Entry
(key field and disk address)
Key Fields
1 D4 2 D3 3 D5 5 D2 6 D1
6 1st Rec 5 2nd Rec 2 3rd Rec 1 4th Rec 3 5th Rec
4
Key Field
Record to be Added
1 D4 2 D3 3 D5 5 D2 6 D1 4 D6
6 1st Rec 5 2nd Rec 2 3rd Rec 1 4th Rec 3 5th Rec 4 6th Rec
1 D4 2 D3 3 D5 4 D6 5 D2 6 D2
6 1st Rec 5 2nd Rec 2 3rd Rec 1 4th Rec 3 5th Rec 4 6th Rec

Tagging Records for Deletion

Inactive records in an indexed file must be handled the same way as inactive records in a sequential file. Since the record is not removed from the file during regular processing, you must identify or tag the record so it can be bypassed. To do this, put a code called a delete code, in a particular location in the record. When the file is processed, your program can then check for the delete code; if the code is present, the program can bypass that record.

Updating Records

When you update records in a file, the record to be updated is read into storage, changed, and written back on the disk in its original location. Records in an indexed file can be updated:

1. Sequentially by key

2. Randomly by key

3. Sequentially within limits

Records are usually updated sequentially by key when you want to update all the records in the file. Each record is updated in order.

To update your file randomly by key, you specify the key you want. This key is then found in the index so the desired record can be located and moved into storage for updating.

For a discussion on updating an indexed file sequentially within limits, see Chapter 5 in this manual.

Reorganizing a File

It may be necessary at times to reorganize your indexed file in order to increase processing efficiency and free disk space. This can be done by physically merging added records in sequence with the records originally created and by physically removing records tagged for deletion.

For example, suppose an indexed file was created with the records in ascending key field order. Since that time, several records were added to the file. These records were added at the end of the file, but the index is in sequential order by key field. When the file is processed sequentially by key, the disk access arm must move back and forth between the sequenced records (those originally created) and the added records. This situation often increases processing time for a particular job. During reorganization, the added records can be placed in sequence.

As records are added to a file, the space reserved for the file becomes filled. Reorganizing is a means of freeing space since inactive records, those with a delete code, can be physically removed.

A file is reorganized by copying the old file into a new disk area. During the copy, deleted records can be removed from the file. Records previously added to the old file will be copied into the new file in sequence with the original records.

Chapter 4. Direct Files

A direct file is a file on disk in which records are assigned specific record positions. Direct file organization enables you to directly access any record in the file without examining other records or searching an index. Thus, in some processing situations, direct file organization has advantages over sequential and indexed organizations.

Figure 4 shows direct file organization. Records are assigned specific locations, independent of the order they are put into the file. All records which can be put into the file have record locations, although not all locations contain records. The specific location in the file assigned to a record is determined from a control field in the record. Records can be scattered throughout the file, depending on the distribution of the control fields. The unused record locations contain blanks.

Direct files may span multiple disk volumes. When a direct file is processed, however, all volumes containing portions of the file must be mounted on the disk drives, since every record in the file must be accessible. Therefore, multivolume direct files are limited to two volumes with a single disk drive (one fixed volume and one removable volume) and four volumes with dual disk drives (two fixed volumes and two removable volumes). For more information on processing considerations when using multivolume direct files, see the discussion on multivolume files in Chapter 8.

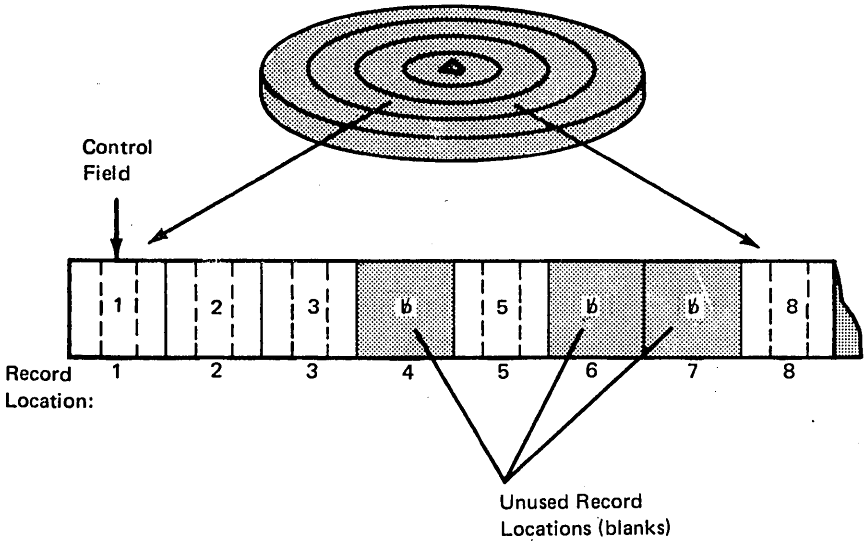


Figure 4. Direct File Organization

RELATIVE RECORD NUMBER

In a direct file, a record is written and retrieved *directly* by specifying the location of the record in relation to the beginning of the file. This relative position is called the relative record number. The relative record number is not a disk address, but is a positive, whole number that is converted by disk system management to the disk address of the record to be accessed.

Deriving the Relative Record Number

A relative record number is similar to the key of an indexed file or the control information in a sequential file: it is dependent upon a specific field (control field) in the record. The control field can either be used directly (without change) as a relative record number or it can be mathematically converted to provide an acceptable relative record number.

Direct Method

An easy way to derive relative record numbers is to have them correspond directly to the control fields in the records. Because the control information need not be converted into a relative record number, manipulation and programming are kept to a minimum. For example, in Figure 4 the record with a 1 in the control field becomes relative record number one; the record with a 5 becomes relative record number five, and so forth. This method is practical where control numbers can be assigned on a sequential basis, such as employee numbers for payroll records, student numbers in a school, and customer numbers for customer files.

Suppose a small college has an enrollment of 5,000 students. A master student file is maintained including currently enrolled students and graduates for the last two years. The master file contains approximately 7,000 records. Each student is assigned a 6-digit file number as follows:

749397

Expected year of graduation	A unique identification number from 1-9999

The identifying numbers are assigned on a sequential basis; numbers retired from the master file are available for reassignment.

A direct file with 10,000 record locations is used for the student master file, satisfying a need for fast access to each student's record. Since the identifying numbers range between 1 and 9999 and there are no duplicates, the relative record number is taken directly from the student file number. Figure 5 shows relative record numbers taken from the student file number being used to update student addresses.

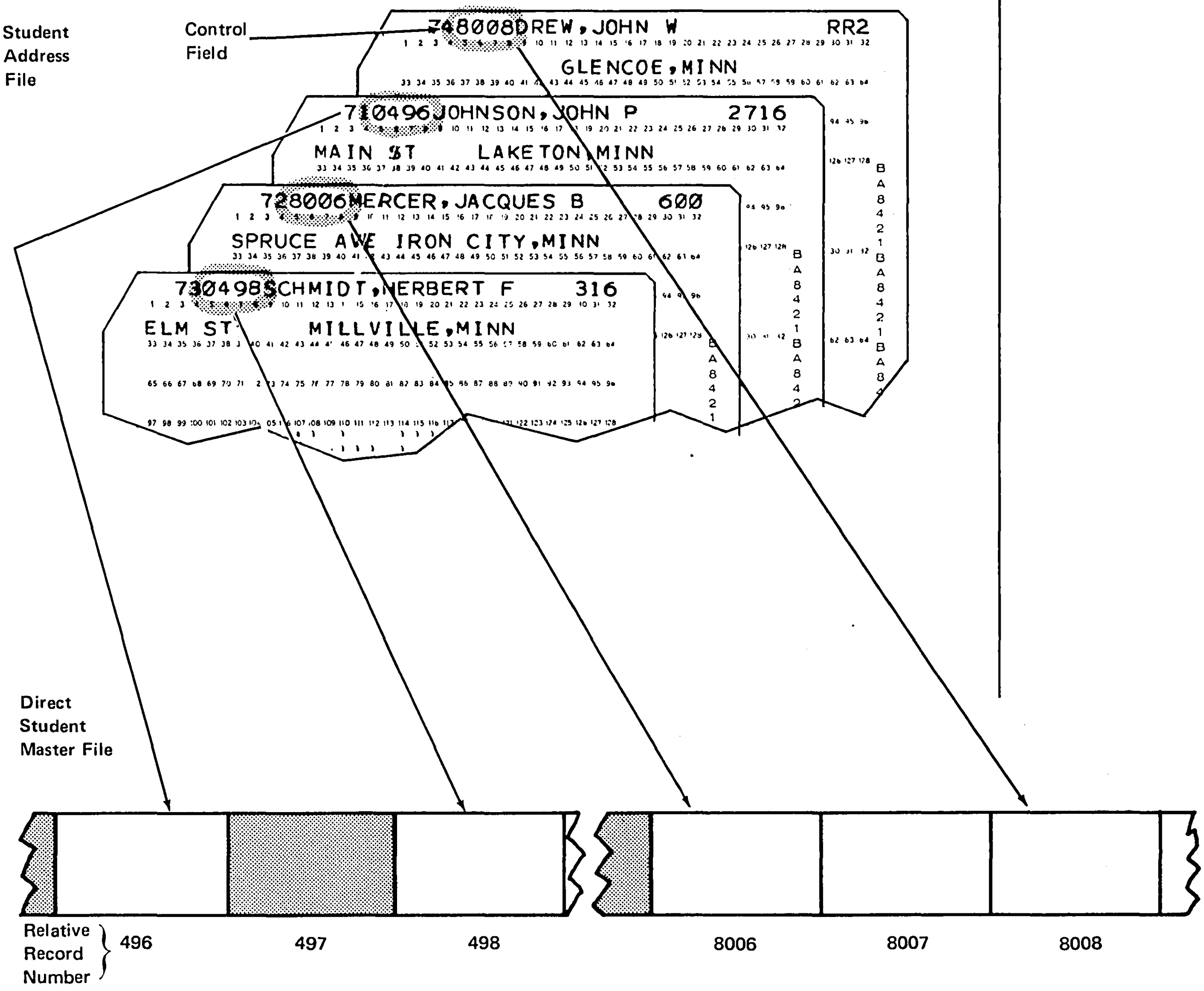


Figure 5. Relative Record Numbers Corresponding Directly to a Control Field

Conversion Method

Conversion refers to any technique for obtaining a desirable range of relative record numbers from the control fields of the records. The conversion method must be used when the values in the control fields cannot be used directly as relative record numbers. For example, employee numbers in a factory range from 0001 to 1500, but only 450 numbers are in use since numbers belonging to employees who have retired or terminated have not been reused. A file large enough for 1500 records is not needed; therefore, a technique for converting the employee numbers to approximately a 1 through 500 range must be found. This provides 50 locations for file expansion.

When the conversion method is used, every possible control field in the file must convert to a relative record number in the allotted range (in this case, 1 through 500), and the resulting relative record numbers should be distributed evenly across the allotted range so that there are few *synonym* records. Synonym records are two or more records whose control fields yield the same relative record number (see *Synonym Records*). Your program must allow for synonyms if they are generated.

One way to convert the range of employee numbers from 1500 to 500 is to divide the employee number by 3 and drop the remainder (thus 3 becomes 1; 6 becomes 2; 1500 becomes 500). However, unless the file is perfectly distributed, there will be synonym records. For example, if the numbers 6, 7, and 8 are present, all three become relative record number 2.

An alternate technique that produces fewer synonyms is to divide the employee number by 2 and drop the remainder. This compresses 1500 numbers to 750. There are 300 unused locations in this case, but fewer synonyms.

If there is no sequence to numbers in a control field (such as part number), a conversion technique that produces random numbers can be used. The resulting numbers should be distributed evenly within the selected range (depending upon the number of record locations needed) and should be suitable as relative record numbers (positive, whole numbers). One such technique is squaring the number in the control field and selecting certain digits from the resulting number as the relative record number. The calculation must be performed every time the program must seek a record. For example, suppose you have part numbers that consist of six digits, with certain digits having a special meaning. No two part numbers are alike. The part number is squared and, of 11 resulting digits, the center four are used as the relative record number for the parts inventory file.

Part number = 468152

$$468152 \times 468152 = 2191\boxed{6629}104$$

Relative record number = 6629

Since four digits are selected, random numbers from 1 to 9999 could be developed. Therefore, a file containing 10,000 record locations should be provided for the parts inventory.

Even the technique used in the example above is likely to produce synonym records, since the center four digits of the square of two different part numbers can be identical. If a conversion technique produces too many synonyms, it may be necessary to find a different technique or even a different file organization. The complexity of processing and programming for synonyms may outweigh the advantages of direct file organization.

Synonym Records

Two or more records whose control fields yield the same relative record number are called synonym records. Synonyms have the same relative record numbers, but contain different data. Only one of a group of synonyms can be stored in the record location which agrees with its relative record number. Therefore, you must find a way to store and retrieve the other synonyms.

One way to handle synonyms is to link them together so that all can be found by locating the first, as in Figure 6. The first record is stored in the record location indicated by its relative record number. That location is called the *home location;* the record placed there is called the *home record.* The first synonym is stored in the first unoccupied record location (a location for which no relative record number was developed). The relative record number of the second location is then stored in the home record; that is, the first synonym is *linked* to the home record. The second synonym, if present, would be stored in the next unoccupied record location and would be linked to the first, and so forth. In Figure 6, all records that are synonyms are loaded into the file after records that can be stored in their home location have been loaded. Loading the records in this manner simplifies the programming because the coding for loading synonym records can be isolated into a separate program.

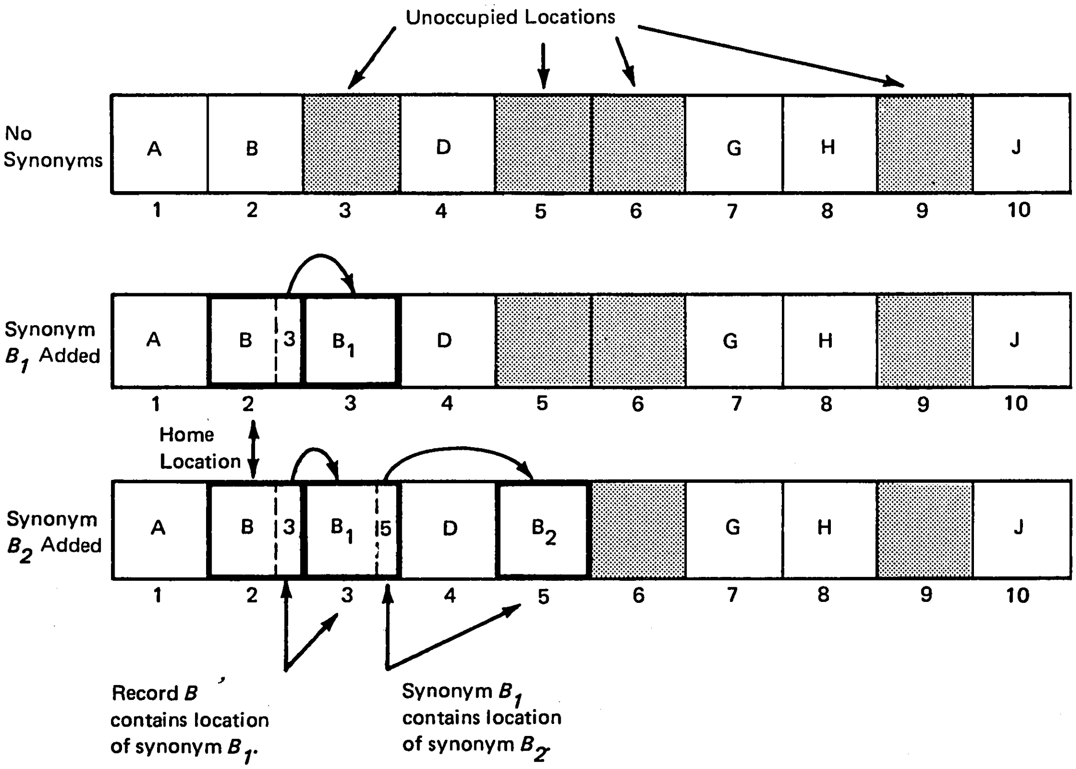


Figure 6. Storing Synonym Records in a Direct File

If a new record is added to the file, but its home location is occupied by a synonym for a different record location, that record must be treated as a synonym for its home location. Figure 7 shows the file that resulted from the addition of synonyms in Figure 6. The home location for record C is occupied by a synonym for record B, so record C is placed in the first unoccupied location. Since record B_1 is already linked to record B_2, record C must be linked through B_2 to its home location.

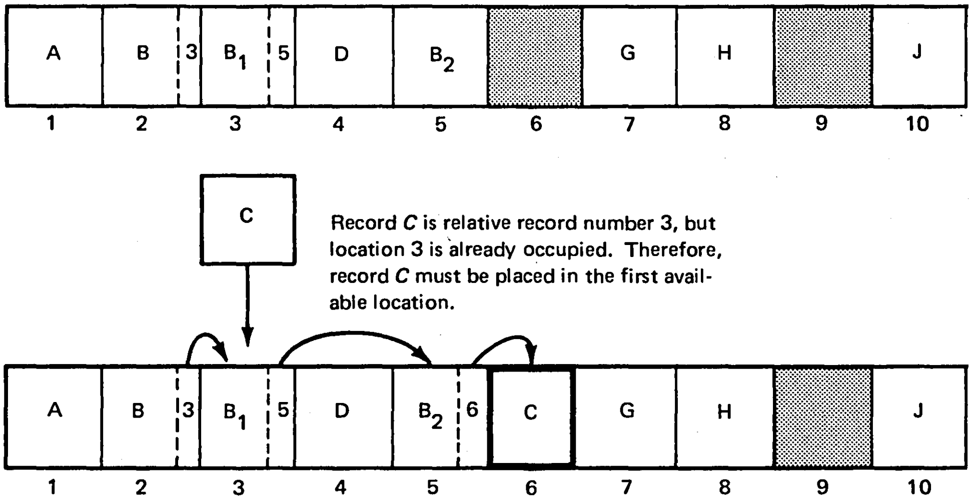
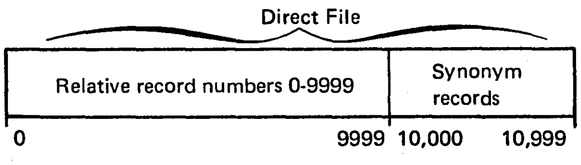

A
B
3
B_1
5
D
B_2
G
H
J
1
2
3
4
5
6
7
8
9
10
C
Record *C* is relative record number 3, but location 3 is already occupied. Therefore, record *C* must be placed in the first available location.
A
B
3
B_1
5
D
B_2
6
C
G
H
J
1
2
3
4
5
6
7
8
9
10

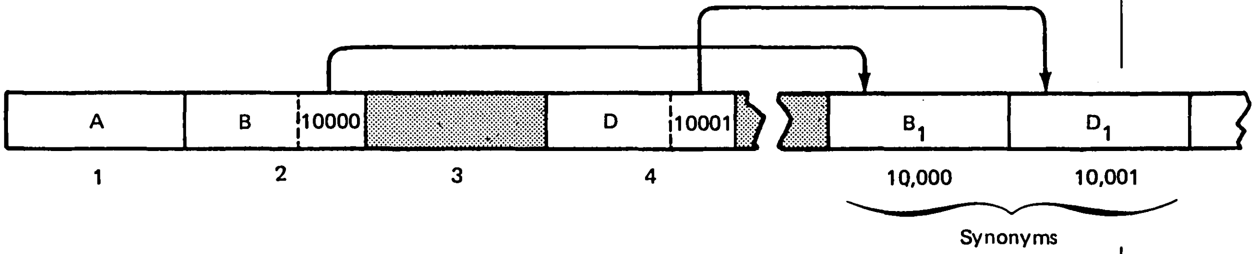
Figure 7. Storing a Record When Its Home Location Is Occupied

When you process a direct file containing synonyms, you must verify every record retrieved. For example, when you retrieve relative record 3 from the file in Figure 7, you get record B_1, which is a synonym for relative record 2. This is unacceptable. However, if you check the record retrieved, you find that it is a synonym. You can now chain to the relative record location, if any, indicated by the first record and retrieve the second record. You can continue this process until you find the record you want or until the chain of synonyms ends. In this case, you may have an error condition because the requested record is not in the file.

A similar method for handling synonyms is to set aside a portion of the file for synonym records. Suppose, for example, a file for 8500 records is set up to provide relative record numbers between 0 and 9999. By actually setting aside enough area for 11,000 records, any synonyms developed can be stored in record locations from 10,000 to 10,999.



The relative record number of a synonym is stored in the home location, and a chain of synonyms is built as in the previous method.



This method is faster if records are added to the file because a home location is kept free for each different relative record number. Only one seek is required for records without synonyms. However, this method wastes more space because 11,000 locations are used for 8500 records.

Other methods for handling synonyms can be devised. Whatever the method used, plan on extra accesses for synonym records and coding for verifying the records.

CREATING A DIRECT FILE

To create a direct file, you must define a disk file as a chained output file. In this way, the file is uniquely identified as a direct file to disk system management. Disk system management then allocates disk space for the file and clears that space to blanks. From that point, the method you use to write data records on the file depends on whether or not you must check for synonyms among those records.

Whether or not you must check for synonyms, relative record numbers are used in your program to make the corresponding record locations available for loading. The data used as a relative record number can come from a field in the input record, or it can be created in your program.

Creating a Direct File without Synonyms

If you will not have synonyms, you can load records into a direct file in a single pass. Record locations cannot be inspected before they are filled with data. If a synonym is encountered, it is written over the previous record and the previous record is lost.

Creating a Direct File with Synonyms

If you have synonyms, you can create a direct file by using more than one pass to load records into the file. The exact method you use depends on your scheme for handling synonym records (see *Relative Record Number, Synonym Records).* Your first job must define the disk file as a direct file and clear the file to blanks. Once the file has been cleared, one or more subsequent jobs can be run to read record locations and check for synonyms while loading the file.

PROCESSING A DIRECT FILE

Direct files can be processed in three ways:

1. Consecutively

2. Randomly by relative record number

3. Randomly by ADDROUT file

Consecutive Processing

Direct files are often employed where the activity of a file is low and direct inquiry of the file is necessary. However, when the activity on a direct file is high for certain jobs, such as writing a report where the entire file is listed, you may want to process the file consecutively.

Consecutive processing of direct files is similar to consecutive processing of sequential files. Record locations are processed one after another until end of job requirements are met. Remember that a direct file is cleared to blanks when it is created, and record locations which are not filled remain blank. Thus, in consecutive processing, blank record locations will be read along with those containing data. Your program should check for blank record locations and bypass them so that only valid records are processed.

When retrieving and updating a direct file consecutively, you also may want to check each record for synonyms and then handle the synonyms differently from other records. However, since consecutive processing is not dependent upon relative record numbers, a direct file can be processed consecutively without regard for synonyms.

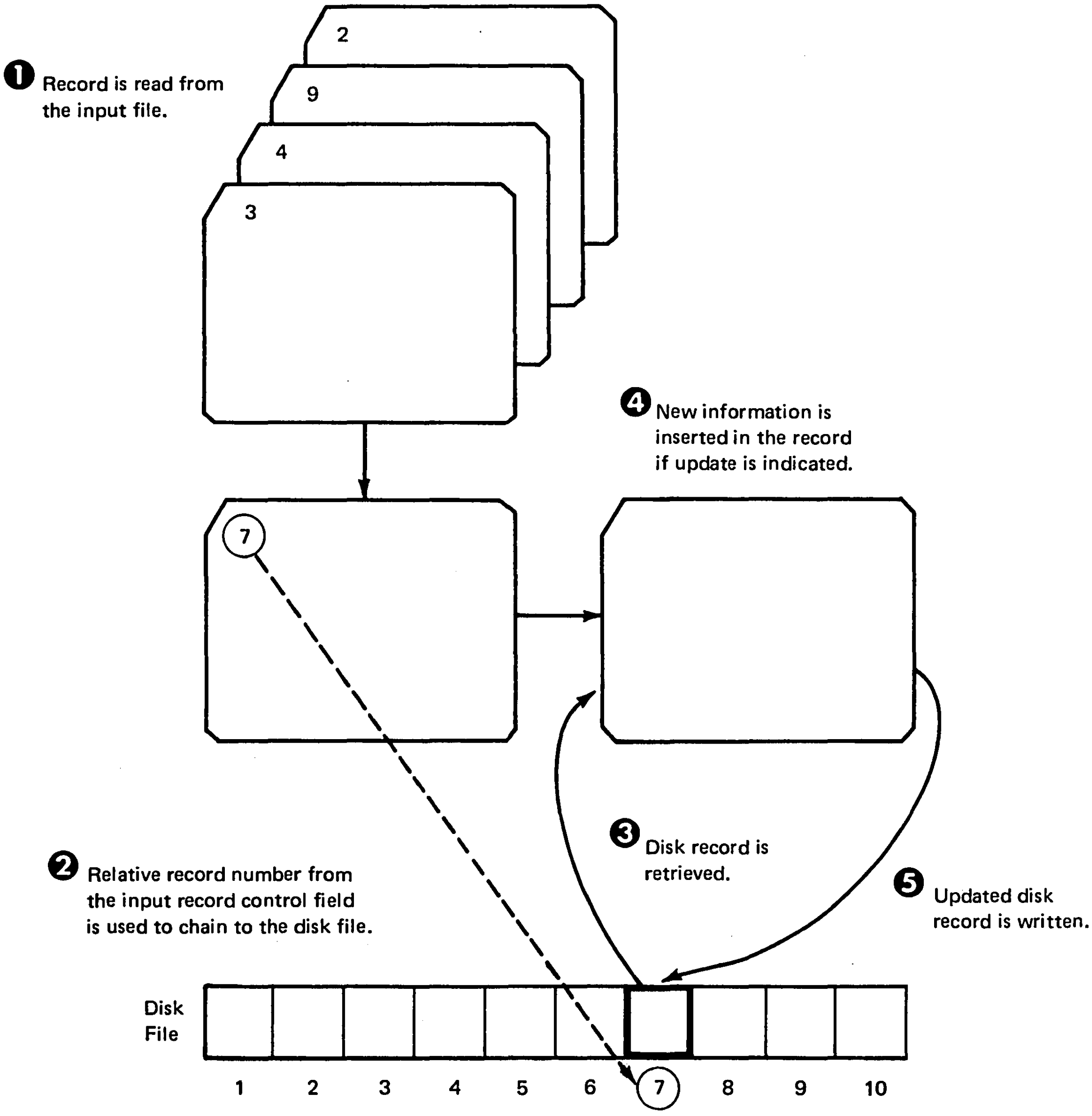
Random Processing by Relative Record Number

Remember that random processing of indexed files is accomplished by using the control field value (record key) to search an index. If a match is found, the record at the disk location contained in the index entry can be accessed. The control field value, therefore, is not related to the actual location of the record on disk. When processing randomly by relative record number, however, the relative record number is used by disk system management to calculate the disk location of the record. No index area and index search are required, since the control field value is directly related to the record location. Therefore, random processing by relative record number can be faster than random processing by key of an indexed file. If a large number of synonyms exist in the file, however, the advantage of fewer accesses required to retrieve a record may be negated by more complicated programming to handle synonyms and an increase in the average number of seeks per record due to synonyms.

Records can be processed either in an ordered or an unordered manner. Processing of records in order according to relative record number is usually faster than unordered processing since less movement of the disk access mechanism is required. Figure 8 shows the steps involved in random processing of a disk file by relative record number. In the figure, relative record numbers are obtained for control fields in the input records; however, they could also be generated by your program. Random retrieval includes steps one, two, and three in the figure; random update includes all five steps.

Random Processing by ADDROUT File

For a discussion on this type of processing, see Chapter 5.



❶ Record is read from the input file.
2
9
4
3
7
❹ New information is inserted in the record if update is indicated.
❸ Disk record is retrieved.
❷ Relative record number from the input record control field is used to chain to the disk file.
❺ Updated disk record is written.
Disk File
1
2
3
4
5
6
7
8
9
10

Figure 8. Random Processing by Relative Record Number

MAINTAINING A DIRECT FILE

Three file maintenance functions can be used to keep direct files current after they are created:

1. Adding records

2. Tagging records for deletion

3. Updating records

Adding Records

Unlike sequential and indexed files, direct files can have space available between existing records for records to be added. To add records to the file, the relative record number for the added record must first be determined. The location is then read into storage. If the location is blank, the record is stored. Otherwise, if the location already contains a record, the new record is stored as a synonym.

Tagging Records for Deletion

As in other files, records in direct files can be identified for deletion by a delete code. This code is usually a single character at a particular location in the record. When the file is processed, your program must check for the delete code; if the code is present, the record can be bypassed.

Since the delete code indicates that the record has been deleted, however, the record location is available for a new record. Either the location can contain a synonym for a different record or the location can be reused by assigning the relative record number to a new record. If the file contains synonyms, be careful not to delete synonym chaining information when you delete a record and reuse the location.

Updating Records

When you update records in a file, you can add or change some data on the record. The record to be updated is read into storage, changed, and written back on the disk in its original location. Records in a direct file can be updated consecutively or randomly.

Records are usually updated consecutively when you want to update all or most of the records in the file. Records are updated in order. However, synonym records in a consecutively processed direct file may require special handling.

To update your file randomly, you must specify the relative record number of the record you want. The relative record number is used to find the record in the file so it can be moved into storage for updating.

Chapter 5. Record Address Files

Record address files are input files that indicate which records are to be read from disk files and the order in which the records are to be read. There are two types of record address files:

- Files containing relative record numbers

- Files containing record key limits

FILES CONTAINING RELATIVE RECORD NUMBERS (ADDROUT FILES)

A record address file that contains relative record numbers is called an ADDROUT file. ADDROUT files are comprised of binary 3-byte relative record numbers that indicate the relative position (first, twentieth, ninety-ninth) of records in the file to be processed.

Creating an ADDROUT File

An ADDROUT file is created by the Disk Sort program. The input for the Sort program is a file which may be organized as a sequential, indexed, or direct file. The output from the Sort program is a new file consisting of relative record numbers. This file of relative record numbers may then be used during the processing of the original file to provide accessing of the file in a sequence different from the sequence in which the file is stored on disk. For more information, see the *IBM System/3 Disk Sort Reference Manual,* SC21-7522.

The following three points should be considered for using ADDROUT files:

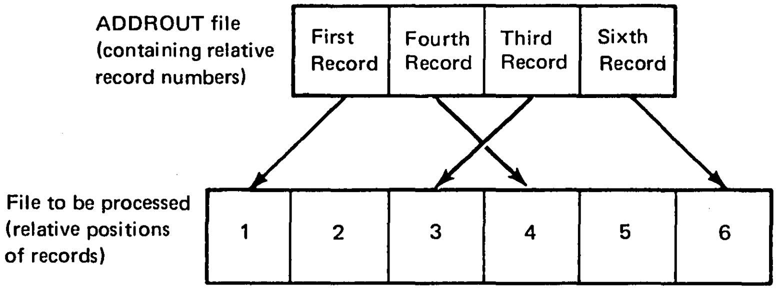
1. One file can be sorted in several sequences based on different control fields in each record of that file. To avoid sorting the entire file each time a different sequence is required, several ADDROUT files can be created by sorting the input file to be used in your programs in several ways. For example, you have a transaction file in order by stock number. By performing two ADDROUT sorts on the transaction file, you could have one ADDROUT file sequenced by customer number and another by invoice number. Consequently, you can access the transaction file by several sequences: stock number, customer number, or invoice number.

2. An ADDROUT file requires less disk space to process a file than the output file of a tag-along sort because the output records of the ADDROUT file are only three bytes long.

3. If an ADDROUT file is used to process multivolume files, all volumes of that file must be mounted during processing because the next record required may be on any volume.

Processing by an ADDROUT File

All types of file organizations (sequential, indexed, or direct) used as primary or secondary files can be processed by ADDROUT files. When an object program uses an ADDROUT file to process another file, it reads a relative record number from the ADDROUT file, then locates and reads the record situated at that relative position in the file being processed. Only those records whose relative record numbers are located in the ADDROUT file are processed. Records are read in this manner until the end of the ADDROUT file is reached. Figure 9 shows an ADDROUT file used to process a disk file.


ADDROUT file (containing relative record numbers)
First Record
Fourth Record
Third Record
Sixth Record
File to be processed (relative positions of records)
1
2
3
4
5
6

Note: The object program will read the ADDROUT file and find that the first record to be read is in relative position one of the file being processed. The second record to be read is in relative position three. Since all records are not read, processing by ADDROUT file is random processing.

Figure 9. Using an ADDROUT File to Process a File

FILES CONTAINING RECORD KEY LIMITS

A record address file with record key limits contains the lowest and the highest key fields for a specified section of an indexed file. Record address files containing record key limits can be entered from disk, card, or printer-keyboard. They are used to process only indexed files. When a section of an indexed file is processed using record key limits, the processing method is known as *sequential within limits.*

Example: You have an indexed file, but want to process only the records with keys 2,000 through 3,000. The record key limits in this record address file would be 2,000 (lowest) and 3,000 (highest key field). Through RPG II specifications the appropriate section (records with keys 2,000 through 3,000) of the indexed file would be processed.

Creating a File with Record Key Limits

In order to create this type of record address file, you must first determine the record key, such as a customer number, of the file to be processed. Each record in the record address file contains the record key limits (the low record key and the high record key) to be used for processing. The file can contain several sets of limits.

For instance, in the example explaining sequential within limits in Chapter 3, the customers were divided into four regions. If you only wanted to process the records for customers in region 3, the low record key would be 30,000 and the high record key would be 39,999. The record in the record address file would specify these limits like this:

```
3000039999
```

Processing Sequentially within Limits

Processing a section of an indexed file by record keys is known as *sequential within limits.* The object program uses one set of limits (one record in a record address file) at a time. Records are read according to the arrangement of the record keys in the section of the indexed file specified by the limits. When the records identified in one section are read, the program reads another set of limits from the record address file. The program continues reading records in this manner until the end of the record address file is reached.

It is not necessary for the record keys that were specified as limits to be in the file. For example, if you specify the high record key as 2999 and the last record in that section of the file is 2800, the program will read another set of limits from the record address file after record 2800 is processed. If you specify the low record key as 2000 and record 2000 is not in the file, the record with the next higher key will be read providing that record is not higher than the high limit.

PART II
ADVANCED DISK CONCEPTS

Chapter 6. Inquiry Program

In some data processing applications, customers may make inquiries that require immediate answers. One customer may want the status of his account; another may want to know if an item is in stock for immediate delivery. To answer these inquiries, your program must be able to access certain disk records. The object program you use to retrieve this information is called an *inquiry program.*

Inquiry programs can be executed as part of a normal job stream, or they can interrupt other programs that are executing provided the executing program can be interrupted. After a request for inquiry is made, the following things occur:

1. A program being executed is interrupted.

2. The current status of the program is stored on disk.

3. The inquiry program is loaded to retrieve and display the requested information.

4. The original program is reloaded.

REQUESTING INQUIRY IN AN INTERRUPT ENVIRONMENT

To interrupt a job prior to loading an inquiry program, you must make an *inquiry request.* To request inquiry, you must have a keyboard such as the IBM 5471 Printer-Keyboard (Figure 10) or the keyboard console attached to the IBM 5406 Processing Unit (Figure 11).

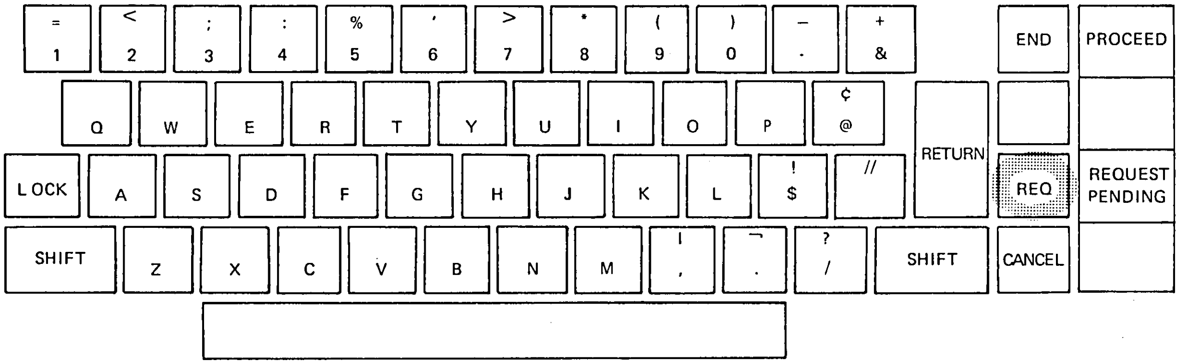
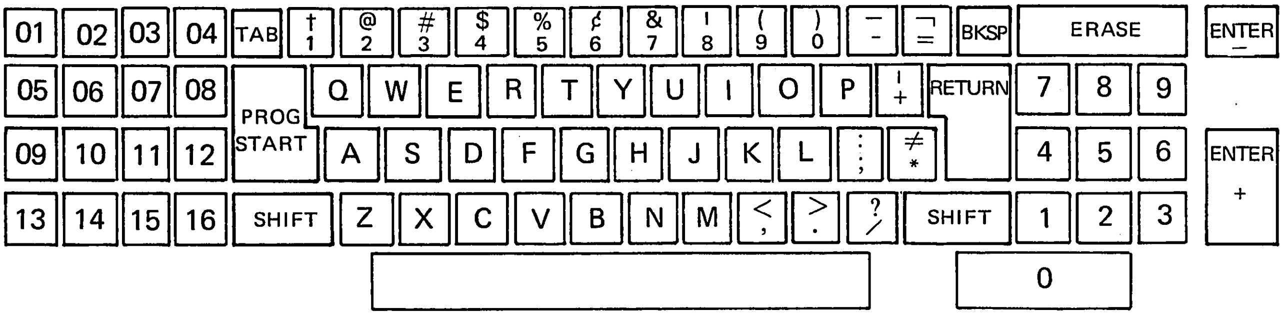


Figure 10. Keyboard Format of the 5471 Printer-Keyboard



01 02 03 04 TAB † 1 @ 2 # 3 $ 4 % 5 ¢ 6 & 7 ! 8 (9) 0 – - ¬ = BKSP ERASE ENTER –
05 06 07 08 PROG START Q W E R T Y U I O P | + RETURN 7 8 9
09 10 11 12 A S D F G H J K L : ; ≠ * 4 5 6 ENTER +
13 14 15 16 SHIFT Z X C V B N M < , > . ? / SHIFT 1 2 3
0

Figure 11. Keyboard Format of the Model 6 Keyboard Console

If you want to make an inquiry request using the 5471, press the REQ key on the keyboard. If you want to make an inquiry request using the keyboard on the IBM 5406, switch on the Inquiry Request switch on the system control panel (Figure 12). Either inquiry request indicates to disk system management that an inquiry program is about to be loaded and the program that is executing must be stored on disk. The OCL statements for the inquiry program are then initiated from the keyboard. (At least the READER statement indicating what input device contains the OCL statements must be entered from the keyboard.)

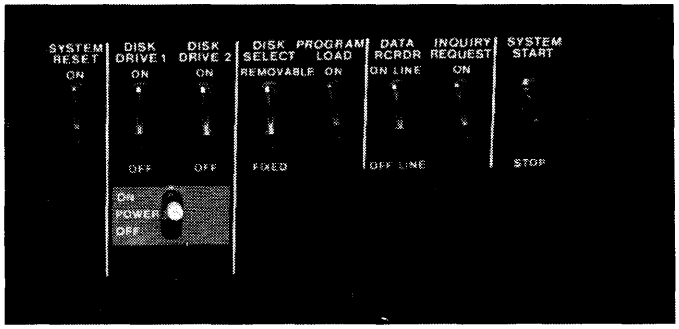


Figure 12. System Control Panel on the Model 6

Functions of the Request Key

The Request key can be pressed to:

1. Interrupt an executing program and thereby enter the interrupt environment.

2. Initiate an inquiry program that is already in main storage waiting for an inquiry request.

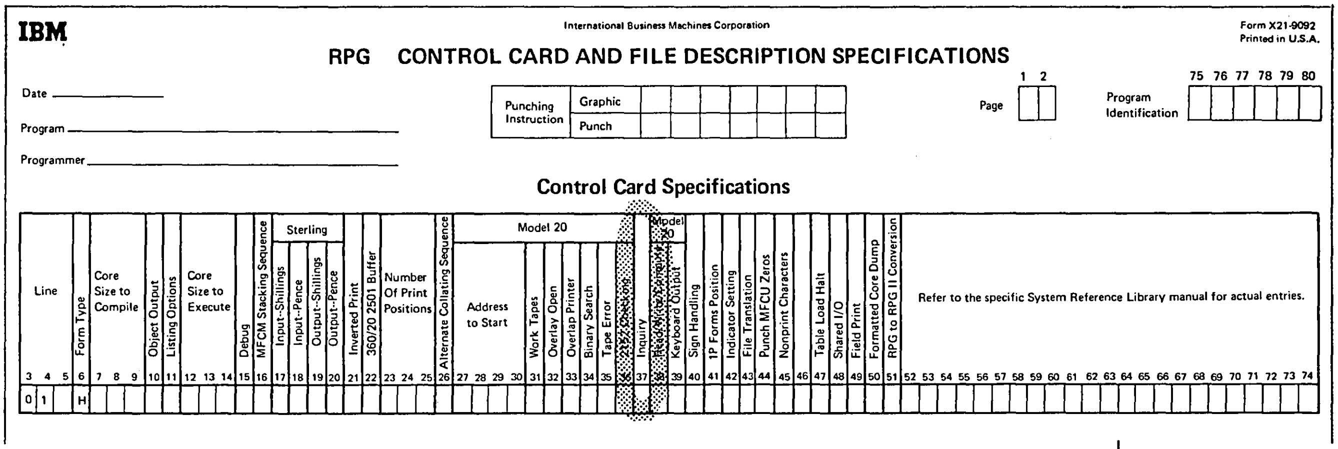
3. Initiate the reading of input data from the printer-keyboard for a program described in the second item of this list.

CLASSIFYING PROGRAMS FOR INQUIRY

Not all programs can be interrupted by an inquiry program. By coding specifications in column 37 (Figure 13) on the RPG II Control Card sheet, you determine whether the program can be interrupted. The entries which classify the program are:

- ƀ (blank) – A ƀ-type program is a processing program that does not recognize an inquiry request. It cannot be interrupted.
- B – A B-type program is a processing program that recognizes an inquiry request, and, therefore, can be interrupted or stored on disk.
- I – While I-type programs can be loaded as inquiry programs in an interrupt environment (see note), a program is usually classified as an I-type when it is used as an inquiry program that is to remain in main storage for the servicing of inquiries. An I-type program can be executed *only* by an inquiry request (pressing the Request key). An I-type program cannot be interrupted and stored on disk. If an input file is to be entered from the printer-keyboard for an I-type program, you must again press the Request key to initiate reading of the input file.

Note: An inquiry program that interrupts a B-type program can be classified as B, ƀ, or I-type. An inquiry program loaded to perform a complete job is usually classified as a B-type program. An inquiry program loaded to answer one request or few requests is usually loaded as an I-type program (see *Planning Inquiry Programs* for further information). If a B-type program is rolled out by an inquiry program also classified as B-type, the inquiry program must complete execution before another inquiry request is made.



IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date

Program

Programmer

Punching Instruction	Graphic
	Punch

Page (1 2)

Program Identification (75 76 77 78 79 80)

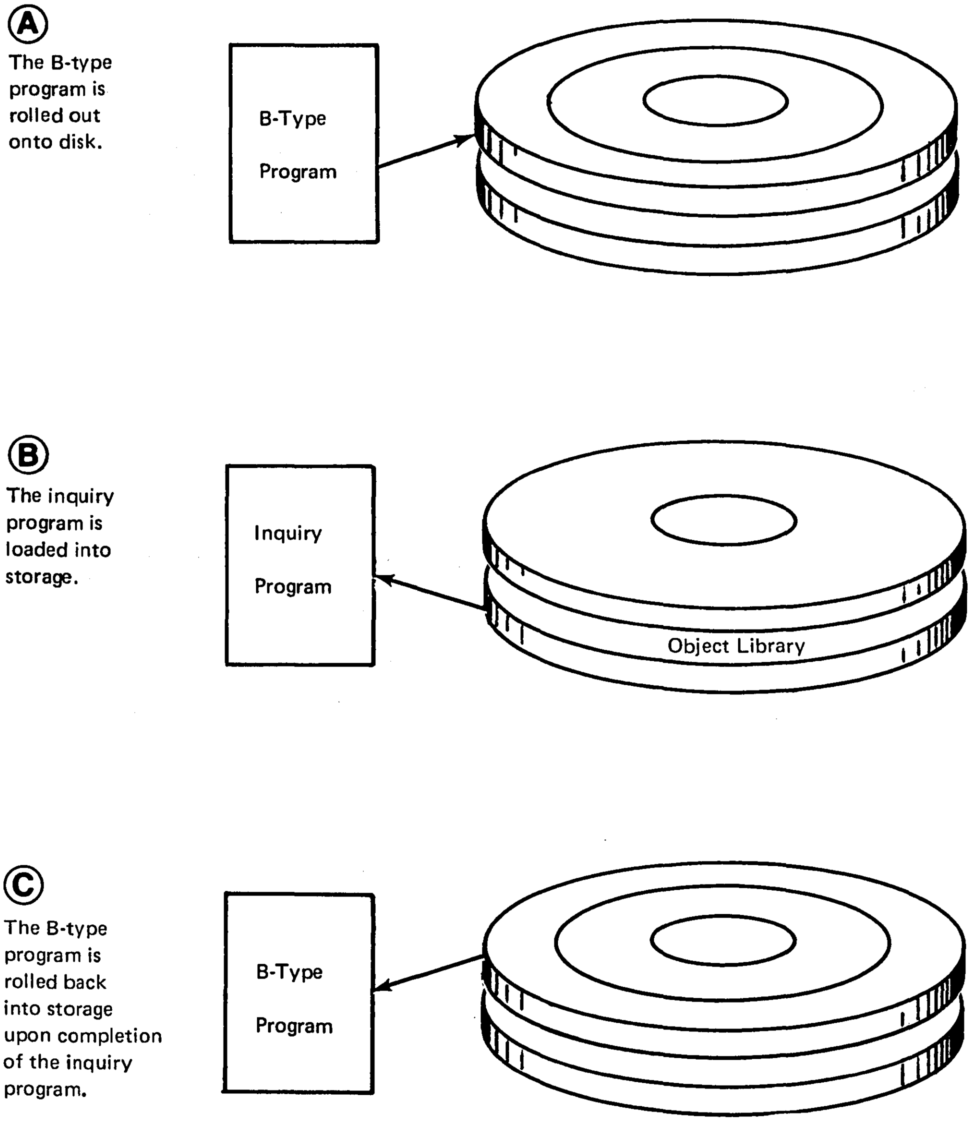
Control Card Specifications

								Sterling								Model 20								Model 20																
Line	Form Type	Core Size to Compile	Object Output	Listing Options	Core Size to Execute	Debug	MFCM Stacking Sequence	Input-Shillings	Input-Pence	Output-Shillings	Output-Pence	Inverted Print	360/20 2501 Buffer	Number Of Print Positions	Alternate Collating Sequence	Address to Start	Work Tapes	Overlay Open	Overlap Printer	Binary Search	Tape Error	2152 Checking	Inquiry	Read/Write Compiler	Keyboard Output	Sign Handling	1P Forms Position	Indicator Setting	File Translation	Punch MFCU Zeros	Nonprint Characters		Table Load Halt	Shared I/O	Field Print	Formatted Core Dump	RPG to RPG II Conversion	Refer to the specific System Reference Library manual for actual entries.
3 4 5	6	7 8 9	10	11	12 13 14	15	16	17	18	19	20	21	22	23 24 25	26	27 28 29 30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	H																																					

Figure 13. Inquiry Specification on the Control Card Sheet

INQUIRY IN AN INTERRUPT ENVIRONMENT

An inquiry program can be loaded into storage as any other program, or it can be loaded when an inquiry request is made to interrupt a program that is executing. When your system is controlled by one program at any one time, you have a *dedicated system.* Therefore, in an interrupt environment you must interrupt the executing program to allow the inquiry program to control the system. You request an interrupt by pressing the Request key on the printer-keyboard. You can only interrupt B-type programs. As soon as the Request key is pressed, the system sets an indicator and the executing program completes the execution cycle it is in. A system routine called roll-out then transfers the B-type program from main storage onto disk, retaining the current status of the program (Figure 14, insert A). Space is allocated for the rolled out program at system generation time. (See the *IBM System/3 Disk System Operator's Guide,* GC21-7508, or the *IBM System/3 Model 6 Operator's Guide,* GC21-7501, for system generation procedures.) *Chapter 10. Storing Programs and Procedures on Disk* describes the scheduler work area size including space requirements for roll-out/roll-in.



The B-type program is rolled out onto disk.

The inquiry program is loaded into storage.

The B-type program is rolled back into storage upon completion of the inquiry program.

Figure 14. Roll-Out and Roll-In

PLANNING INQUIRY PROGRAMS

Since B-type programs can be interrupted, you must determine what types of programs should be classified as B-type. Usually long reports that do not have to be finished immediately are classified as B-type. Such a report might be an end-of-month stock status report.

Inquiry programs that can interrupt B-type programs can be classified as ƀ, B, or I-type. For example, suppose you are running an end-of-month stock status report, and now find you must run a payroll job. The payroll job can roll out the stock status job to satisfy this requirement. Another example of an inquiry program that might need to be loaded immediately would be a request to determine where a certain inventory item is located so that it can be shipped. Since the inventory file is online for the stock status report, the location of the item could be determined quickly by an inquiry program.

Those programs you do not want rolled out should be ƀ-type. For example, you may be running a payroll job and checks are positioned for the printer. You may not want the payroll program rolled out since the operator may have to remove the checks and not reposition them correctly. If you are running a teleprocessing program, you cannot roll out the program because you will lose telephone connections.

Programs classified as I-type can serve two purposes. In dual programming (see Chapter 7 for information on the dual programming feature), an inquiry program can be loaded into one level and remain there to service inquiries. Such a program must be classified as I-type. In a dedicated system, an I-type program could be loaded for a length of time to answer requests. For example, an I-type program could be loaded during the second shift of a day to answer inquiries into the amount or location of items in a warehouse. An I-type program remaining in main storage can only be executed by pressing the Request key.

Chapter 7. Dual Programming Feature

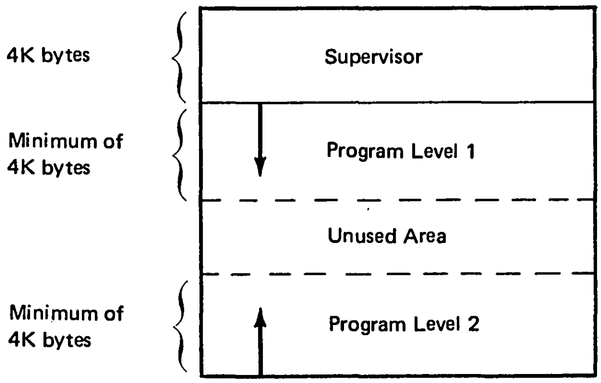
Under normal operating conditions, only one program can be processed at a time. With the dual programming feature (DPF), you can have two programs in main storage at the same time. Only one, however, can be executing instructions at any one time. This feature applies only to the IBM System/3 Model 10 Disk System.

Note: A modification to the dual programming feature will be made in a future release which will affect the object program size and program initiation/termination procedures. The following discussion, however, will help you understand the basic concepts of the feature. After reading this chapter, you can refer to the *IBM System/3 Disk System Operating Control Language and Disk Utilities Reference Manual,* GC21-7512, for planning information on the feature modification.

When DPF is operating, main storage contains the supervisor and two programs. Control is transferred from one program to the other whenever the program that is executing must await completion of an input or output operation. For example, one program requests a print operation, but the printer is still busy with a previous request. Control is then transferred to the other program. Similarly, one program requests that a card be read for processing. Since the program must wait until reading is completed before it can process the data, control is transferred to the other program. Similarly, control is transferred when a halt occurs in one program level.

Most programs have a significant amount of time when they are waiting for I/O completion. If both programs are waiting, the program whose I/O is completed first receives control.

Figure 15 shows how main storage is organized in a DPF environment. The supervisor occupies a minimum of 4K (4,096) bytes of storage in DPF. The storage areas occupied by the two programs are called program level 1 and program level 2. Each level must have a minimum of 4K bytes if the level is executing instructions.



The arrows indicate the direction in which storage is allocated to each level. If the two programs do not occupy the entire amount of storage you have, an unassigned area exists between program levels. This area can then be used by disk system management to increase the efficiency of your system operation.

Figure 15. Main Storage in a DPF Environment

ADVANTAGES OF RUNNING PROGRAMS IN A DPF ENVIRONMENT

Main Storage

DPF enables you to make more efficient use of your system storage. For example, if you were to run a 6K program on a 16K system in a dedicated environment, you would only be using 9K of your storage:

Program = 6K

Supervisor = 3K

Used storage = 9K

Consequently, 7K is unused.

In a DPF environment, you could run two 6K programs on a 16K system and use the entire storage capacity:

Two programs = 12K

Supervisor = 4K

Used storage = 16K

Processing Time

DPF permits more efficient use of the computer's processing time. When a program is executing, the processing unit is executing the program's instructions. When instructions are not being executed, the processing capabilities of the computer are not used. For example, if an instruction cannot be executed because data is not available to be processed (waiting for a card to be read) or because a device is not ready to execute the requested instruction (printer is busy with a previous print instruction), execution of the program is suspended until the required conditions are satisfied. When execution is suspended, the computer's processing time is lost because no instructions can be executed. DPF allows control to transfer to another program. That program can then begin executing instructions, thereby using the processing time.

Inquiry and Teleprocessing

Two system operations for which DPF is most effective are the inquiry function and teleprocessing (BSC). In a dedicated environment, an inquiry program must reside in storage or be loaded every time a request is made, consequently rolling out a program that is executing and rolling it back in at the end of the inquiry. In DPF, the inquiry program can be loaded into one program level, and another program can still execute in the second level.

If you are using teleprocessing (BSC), one program level can be dedicated to teleprocessing; the other level can be available for running other programs. For example, if messages are being relayed from one terminal to another, program level 2 can be assigned to teleprocessing. Although messages are not relayed constantly through the day, the teleprocessing program may have to be in storage at all times. Therefore, if the teleprocessing program is loaded into program level 2, it is available when needed. When the teleprocessing program is inactive, normal processing programs can use system resources.

Input/Output Devices

With proper planning, DPF also enables you to use your system input/output devices more effectively. In a dedicated environment, you may run a program to copy one disk to another. The program is using only one I/O device, a disk. The MFCU and printer are not used. In DPF, you could have two programs in storage. While one program is using the disk to copy one disk to another, the other program can use the MFCU or the printer.

CONSIDERATIONS FOR OPERATING UNDER DPF

You must consider the following points when planning to use DPF:

1. You must determine that you have enough storage. Because the supervisor requires 4K bytes, you could not, for example, run one 8K program and one 5K program on a 16K system. One of the two programs could not be loaded.

2. Two programs in storage must use the proper combination of I/O devices. Both program levels cannot use the MFCU or the printer.

   For example, if you were running two jobs that both required the printer, such as an invoicing and a sales analysis job, one program could not execute because the printer would not be available. Disk drives and the printer-keyboard can be shared depending upon the type of disk file processing. Figure 16 shows the restrictions when a data file is shared by two program levels.

		Program Level 1		
		Read a File	Create or Add to a File	Update Records in a File
Program Level 2	Read a File	Yes	No	Yes
	Create or Add to a File	No	No	No
	Update Records in a File	Yes	No	No

Figure 16. Disk File Processing of a Data File Stored by Two Program Levels

3. The operator should be careful when the printer-keyboard is used as:

   a. The system input device for both program levels.

   b. The system input device in one level and as an input device for an RPG II program in the other level.

   If the printer-keyboard is used by both levels, the operator must first determine which level is requesting information. The system provides this information by printing out a 1 or a 2 depending on whether the level using the printer-keyboard is program level 1 or program level 2.

   The performance of DPF may also be less efficient when the printer-keyboard is used because the operator may hold up your system when keying in information. This may happen because keyboard operations tie up the system transient area (an area used by the system to load programs for special purposes) so the other program level cannot use that area during keying.

4. The following IBM programs cannot use DPF because they require dedicated use of the system: RPG II Compiler, Library Maintenance, Basic Assembler, and IBM 1255 Utility program.

   Note: Object programs denoted by a LOAD * OCL card cannot be loaded into level 2. In order for an object program on cards to be loaded into level 2, it must first be copied from the reader to an object library and then loaded from the object library.

5. File planning is necessary to minimize the increasing access time. For example, if two programs were using two separate files on the same disk (Figure 17), the access arm may have to move every time each program requests I/O. Movement of the access arm will increase access time, slowing the performance of the program. To avoid this problem, it is most advantageous to have files for each active program on separate disk drives. You could, however, have separate files on two removable disks or one file immediately above the other file on fixed and removable disks as shown in Figure 18.

   Keep the following points in mind when planning your files:

   a. If two programs reference the same removable disk unit (R1, R2), they *must* be processing the same disk pack because you cannot change a pack on that unit for each program's I/O request.

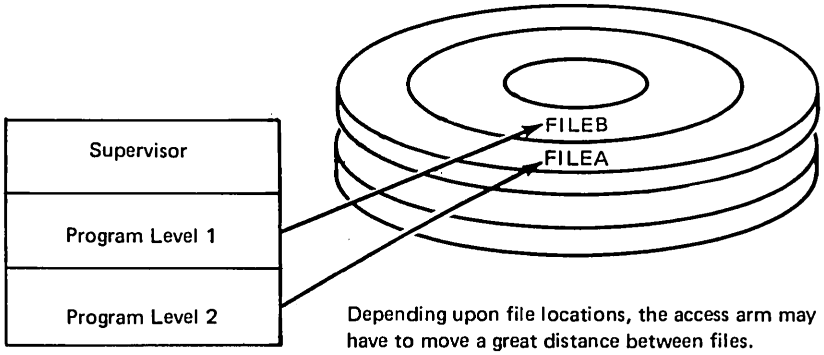
   b. If you load programs or procedures from a disk, or use a disk for IPL, the disk cannot be removed. In this case, it may be best to have programs and procedures on the fixed disk, leaving the removable disk free for changing.

   c. If one of the programs uses offline, multivolume files, the other program must not have files on the same volumes. When a disk is replaced for one program, it may contain files still needed by the other.

   d. If two programs are initiated, one of which uses data files on the system drive, the job that does *not* use data files on that disk should be initiated first. Program initiation involves numerous accesses to the system programs that could greatly increase your access time. If the program using data files on the system drive is initiated first, initiation of the other program will cause the access arm to move frequently from the data files to the system programs. If the program not using data files on the system drive is initiated first, it will read the system programs and be finished with the drive, leaving it free for initiation and execution of the program using the data files on the system drive.

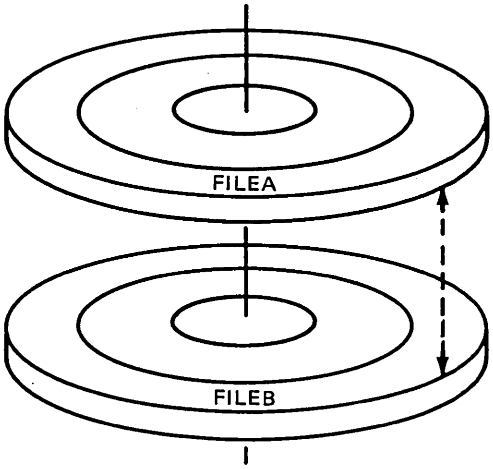
6. Individual programs will not necessarily run in any less time under DPF than they would in a dedicated environment. In fact, an individual program may take longer to run in DPF. A set of programs, however, may finish sooner than they would if they were run in a dedicated environment. For example, if you had two jobs to run, neither of the individual jobs may run in any less time. However, the set may be finished sooner in a DPF environment because one program would be using any processing time that the other could not use. If the programs were run consecutively, processing time may be wasted during each program's run.

7. DPF requires efficient job scheduling because of the preceding considerations. Suppose you had four jobs to be run requiring the I/O shown in Figure 19. Jobs 1 and 2 and Jobs 3 and 4 can be run together, because they do not require the same I/O devices. If Job 2 finishes before Job 1, you could run Job 4 because Jobs 1 and 4 do not require the same devices. If, on the other hand, Job 1 finishes first, Job 3 could not be run with Job 2 because both jobs require the printer for output.



Depending upon file locations, the access arm may have to move a great distance between files.

Figure 17. File Locations Causing Arm Movement



If files are located one above the other, the access arm may not have to move as far when each program requests I/O.

Figure 18. File Locations Causing Less Arm Movement

Program Level 1	Job 1 An inquiry program that: ● Reads printer-keyboard. ● Reads disk. ● Writes printer-keyboard.	Job 3 A stock status report that: ● Reads disk. ● Prints.
Program Level 2	Job 2 An inventory updating program that: ● Reads cards. ● Reads disk. ● Updates disk. ● Prints.	Job 4 A detail punching job that: ● Reads cards. ● Punches cards.

Figure 19. Job Scheduling for DPF

CONSIDERATIONS WHEN RUNNING SYSTEM/3 PROGRAMS IN A DPF ENVIRONMENT

The inquiry function, the Disk Sort program, the Alternate Track Assignment program, and the Disk Initialization program require special considerations when operating in a DPF environment.

Inquiry

An inquiry program can either reside in one of the two program levels in main storage or not reside in main storage. If it is not in storage, an executing program must be rolled out when an inquiry request is made. Remember the three classifications of programs for inquiry:

- I-type is an inquiry program that cannot be rolled out.
- ƀ-type cannot be rolled out.
- B-type can be rolled out.

If the inquiry program is in main storage, it must be an I-type program, and the other level must contain a ƀ-type program. The I-type program is then executed when the Request key is pressed.

If the inquiry program does not reside in main storage, it can be any of the three program types. However, if both partitions are active, you must have a B-type program in level 1 and a ƀ-type program in level 2 to operate inquiry when the inquiry program is not resident in storage. This is because the system does not allow level 2 to be rolled out upon an inquiry request. Consequently, no B-type program can reside in level 2.

When a B-type program is rolled out in level 1, the OCL statements for the inquiry program must be initiated from the printer-keyboard (at least a READER statement indicating what device contains the OCL statements must be entered). The same storage and I/O devices are available to the inquiry program as were available to the B-type program when it was rolled out. However, if the inquiry program is to share the same disk file as the B-type program, the file processing restrictions in Figure 16 apply.

If the printer-keyboard is used, the inquiry program should be an I-type program. Then the keyboard input will not be requested until the operator indicates that he is ready to enter data.

Disk Sort, Alternate Track Assignment, and Disk Initialization

The Disk Sort, Alternate Track Assignment, and Disk Initialization programs require a minimum of 5K bytes each to execute.

If they are loaded into program level 2, they are assigned 5K bytes unless you use an OCL PARTITION statement to reserve a larger area in level 2.

Note: If you load the Disk Sort program into level 1, all storage except 4K bytes for the supervisor is used unless level 2 is already active or you preassigned storage to level 2 using a PARTITION statement.

EXECUTING RPG II OBJECT PROGRAMS IN A DPF ENVIRONMENT

The amount of storage available for object program execution may differ from the amount of storage available for object program generation. When the storage sizes differ, you should indicate in columns 12-14 (Core Size to Execute) on the RPG II Control Card sheet the amount of main storage the object program can use. If this amount results in overlays, some of the DPF performance advantage may be lost. For an explanation of the entries to make in these columns, refer to the *IBM System/3 Disk System RPG II Reference Manual,* SC21-7504.

LOADING PROGRAMS IN A DPF ENVIRONMENT

A program can be loaded into either program level first. You tell the supervisor which system input device contains the job streams for the programs by selecting the device on the Dual Program Control switch. (Refer to the *IBM System/3 Disk System Operator's Guide,* GC21-7508, for further operating procedures.) When preparing your job streams, you should be aware of the OCL considerations. These considerations are discussed in detail in the *IBM System/3 Disk System Operation Control Language and Disk Utilities Reference Manual,* GC21-7512.

PART III
FILE PLANNING

Chapter 8. Choosing a File Organization

Part I of this manual described several disk file organizations that can be used with the IBM System/3 Model 6 and Model 10 Disk Systems and has explained the flexibility they provide to perform a variety of jobs. Because of the flexibility and variety of these different methods, it is important for you to analyze each of your jobs and choose the file organization method that gives you the best possible performance.

In many cases, the most appropriate file organization is immediately evident. Some applications, however, may require additional study because of their complexity, because a file is used in several jobs, or because special processing is required. Studying existing applications is an important aspect of planning for a data processing system. Decisions in this area must be made *before* programming begins, since efficiency of your data processing installation may be affected. This section describes factors to consider when making these decisions.

There are no absolute rules for choosing a file organization method. However, several characteristics of the file to consider are:

1. Use of the file

2. Volatility of the file

3. Activity of the file

4. Size of the file

USE OF THE FILE

The use of the file takes priority over all other considerations.

Is the file a master file? Recall that a master file is fairly permanent, is generally used in several jobs, and is often used with several other files. For example, a customer file contains a record for each customer. Each record may contain such data as customer name and address, shipping information, credit status, accounts receivable, and sales information. Although certain data in a record, such as accounts receivable, may change, the record remains in the file as long as the customer does business with the company. (These changes are made with a transaction file.) Since this master file contains so much information about each customer, it may be used in several jobs to produce various reports. Likewise, the file may be used with several other files, master or transaction.

In comparison to a master file, a transaction file contains records of a less permanent nature than a master file; transaction files may also contain data that is used to update a master file.

When choosing a file organization method for a master file, the major question to ask is: *What are the processing requirements of the file?* To answer this question, you must study the applications in which the file is used:

- Is the file used with other files or in several jobs?

  1. If so, what is the organization of the other files?

  2. If used with transaction files, are the transaction records ordered or unordered?

- Must the file be sorted for any jobs?

- Must the file provide for inquiry?

Using a Master File with Several Files or in Several Jobs

If a master file is used with several files (a transaction file, another master file, or both), the master file can be either sequential, indexed, or direct. The determining factors are the processing requirements of the various runs that will be using the file and the organization of the other files.

If the other files are ordered (sorted in the same sequence as the master file), then the master file may be either sequential or indexed. However, to process unordered files against a master file, the master file must either be indexed, and processed randomly by key, or direct. Random access of direct files is faster since a record can be retrieved by a single access. Similar access of an indexed file requires two accesses, one for the index and one for the record.

If the master file is used in several jobs, and records must be processed both in order and randomly, then indexed or direct is a better organization.

Note: Remember that a sequential file processed randomly by relative record number has the same retrieval and update characteristics as a direct file. Therefore, whenever the discussion says a direct file could be used, you can also use a sequential file if other file needs warrant that type of file organization.

Sorting a Master File

If the master file must be sorted for some jobs, you may not want to use an indexed or direct file because the Disk Sort program cannot produce a sorted indexed or direct file. That is, indexed and direct files can be sorted, but the sorted output file will be a sequential file. Instead of keeping the sorted file as the master file, the original file must be kept.

Inquiring Against a Master File

Most businesses need to get information from a file on an *inquiry* basis. An inquiry is a request for information from some type of storage.

Some jobs that emphasize the importance of immediate inquiry and response are:

Demand Deposit Accounting	What is the balance of account number 133420?
Inventory Control	How many of part number 55632 are on order?
Manufacturing	What is the quantity on hand for part number 16414?
Payroll	What are the year-to-date earnings for employee number 13862?

The IBM System/3 Model 6 and Model 10 Disk Systems provide for inquiry. The ability to use inquiry depends upon the organization of the file.

Where inquiry is required, a critical question in choosing the best file organization method is: *How fast must the inquiry be answered?* The less critical the response time, the greater the choice of organization and processing methods.

To decide how fast the inquiry must be, ask a series of questions. *Can the answer to the inquiry wait until the next updating of the specific master file?* If it can, then these inquiries can be treated as additional transaction records and so processed. File organization, in this case, could be either sequential, indexed, or direct depending on other processing needs. If the inquiry cannot wait, additional questions must be asked.

Can the answer wait until the end of the present computer run? If so, the disk pack containing the specific master file is mounted at the completion of the current job; the inquiry program is loaded; and the file is processed to produce the required answers. Obviously, response time varies considerably depending on (1) the job that is in progress when the inquiry arrives and (2) the organization of the file that is being searched for information.

A direct file or an indexed file processed randomly by key will usually provide the best response time. (If the desired record were the first record in a sequential file processed sequentially, the response is also fast, but this type of inquiry is rare.)

VOLATILITY OF THE FILE

The number of records added to or deleted from a file affects the type of file organization chosen. *Volatility* refers to number of additions and deletions. High volatility means many records are added and deleted; low volatility means few records are added or deleted.

If the file is highly volatile, you probably should not use a direct file. You may waste file space by allowing for synonym records or by not reassigning relative record numbers when records are deleted. If too many synonyms are produced, the average number of seeks needed to find a record could increase to a level where the direct file is slower to process than an indexed file. Also, if you are using the conversion method to derive the relative record number, future additions and deletions to the file could upset the balance of your conversion technique.

Records in sequential and indexed files are added at the end of the current records. If a file is sequential and the control fields of the added records are higher than the last record on the file, additions cause no problem. However, if they are not higher, and processing of the file depends on the records being processed in control field order, additions do cause a problem. In this situation, records added at the end of the file are out of sequence. To avoid this problem, the disk file must be re-created or sorted when such additions are made.

However, if additions are made to an indexed file, there is no need to rewrite the file. Records are also added at the end of the file, but the keys are in ascending order in the index. Thus, if the records must be processed in order, they can be processed sequentially by key. Thus, one of the advantages of an indexed file is that additions and deletions can be handled without rewriting the file.

However, as the number of additions increases, the efficiency of sequentially processing an indexed file decreases. Sequentially processing the added records by key requires more time than processing the records in the order in which they are written on the disk. This increase occurs because additional access arm movement is required to read records at the end of the file. The arm must move back and forth between the index and the records. Even if the original records are in sequence, the added records are not. The arm must make one additional move for each added record that is processed.

Thus, for a highly volatile file where records must be processed in order, a sequential file with consecutive processing is best although the file would have to be resorted after each addition job. However, if a highly volatile file does not require processing records in order, the file can be indexed and processed randomly by key.

If a highly volatile file requires both sequential and random processing, an indexed file is best. In this case, to overcome the problem of excessive access arm movement in order to retrieve records added at the end of the file, the file should be reorganized frequently.

ACTIVITY OF THE FILE

The next important consideration, after volatility, is the activity of the file. *Activity* refers to the number of accesses to a file. Activity is usually expressed as a percentage. For example, if the file has 6000 records and 12,000 transactions are processed randomly per day using that file, the activity is 200%.

As activity increases, consecutive processing becomes more efficient. This implies a sequential file with consecutive processing or an indexed file processed sequentially by key. Low activity justifies an indexed file processed randomly by key or a direct file.

Total activity against a master file may be reduced by sorting the transaction files so that only one retrieval of a master record is required for each group of transactions with the same key field.

For a high activity file, you should consider *batch processing.* This means the application does not require transaction records to be processed the moment they occur; some time lag is all right. Transactions can be accumulated, or batched, and processed at certain times. The time lag may be hours, weeks, or even months, depending on the application.

SIZE OF THE FILE

Multivolume Files

If your file is too large to fit on one disk (volume), you must consider the effect that a multivolume file has on processing. A multivolume file can be online or offline. Online means that all the volumes containing the file are running on disk drives during processing so that all the records are available for processing. Offline means that only part of the file is available for processing at any one time; the volumes must be removed and replaced with other volumes to process the entire file.

Offline Multivolume Files

If you are creating a sequential file or an indexed file, the file can be created as an offline multivolume file. When this type of file is being created, records are placed in consecutive order on as many volumes as needed. For multivolume indexed files, you must specify the highest record key for each volume. Only records with a key field less than or equal to the specified key will then be placed on the designated volume.

When you process an offline multivolume file sequentially, you mount a disk, wait until all the records have been read, then mount the next disk. If you have a 2-drive system, the first two volumes can be mounted, then the next two, and so on until all the volumes are processed.

Random processing of an indexed file can be done using an offline multivolume file but only if the file is created with this in mind. The records can be written on each volume according to a predetermined grouping. For instance, a customer billing procedure could be done according to groups so that Group 1 would be billed the first week in the month, Group 2 the second week, and so on. The customers in each particular group could be written on separate volumes. Group 1 could be on one volume, Group 2 could be on another volume, and so on. Then only the volume needed for each billing date would be mounted. The file could be processed randomly since all the records needed would be on the volume online.

Online Multivolume Files

If you are creating a direct file, the file must be created as an online multivolume file. When you create this type of file, you can use both fixed and removable disks. The file, however, cannot exceed the number of disks that can be on the system at one time.

When an online multivolume file is processed, the records in the file can be on different volumes but all the volumes must be online. Thus, this type of file must be used when you are processing your entire file randomly (sequential, indexed, or direct) and records may be needed from any one of the volumes.

Sorting a File

If the file will be sorted by the System/3 Disk Sort program, the size of the file also affects the choice of a file organization method.

The System/3 Disk Sort program uses disk *work areas.* A work area is space on the disk that the program uses to arrange records in the specified order. The size of these work areas must be considered when planning files that need sorting.

When an entire disk file is sorted and the output file contains all the data in the input file, the maximum size of the input file on a 1-drive system is a little less than half the total online disk storage drive capacity (a little less than one volume). On a 2-drive system, half the total online capacity is a little less than two volumes. In either case, the volume that contains the input file can be removed before the sort program starts writing the output file. Another volume can be mounted,and in this manner, the input file can be preserved.

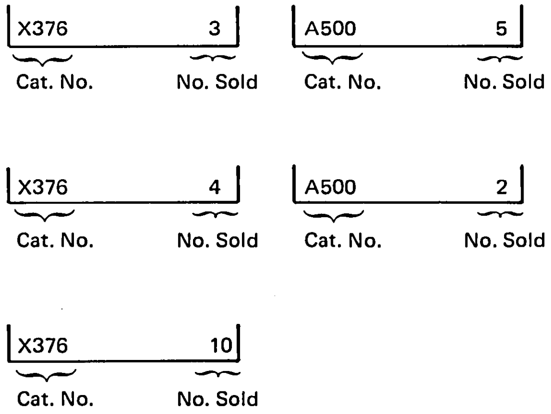
Tag-Along Sort

A tag-along sort allows data fields to "tag-along" with control fields when the records in the file are sorted. These data fields can be only certain fields from the input record or the entire input record. The output for a tag-along sort is a file of sorted records. The sorted records can contain:

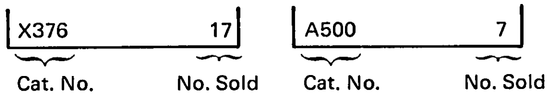
- Control fields and data
- Control fields only
- Data only

Summary Sort

A summary tag-along sort summarizes (adds together) corresponding data fields for those records with identical control fields. The summarizing occurs while the output file is being written. Suppose, for example, that a mail order company wants a sorted file by catalog number of the number of sales for a month. The catalog number is the control field for the record. If a company uses a regular tag-along sort, the sorted file looks like this:



If the company uses a summary sort for the job, all the sales for the same catalog number are summarized and the sorted file looks like this:



The output for a summary sort is a file of sorted records. The sorted records can contain:

- Control fields and summary data
- Summary data only

The output file for a summary sort requires less space than the output file for a tag-along sort because there is only one record for each unique control field.

ADDROUT Sort

Instead of sorting the records, an ADDROUT sort may be performed. An ADDROUT sort produces a file of relative record numbers. The relative record number can be used by an RPG II program to specify the location of a record in the disk file. The record numbers for a file are sorted into the sequence specified by the control fields. These numbers are written on the disk. They can be used as input to an RPG II program that processes the records in the desired sequence.

The ADDROUT sort offers two advantages:

1. The original file is preserved.
2. The work and output areas must only be large enough to provide space for the record numbers, not for the records.

After deciding which file organization method to use, you should design the record and determine file size and location.

DESIGNING A RECORD

The data processing applications that you use when you process a file determine what data is needed in the file's records. You should study these applications and then decide the *layout* of the record. Layout means the arrangement of fields in a record. When you design a record, you must consider processing requirements of the record and then determine field length, location, and name.

To illustrate these design considerations, a name and address file is used in this chapter. Each record in the file contains the following data:

Field	*Size (number of positions)*
Customer Number	6
Name	20
Street Address	20
City and State	20
Record Code	2
Delete Code	1
Other Fields	47
	116 Total

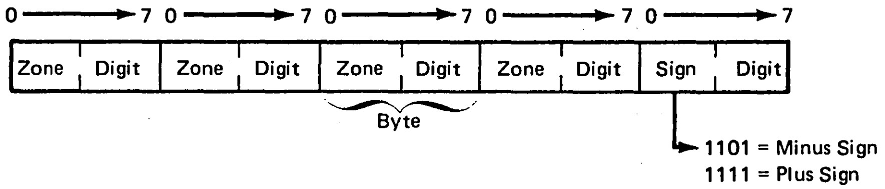
Determining Field Size

Field size depends on the nature of the data in the field. The length of the data may vary, or all data in a field may be the same length. In the example, name is 20 positions. The length of each customer's name varies, but 20 positions should be sufficient for most names. Customer number, however, is six positions, and all six positions are used in each record.

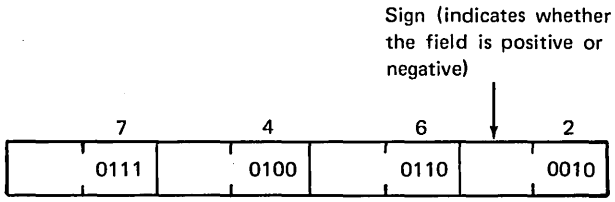
Numeric Fields

If the field is a numeric field, you must determine whether the field is to be in a packed or unpacked decimal format. Pack decimal format can reduce the amount of storage required for a record.

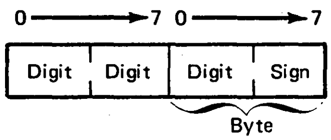
Unpacked decimal format means that each byte of storage, whether on disk or in the computer, can contain one character. (That character may be a decimal number or it may be an alphabetic or special character.) In the unpacked decimal format, each byte of storage is divided into a 4-bit zone portion and a 4-bit digit portion. The unpacked decimal format looks like this:



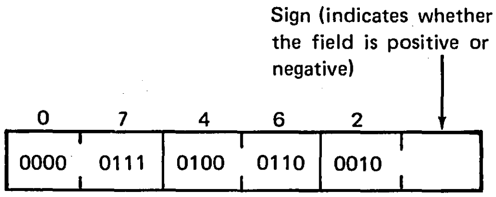
The zone portion of the rightmost byte indicates whether the decimal number is positive or negative. In unpacked decimal format, the zone portion is included for each digit in a decimal number; however, only the zone over the rightmost digit serves as the sign. The unpacked decimal format for decimal number 7,462 looks like this:



Packed decimal format means that a byte of disk storage can contain two decimal numbers. This format allows you to get almost twice as much data into a byte as you can using the unpacked decimal format. In the packed decimal format, each byte of disk storage, except the rightmost byte, is divided into two 4-bit digit portions. The rightmost portion of the rightmost byte contains the sign (plus or minus) for that field. The packed decimal format looks like this:



The sign portion of the rightmost byte is used to indicate whether the numeric value represented in the digit portions is positive or negative. In the packed decimal format, the sign is included for the entire number; the zone portion is not given for each digit in the number. The packed decimal format for decimal number 7,462 looks like this:



The maximum length of a packed field is 15 digits (8 bytes). Figure 20 shows the number of bytes needed for a packed field as compared to the number of bytes needed for an unpacked field.

Unpacked	Packed
1	1
2	2
3	2
4	3
5	3
6	4
7	4
8	5
9	5
10	6
11	6
12	7
13	7
14	8
15	8

Figure 20. Number of Bytes Needed for Packed and Unpacked Fields

Alphameric Fields

There are no firm rules for determining alphameric field size. The major problem involves fields with variable length data. For example, if name is planned as 15 positions and a new customer has 19 characters in his name, a problem arises when adding his record to the file. To avoid this problem, try to estimate the largest length of the data that will be contained in a field. Use this length to determine field size.

Providing for a Delete Code

Recall that records are not automatically deleted. You must place a delete code on a record with your program. Then, when the file is processed, your program must check for this code. In the example, if a customer becomes inactive, you may not want to process his record. Thiis, a 1-position field is included to provide for a delete code.

Providing Extra Space

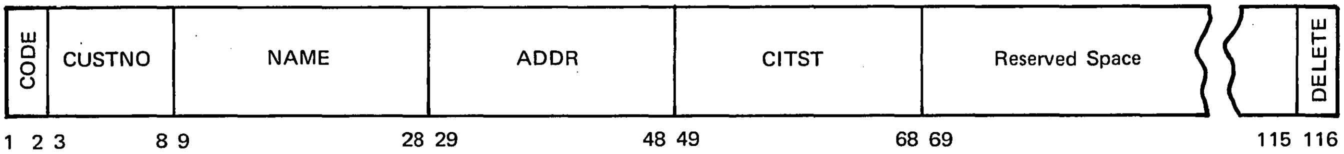
At this stage in planning, it is often desirable to allow for data to be added to a record. For example, suppose the name and address file were created with the fields described, but at a later time each customer's zip code is needed. If all positions in the record are used, there is no place to add the zip code. Since record length is not yet established at the planning stage, we can allow for such additions to this record. Although it is often difficult to imagine what data might be added, it is wise to reserve extra space.

Naming Fields

At the same time you are determining field size and location, you can also decide on names for each field. Since you must specify field names in your source programs, it is a good practice to choose names that follow the coding rules for forming field names. If these rules are considered at this planning stage, your programs are easier to write.

For example, an RPG II field name can be from one to six characters long. The first character must be an alphabetic character, but the remaining characters can be any combination of alphabetic or numeric characters. Blanks and special characters are not allowed. The field names in Figure 21 follow these rules.

One other important consideration when choosing field names is that the name should be meaningful. Since field names may be restricted in length, abbreviations are often necessary. For example, the word *address* has seven letters; it is shortened to ADDR in Figure 21. Meaningful field names contribute to better documentation, and often avoid misinterpretation or confusion while writing programs.



Key

CODE	=	Record code
CUSTNO	=	Customer number
NAME	=	Customer name
ADDR	=	Customer street address
CITST	=	City and state
DELETE	=	Delete code

Figure 21. Layout of Customer Master Record

DOCUMENTING RECORD LAYOUT

When record layouts are documented, your programs are easier to write. Figure 21 shows the layout of a customer master record. A record layout should include the order of the fields in the record, the length of each.field, and the name of each field.

Record Length

A record may contain all pre-defined fields, or space may be reserved for data to be added to the record. In either case, all records in a particular file must be the same length. Record length is the sum of the field lengths (including reserved space).

In our example, the sum of the fields is 116 positions. However, record length is established at 128, thus reserving 12 positions for data that might be needed at a later time.

Block Length

Information about *blocks* may also be required in your programs. A *block* is the number of records that is transferred between a disk file and the processing unit (input) or between the processing unit and a disk file (output). Although only one record at a time is available for processing by your program, one or several records may be transferred at one time. Transferring blocks of records can decrease the time required to perform a job. When records are transferred one at a time, access time is required for the disk access arm to locate each record. When several records are transferred at a time, access time is usually less.

When more than one record is transferred, the records are *blocked*. Transferring blocked records can result in more rapid processing. When only one record is transferred at a time, the records are *unblocked.*

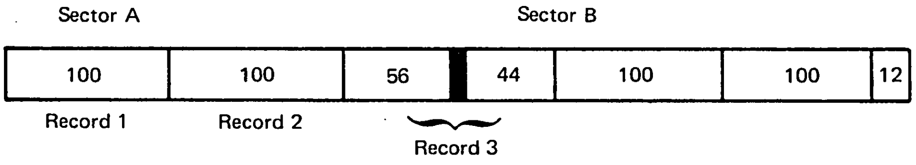
You may want to use unblocked records when a program takes a large amount of storage. In this case, there is a possible increase in total time to do the job, but your program will fit in storage.

Block length is a *multiple* of record length. For example, if your record length is 64, block length could be 256 (64 x 4 = 256). Block length in this case is four times as large as record length. The multiple 4 indicates the number of records you want transferred at one time.

The design of System/3 influences block length. Recall that the smallest division of a disk is a sector, and it can contain up to 256 characters. The system transfers data in sectors, that is, multiples of 256 characters. If your record length is 128, you might have a block length of 256, indicating that you want two records transferred (128 x 2 = 256). Or you might have a block length of 512, indicating that four records are transferred (128 x 4 = 512).

For efficient blocking, you should choose a record length that is a multiple of 256 (256 x 2 = 512) or *submultiple* of 256. A submultiple is a number that divides into 256 a whole number of times. For example, 64 is a submultiple of 256 (256 ÷ 64 = 4).

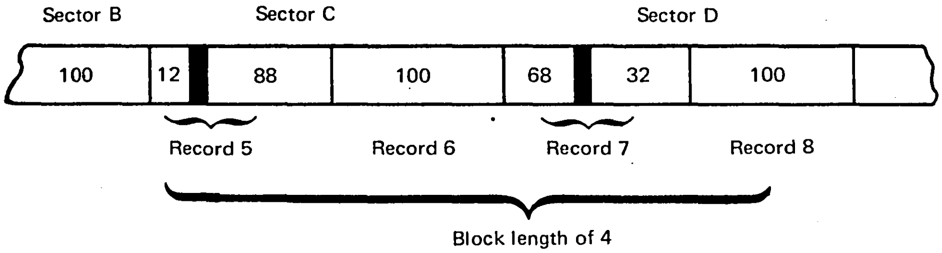
You *can* specify a record length that is not a multiple or submultiple of 256. The system allows you complete flexibility in choosing a record length to fit your application and your disk storage capacity. When you use a record length which is not a multiple or submultiple of 256, no *disk storage* is wasted; some records will simply reside in more than one sector.



However, when you specify 100-character records as shown in the example, the computer requires more *main storage* to process these records.

You recall that the system always transfers data from disk to the computer in increments of sectors. To process record 3, therefore, two sectors must be in main storage, sector A and sector B. The first 56 characters of record 3 reside in sector A; the remaining 44 reside in sector B. Thus, to process 100-character records with a block length of 1 requires that 512 characters (two sectors) be available in main storage.

As another example, suppose you specified 100-character records with a block length of 4. Four 100-character records *can* span three sectors. To process your records in this case requires 768 characters (three sectors) in main storage.



Shared Input/Output Area

Usually an RPG II program uses one input/output (I/O) area for each file. However, if you have a large program that cannot run in the storage available, you may want to use a shared I/O area to reduce the amount of storage needed. A shared I/O area means that all the disk files in the program share a single I/O area. Since a shared I/O area increases the time required to process your program, however, you should not use shared I/O areas unless your program is too large to fit into main storage.

To determine the total I/O area needed when each file has its own I/O area, you find the block length of each I/O area and add them together. Determining the block length is discussed under *Block Length* in this chapter.

Shared I/O does not allow records to be blocked so the block length calculation does not apply. To determine the size of the shared I/O area needed, you find the largest record size in any one disk file used by the program. The I/O area size is then determined as follows:

1. If the record size is 256 bytes, or a submultiple of 256, the I/O area size is 256 bytes.

2. If the record size is a multiple of 256 bytes, the I/O area size is equal to the record size.

3. If the record size is neither a multiple nor a submultiple of 256 bytes, the I/O area size is equal to the record size plus 255 bytes, rounded to the next higher 256-byte increment.

DETERMINING SIZE AND LOCATION OF A DISK FILE

Another aspect of the planning stage is (1) determining how much disk space a file requires and (2) deciding where to locate the file on the disk. These two factors must be considered together since they directly affect each other. For example, two files are already written on a disk on cylinders 8-155. A third file is to be created; it will occupy 55 cylinders. Since a disk contains 200 cylinders, this file is too large to be contained on this disk (155 + 55 = 210). The file must be written on another disk.

Determining Number of Records in a File

Let's first consider the disk space required for a file. To determine this space, you must plan how many records will be in the file at a specified time.

To determine the number of records in a file, you must consider several factors. First, you must know how many records will be in the file when it is created. If the file already exists, perhaps as a card file, use the number of records in this file as a base.

You must also know if records will be added or deleted. If additions are expected, how many records are expected, and how often will they occur? If records will be tagged for deletion, consider periodically removing them from the file. By removing records that you no longer need, you free disk space and allow more records to be added.

Only after considering these factors and the applications that use the file can you determine the number of records in the file. For example, the customer name and address file will contain 6000 records at creation time. It is estimated that each month 200 records will be added and 80 records will be deleted. It is also planned that the deletion records will be removed once a month. At the end of six months the file will contain 6720 records (1200 records are added; 480 records are deleted).

6000	Records at creation
+1200	Records added in six months
7200	
- 480	Records deleted in six months
6720	Records in file after six months

This example points out another factor to consider. When determining the number of records in a file, consider expansion for a reasonable time into the future (at least six months). Of course, most files have deletions, and thus growth is usually slow. In a file where the number of additions and deletions are about the same, deleted records need be removed only when the disk space allowed for the file is filled or when reorganization will improve file access time.

Calculating Record Space

The amount of space required for a file depends upon whether your file organization method is sequential, indexed, or direct. If an indexed file, a sequential file, and a direct file contain the same number of records, the amount of space required for the records in all files is the same. However, additional space is required for the index of an indexed file.

Since the same amount of space is required for the records in any file organization of the same size, record space is calculated in the same way for all files. To determine record space, you must know the number of characters in the file.

To calculate the number of characters in a file, multiply the number of records (allowing for expansion) by the length of each record. For the customer name and address file, there will be 6,720 records in the file at the end of six months. Each record contains 128 characters. Thus, the number of characters in the file is calculated as:

```
     6720        Number of records in the file
     x128        Number of characters in each record
   -------
    53760
   13440
   6720
   -------
  860,160        Total characters in the file
```

You must know how many tracks are needed. Since a track contains 24 sectors, and a sector contains 256 characters, each track can contain 6,144 characters (24 x 256 = 6144). To calculate the number of tracks the file requires, divide the number of characters in the file by 6144. In our example, this calculation is:

```
              140          Tracks
            ---------
  6144   )  860160
            6144
            -----
            24576
            24576
            -----
                0
                0
                -
```

The calculation results in a quotient of 140 and no remainder. So 140 tracks are needed for the name and address file.

When your calculation has a remainder, always add one more track to the quotient. Otherwise, space is not reserved for the last record.

Calculating Index Space

If the file is indexed, you must also determine the amount of space for the index. To find the space needed for the index, you must know the size of the index entry. Recall that an index entry is composed of a key and a disk address. Key lengths vary, depending upon the application, but disk addresses are always three characters long. Thus, the size of an entry is the key field length plus 3.

```
Entry Length  =  Key Field Length  +  3
```

For the name and address file, the key field is customer number (CUSTNO), and it is six characters long. In this case, the index entry length is 9 (6 + 3 = 9).

Another factor affecting index space is sector length. Recall that a sector is the smallest division of a disk and can contain up to 256 characters. For System/3 an index entry must be completely contained within a sector: *an entry cannot start in one sector and end in a different sector.*

To determine the number of entries that can be written in a sector, divide 256 by the entry length. For the name and address example (entry length is 9), this calculation is:

```
                          28         Entries in a Sector
Entry  Length  9     )   256
                          18
                          76
                          72
                           4         Remainder
```

Notice that the division results in a remainder of 4. Thus, 28 entries can be written in one sector. The last four positions of the sector are not used since a complete entry must be written in a sector. The twenty-ninth entry is written in the first nine positions of the next sector.

Remember, when calculating the number of index entries in a sector, *drop the remainder.*

Since index space, like record space, is specified in number of tracks, you must convert the sector space to track space. To do this, you must perform two calculations.

First divide the number of index entries in a sector into the number of records. In our example, this calculation is:

```
                                 240          Sectors
Entries in a  Sector  28   )    6720          Records
                                 56
                                 112
                                 112
                                   0
```

The result of this calculation (240 in this example) specifies how many sectors are needed for the index. This result must then be converted to tracks.

Since there are 24 sectors in a track, to find the number of tracks required, divide the number of sectors by 24.

```
         10          Tracks
24 ) 240
     240
       0
```

In this example, there isn't any remainder. However, if your calculation results in a remainder, add one track to your answer. Otherwise, not enough space will be reserved for the index.

Finally, for an indexed file, add the number of tracks required for the index to the number of tracks required for the records of the file. In our example, the sum is 150 tracks.

```
140 (records) + 10 (index) = 150
```

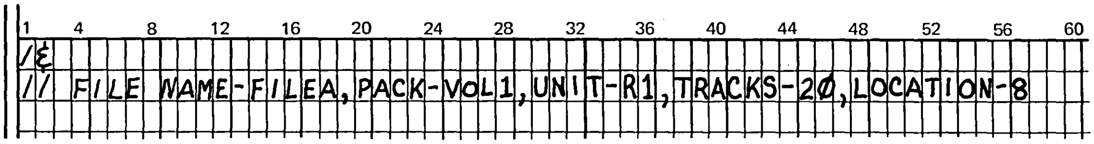
Deciding Where to Locate the File

After you determine the amount of space the file requires, you can decide where to locate the file on the disk. Since a disk can contain several files, depending upon their size, it is good practice to document what files are on which disk.

The Disk File Layout Chart (Figure 22) is available for this purpose. The Disk Layout Chart shows space available on the fixed and removable disks. There are 406 positions (0-405), represented on the chart. Each position corresponds to a track. In Figure 22, notice that tracks 0 through 7 have a line through them. These tracks are reserved for system use only and are not available for data files.

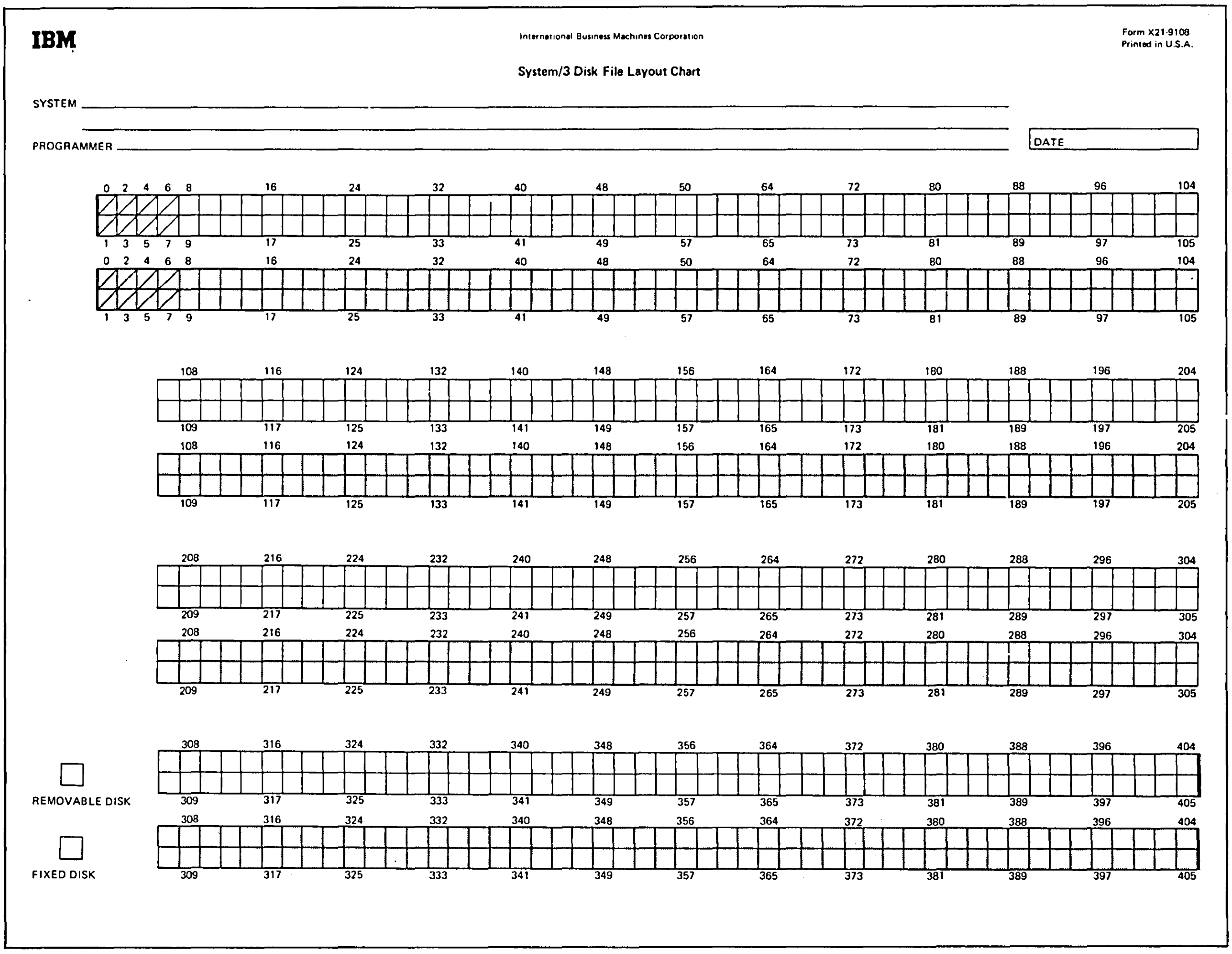
As you create more files, you can refer to the chart of a particular disk to determine the amount of available space on that disk. It is helpful then to indicate the required space for each file on a Disk Layout Chart. It is also helpful to indicate the name of the file on the chart. Figure 23 shows the space and location of the name and address file using the indexed method. The calculations performed to determine the amount of disk space can be documented on the back of the chart.

After you have determined where to place your file, you can code the LOCATION parameter of the FILE statement to tell disk system management on which track the file is to begin. This sample FILE statement contains a LOCATION parameter to tell disk system management that FILEA is to be located on disk pack VOL1 beginning on track 8:



1
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60

```
/&
// FILE NAME-FILEA,PACK-VOL1,UNIT-R1,TRACKS-2Ø,LOCATION-8
```



IBM

International Business Machines Corporation

Form X21-9108
Printed in U.S.A.

System/3 Disk File Layout Chart

SYSTEM ____________________

PROGRAMMER ____________________

DATE

0 2 4 6 8 16 24 32 40 48 50 64 72 80 88 96 104
1 3 5 7 9 17 25 33 41 49 57 65 73 81 89 97 105
0 2 4 6 8 16 24 32 40 48 50 64 72 80 88 96 104
1 3 5 7 9 17 25 33 41 49 57 65 73 81 89 97 105

108 116 124 132 140 148 156 164 172 180 188 196 204
109 117 125 133 141 149 157 165 173 181 189 197 205
108 116 124 132 140 148 156 164 172 180 188 196 204
109 117 125 133 141 149 157 165 173 181 189 197 205

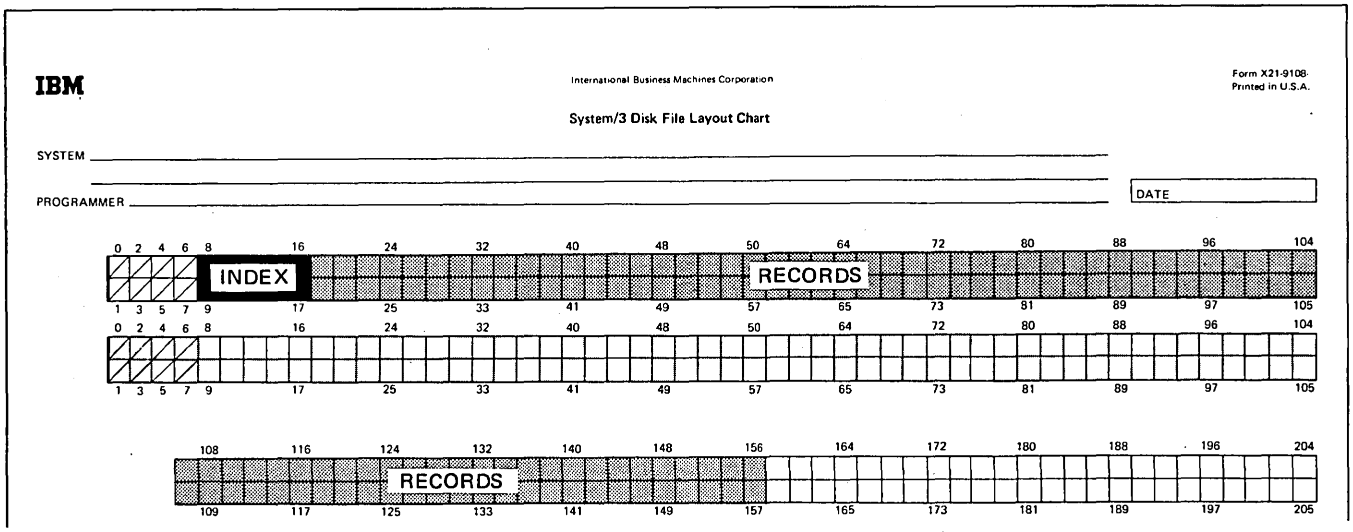
208 216 224 232 240 248 256 264 272 280 288 296 304
209 217 225 233 241 249 257 265 273 281 289 297 305
208 216 224 232 240 248 256 264 272 280 288 296 304
209 217 225 233 241 249 257 265 273 281 289 297 305

□ REMOVABLE DISK

□ FIXED DISK

308 316 324 332 340 348 356 364 372 380 388 396 404
309 317 325 333 341 349 357 365 373 381 389 397 405
308 316 324 332 340 348 356 364 372 380 388 396 404
309 317 325 333 341 349 357 365 373 381 389 397 405

Figure 22. Disk File Layout Chart



IBM

International Business Machines Corporation

Form X21-9108
Printed in U.S.A.

System/3 Disk File Layout Chart

SYSTEM ____________________

PROGRAMMER ____________________

DATE

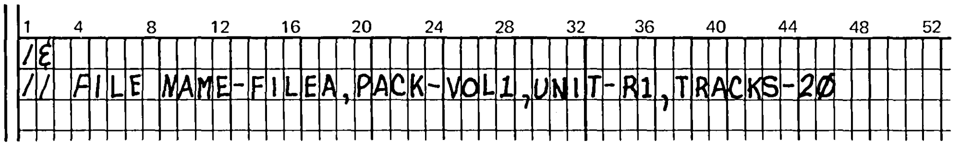
0 2 4 6 8 16 24 32 40 48 50 64 72 80 88 96 104
1 3 5 7 9 17 25 33 41 49 57 65 73 81 89 97 105
0 2 4 6 8 16 24 32 40 48 50 64 72 80 88 96 104
1 3 5 7 9 17 25 33 41 49 57 65 73 81 89 97 105

108 116 124 132 140 148 156 164 172 180 188 196 204
109 117 125 133 141 149 157 165 173 181 189 197 205

Figure 23. Disk Layout for an Indexed File

Automatic File Allocation

If you do not specify the LOCATION parameter on the FILE statement, FILEA is located on the disk pack automatically for you.



```
/ξ
// FILE MAME-FILEA,PACK-VOL1,UNIT-R1,TRACKS-2Ø
```

The process used by disk system management to allocate file space for you is known as automatic file allocation.

When allocating file space, disk system management calculates the length of the file and checks the volume label to determine which tracks are available for allocation. (The volume label contains the status of each track and indicates which tracks are available for allocation.) Disk system management then:

1. Finds a continuous string of available tracks.

2. Allocates space for permanent files, then temporary files, and finally scratch files, if multiple files are being allocated.

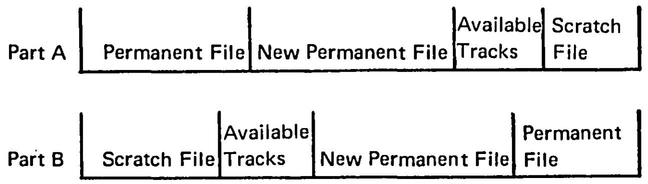
Disk system management places your file on the smallest continuous string of available tracks that can contain your file. For example, it can determine that your file is 10 tracks long and find one string of 12 available tracks and another of 15 tracks. It places your file in the string of 12 tracks because the 12-track string is closer to the length of the file.

If disk system management finds two strings and both have the same number of available tracks, the file is placed at the highest numbered available location. Also, if your file is the first file placed on a disk, the system allocates space for the file beginning at the highest numbered track. The system allocates space beginning at the highest location to allow you as many available tracks as possible next to the object library (the object library is located at the lowest numbered tracks) so the object library can expand if necessary.

If an area is found containing the same number of available tracks and two files are already on either side of the area, disk system management determines the type of file to the left of the available track. If the file to the left has similar attributes, the new file is left-adjusted; if the file to the left is not similar, the new file is right-adjusted (Figure 24). Files are placed adjacent to files with similar attributes, so there will be as few unused tracks between files as possible. It is more important, however, to locate a new file on a string of tracks as close to the length of your file as possible. Therefore, a permanent file could be allocated space next to a temporary or scratch file if the number of tracks at that location can contain the permanent file.

Considerations for Using Automatic File Allocation

It is easier to let disk system management allocate file space because you do not have to determine where to locate files. It is easier, but there are some considerations in determining whether to use automatic file allocation. After you have gained experience, you should be able to locate a file more efficiently than disk system management. Disk system management may leave a string of available tracks between files which is unusable because the string is not long enough to contain another file.



Disk system management determines the type of file to the left of the available tracks. If the file to the left is similar, the new file is left-adjusted (Part A). If the file to the left is not similar, it is right-adjusted (Part B).

Figure 24. File Placement of Automatic File Allocation

If you plan your own files, you can determine where files are located by checking the Disk File Layout Chart, if you keep your layout chart up-to-date. If you automatically allocate some files and then want to locate a file yourself, however, you must check the volume label to determine what tracks are available. This can be done by using the File and Volume Label Display utility program. (See the *IBM System/3 Disk System Operation Control Language and Disk Utilities Reference Manual,* GC21-7512, or the *IBM System/3 Model 6 Operation Control Language and Disk Utility Programs Reference Manual,* GC21-7516, for more information on this utility program.)

Automatic file allocation can increase the time needed to copy programs using the Disk Copy/Dump utility program. (See the appropriate disk utilities reference manual previously referenced for more information on this utility program.) For example, you have used automatic file allocation and now wish to copy a file onto tracks 30 through 50 of the disk on F1. However, disk system management placed the file to be copied on tracks 50 through 70 of the disk R1. Copying time increases when a file is copied from one location on a disk to another location on another disk, because the access mechanism must move. It would be more advantageous to allocate the file space on tracks 30 through 50 of R1 yourself so that the file can be copied onto the same tracks (tracks 30 through 50) of F1.

Automatic file allocation considers effective use of file space, but not the usage of the files. It does not consider file planning for multiple input files in a program or job-to-job transitions. If you plan your own file locations, you can place files that are used together near one another on disk. When files used together are located near one another, processing time may be improved.

DATA FILE SECURITY

Once you have stored your data files on disk, you will want to ensure that the files are not accidentally destroyed. For instance, a wrong disk pack could be mounted, a wrong program could be loaded, or a valid data file could be written over. To avoid this problem, file labels and volume labels are used to provide file protection.

Every data file stored on disk is protected by a file label containing file characteristics. Some typical fields in the file label are the filename, creation date, retention status of the file, and file type. A file cannot be accessed or changed without first checking the file label.

The volume label defines the characteristics of the volume. Some typical fields in the volume label are the volume serial number, owner identification, and available tracks.

In order to use a particular disk file required in a program, the operator must provide the necessary specifications by means of OCL statements. The system uses the information given in these statements to verify that the correct pack is mounted and that the required disk file or disk area is available.

Chapter 10. Storing Programs and Procedures on Disk

In the IBM System/3 Model 6 and Model 10 Disk Systems, programs and OCL statements can be stored on disk and transferred as needed into main storage.

The area in which programs are stored on disk is called a library. Two types of libraries can be located on a disk: *object libraries* and *source libraries.* Object libraries contain object programs; source libraries contain source programs, OCL statements, and utility program control statements.

When OCL statements and utility program control statements are stored in a source library, they are called *procedures.*

The System/3 Library Maintenance program can be used to:

- Allocate space for libraries.
- Enter programs and procedures into libraries.
- Maintain libraries.

More information about this program and its functions is given later in this chapter under *Library Maintenance Program.*

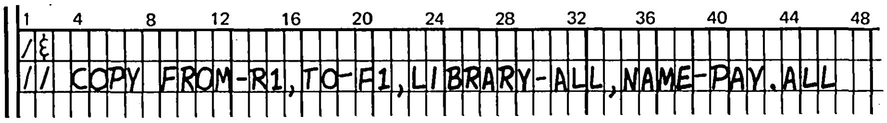
ADVANTAGES OF STORING PROGRAMS AND PROCEDURES ON DISK

Increasing System Efficiency

All programs and procedures can be placed on a master pack and copied to the fixed disk for execution. For example, you can load an entire series of application programs and procedures on a fixed disk. Once your programs and procedures are located on disk, programs can be transferred quickly into main storage, thereby decreasing the amount of time to run your jobs. Assume you run payroll every Friday morning. On Friday, you can use a pretested procedure to transfer all the required programs and their procedures from the master pack to a fixed disk, then run payroll.

There are two library functions that make this method particularly efficient: naming conventions and object library expansion.

Naming Conventions: If you establish and use a naming convention, you can transfer all the correct programs and procedures from the master pack to the fixed disk using one Library Maintenance control statement. The names of all programs and procedures used in an application series should begin with the same letters. For example, you might name all payroll programs and their corresponding procedures beginning with the letters PAY. Then, with one COPY control statement, all payroll programs and procedures in both libraries will be copied onto the fixed disk.



```
1  4   8   12  16  20  24  28  32  36  40  44  48
/&
// COPY FROM-R1,TO-F1,LIBRARY-ALL,NAME-PAY.ALL
```

(The COPY control statement is described under *Library Maintenance Program.)*

Object Library Expansion: Object libraries are capable of expanding their size for temporary entries. When you copy an object program onto the object library on fixed disk, you can designate it as a temporary entry. Then if you add a permanent entry, reallocate the library, or delete all temporary entries, the object library will return to its normal size. Consequently, by using this function, you use a minimum amount of storage on the fixed disk, leaving it free to perform other functions when you are not using the object library.

Storing Programs and Their Data Files on Removable Disks

If space on the fixed disk is limited, or if you prefer to do so, you can store programs and data files on a removable disk. By placing programs and data files on the same removable disk, you can reduce the number of times disk packs must be changed. This is especially true if a program uses only one data file. This also provides more available space on the fixed disk.

There are certain things you must consider when placing both programs and data files on a removable disk, however. First, additional space is required on the removable disk.

Maintaining programs on a removable disk is more difficult, because they are scattered across several disks instead of all located on a master pack. For example, if the format of an inventory record changed, you might be required to search several packs to update all the programs using that record, rather than searching just one master pack. You should have a master pack so that you have copies of your programs if something happens to one of the other disks.

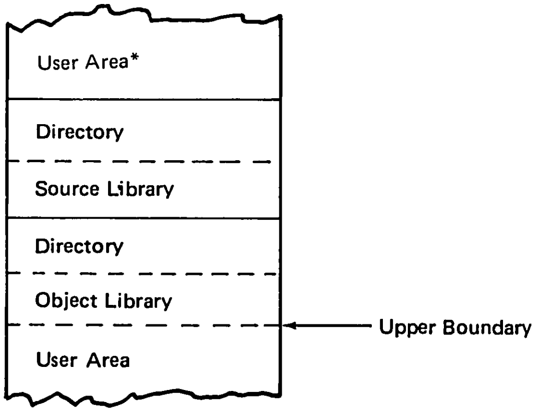
You should not place data and programs on the same packs if your are processing multivolume files. The pack containing the program cannot be removed during the job.

LOCATIONS OF LIBRARIES ON DISK

You can place a source library, an object library, or both on a disk. If space is allocated for only a source or object library, the Library Maintenance program places the library in the first available disk area large enough to contain the library.

If you are allocating space for a source library on a disk containing an object library, a disk area large enough for the source library must immediately precede or follow the object library (Figure 25). If the disk area follows the object library, the Library Maintenance program moves the object library to allow space for the source library preceding it.

If an object library is being allocated on a disk with a source library, space for the object library must immediately follow the source library.



* If there are no user files present at the time the library is created, this area contains alternate tracks.

Figure 25. Relative Positions of Libraries on Disk

SOURCE LIBRARIES

Source libraries can contain source program statements and procedures. Examples of source statements are RPG II source programs and sequence specifications for the Disk Sort program.

Procedures are sets of OCL statements. The procedures for utility programs can include program control statements.

Entries in the source library can be comprised of any valid System/3 characters. Figure 26 shows the format of the source library.

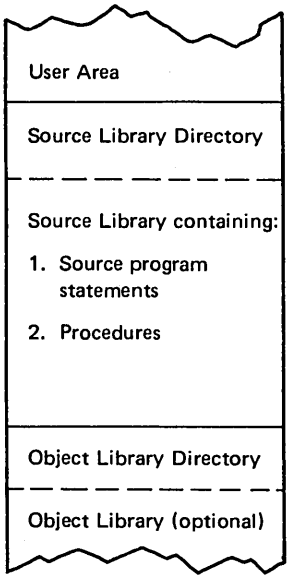
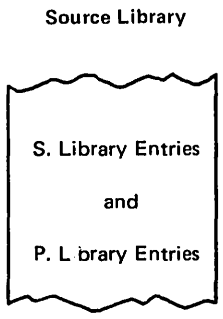

User Area
Source Library Directory
Source Library containing:
1. Source program statements
2. Procedures
Object Library Directory
Object Library (optional)

Figure 26. Format of the Source Library

The source library is one physical area containing two logically different types of entries. When these entries are copied into source libraries, they are given different source library designations. Source programs are given an S library designation; procedures are given a P library designation. Figure 27 shows the logical entries within the source library.



The *S* library entries are source programs. Procedures *cannot* be executed from this library.

The *P* library entries are procedures which can be executed.

Figure 27. Logical Entries within the Source Library

Physical Characteristics of the Source Library

Size: The minimum size of a source library is one track.

Directory: Note the area labeled source library directory in Figure 26. The directory acts as a table of contents and contains the name and location of each source library entry. The first two sectors of the first track are always assigned to the directory with additional sectors used as needed.

Organization of Entries: Entries within the source library need not be stored in consecutive sectors. An entry can be stored in widely separated sectors with each sector containing a pointer to the next sector that contains the next part of the entry.

The boundary of the source library cannot be expanded; therefore, an entry must fit within the available library space. The system provides maximum space within the prescribed limits of the source library by compressing entries. That is, all duplicate characters are removed from entries. Later, if the entries are printed or punched, the duplicate characters are reinserted.

OBJECT LIBRARIES

The object library is an area on disk used to store object programs and routines. Object programs, or executable programs, are programs and subroutines that can be loaded for execution. Routines, or nonexecutable programs, are programs and subroutines that need further translation before being loaded for execution. Nonexecutable programs are used by a compiler and must be on the same disk pack as the compiler. Figure 28 is a sample object library.

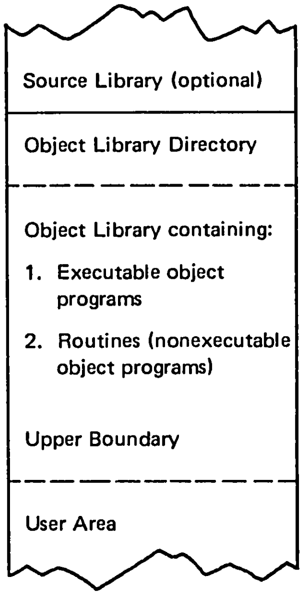
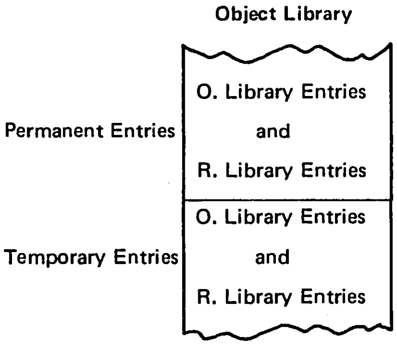

Source Library (optional)
Object Library Directory
Object Library containing:
1. Executable object programs
2. Routines (nonexecutable object programs)
Upper Boundary
User Area

Figure 28. Format of the Object Library

The object library is one physical area containing two logically different types of entries: object programs and routines. When these entries are copied into the object library, they are given different object library designations. Object programs are given an O library designation; routines are given an R library designation. Figure 29 shows the logical library entries within the object library.



The *O* library entries are executable programs. They are loaded by the LOAD statement.

The *R* library entries are nonexecutable routines used by the compiler. They must be on the same disk as the compiler.

Figure 29. Logical Parts of an Object Library

Physical Characteristics of the Object Library

Size: The size of the object library depends on whether or not the library is on a system pack (a disk pack containing the system programs). You can build an object library on any disk pack, but you must have one library online containing the system programs. The minimum size of an object library on a system pack is 30 tracks. The minimum size of a library not on a system pack is three tracks.

The disk area for the object library consisting of system programs must also be large enough to contain a work area for disk system management. The number of tracks for the work area space need not be included in the number of tracks you specify for the library; the Library Maintenance program calculates and assigns the additional space for you. The amount of additional space needed depends on the capacity of your system and whether your programming system has inquiry capability or the dual programming feature. Figure 30 is a table showing the work area size required for various system capacities.

Scheduler Work Area Size

Capacity	No Inquiry without DPF [1]	No Inquiry with DPF	Inquiry without DPF	Inquiry with DPF	Roll-In/ Roll-Out [2]
8K bytes (Model 6 only)	2 tracks	——	6 tracks	——	3 tracks
12K bytes	2 tracks	4 tracks	6 tracks	8 tracks	4 tracks
16K bytes	2 tracks	4 tracks	7 tracks	9 tracks	5 tracks
24K bytes	2 tracks	4 tracks	8 tracks	10 tracks	7 tracks
32K bytes	2 tracks	4 tracks	9 tracks	11 tracks	9 tracks

[1] Dual Programming Feature.

[2] Tracks needed by the scheduler to retain information concerning an interrupted program.

Figure 30. Work Area Size

Directory: The Library Maintenance program creates a directory for every object library (Figure 28). The directory acts as a table of contents and contains the name and location of the object library entries. If the object library is on a system pack, three of the requested tracks are reserved for the directory. If not, only the first track is reserved for the directory.

Upper Boundary: The upper boundary of the object library (Figure 28) will automatically expand only if more space is needed for temporary entries and if the area next to the library is available. When permanent entries are placed in the library, all the temporary entries are deleted and the object library returns to its normal size.

To make efficient use of this feature, the area next to the upper boundary of the object library should be kept free of data files. When disk system management automatically allocates file space for you, the area next to the object library is probably free because your files are placed as close to the end of the disk pack as possible. When allocating your own file space, you should allocate your files toward the end of the pack also. This leaves room for object library expansion.

Organization of Entries: Entries are stored in the object library serially; that is, a 20-sector program occupies 20 consecutive sectors. Temporary entries follow all permanent entries in the object library. A new permanent entry is loaded into the first available space large enough to hold it, usually the space following the last permanent entry.

Gaps can occur in the object library when a permanent entry is deleted and replaced with one using fewer sectors. The Library Maintenance program scans the library to locate available sectors, then places the entry into the smallest gap large enough to hold it.

You should use the Library Maintenance program to reorganize the library when you delete permanent entries, when a great number of additions and deletions take place, or when there is no apparent room.

In reorganizing the library, the Library Maintenance program shifts entries so that gaps do not appear between them. This makes more sectors available for use.

Frequent adding, replacing, and deleting of entries causes unusable sectors. You can determine how many sectors are unusable by printing the library directory using the Library Maintenance program.

STORING PROGRAMS AND PROCEDURES INTO LIBRARIES

There are three methods you can use to store programs into libraries: the Library Maintenance program, a specification on the RPG II Control Card sheet, or the COMPILE OCL statement.

Library Maintenance Program

Depending upon your specifications, the Library Maintenance program can:

- Allocate space for a library. It can create, reorganize, change the size of, or delete a library.
- Delete entries from a library.
- Copy entries from one location to another within a library, from one library to another, from the input device to a library, from the library to a printer, or from a library to a punch, and give new names if requested.
- Rename library entries.

For information on the specifications necessary to perform these functions, refer to the *IBM System/3 Disk System Operation Control Language and Disk Utilities Reference Manual,* GC21-7512, or the *IBM System/3 Model 6 Operation Control Language and Disk Utility Programs Reference Manual,* GC21-7516, depending on the system you are using.

RPG II Control Card

You can use RPG II to indicate the type of object output you want after compiling a source program. The compiled program can be stored in an object library or punched into cards. You usually want the object program written in the object library until you have corrected the severe errors in your program. When a program is written temporarily in the object library, it is overlaid by the next program written in that object library. The object library is written in the same object library containing the compiler, unless a COMPILE statement indicates otherwise.

Column 10 on the RPG II Control Card sheet is used to specify the object output. Columns 75-80 are used to name your object program. For detailed information on the specifications you should make in these columns, see the *IBM System/3 Disk System RPG II Reference Manual,* SC21-7504, or the *IBM System/3 Model 6 RPG II Reference Manual,* SC21-7517, depending on the system you are using.

COMPILE OCL Statement

The COMPILE OCL statement tells disk system management to:

1. Compile a source program from a source library and store the object program in an object library, or

2. Compile a source program from cards and store the object program in an object library.

For a detailed description of the COMPILE statement, refer to the *IBM System/3 Disk System Operation Control Language and Disk Utilities Reference Manual,* GC21-7512, or the *IBM System/3 Model 6 Operation Control Language and Disk Utility Programs Reference Manual,* GC21-7516, depending on the system you are using.

Appendix. IBM 5445 Disk Storage Drive

GENERAL DESCRIPTION

The IBM 5445 Disk Storage Drive can be attached to the IBM System/3 Model 10 Disk System to provide additional storage capacity. No libraries can reside on the 5445; it is only for data storage. The disk pack for the 5445 contains 11 disks. The upper surface of the top disk and the lower surface of the bottom disk are unused. There are, therefore, 20 usable surfaces. The disk pack is removable.

The access mechanism contains 20 read/write heads for the usable disk surfaces. This mechanism moves back and forth across the disk surfaces to position the heads to read or write data. When the access mechanism is in any one position, all 20 heads are positioned in the same relative location on the 20 disk surfaces (Figure 31).

Each surface of each disk contains 200 tracks. Tracks are divided into 20 sectors; each sector has a unique address. Each sector contains 256 characters of data.

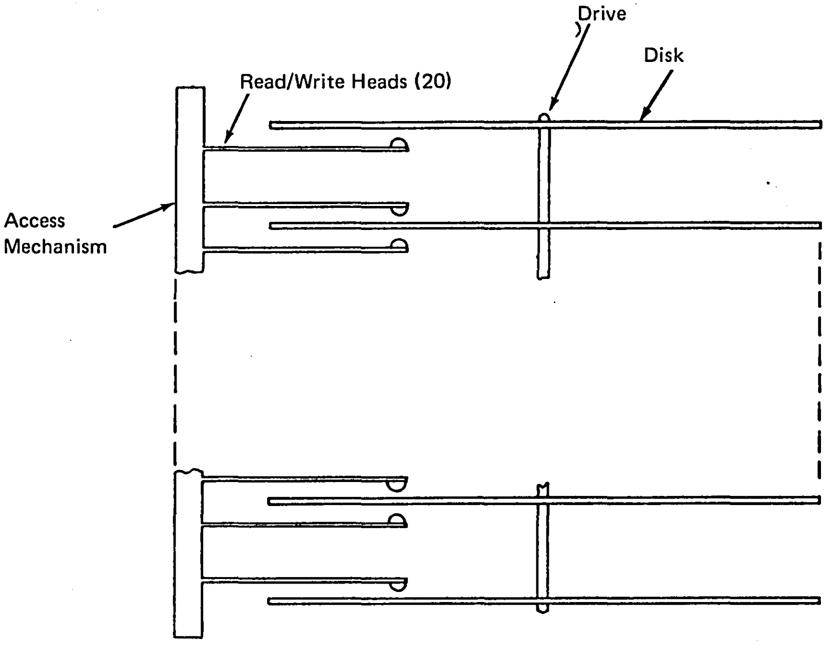


Figure 31. IBM 5445 Disk Storage Drive

A 5445 cylinder consists of all the tracks on a disk pack in one vertical plane (Figure 32). Since 20 disk surfaces can be accessed, a cylinder is made up of 20 tracks. The same cylinder address is used for all corresponding tracks on the disk.

The 5445 has a split cylinder capability. This means that two or more files can be arranged on two or more cylinders with each file occupying a corresponding part of each cylinder. For example, you may allocate File A on tracks 0-3 of cylinders 3-5 and File B on tracks 4-7 on cylinders 3-5. The advantage of the split cylinder capability is that you can arrange your files in combinations to decrease the access time required. For instance, the first file on the cylinder could be a master file and the remaining tracks on the cylinder could be reserved for files associated with the master file.

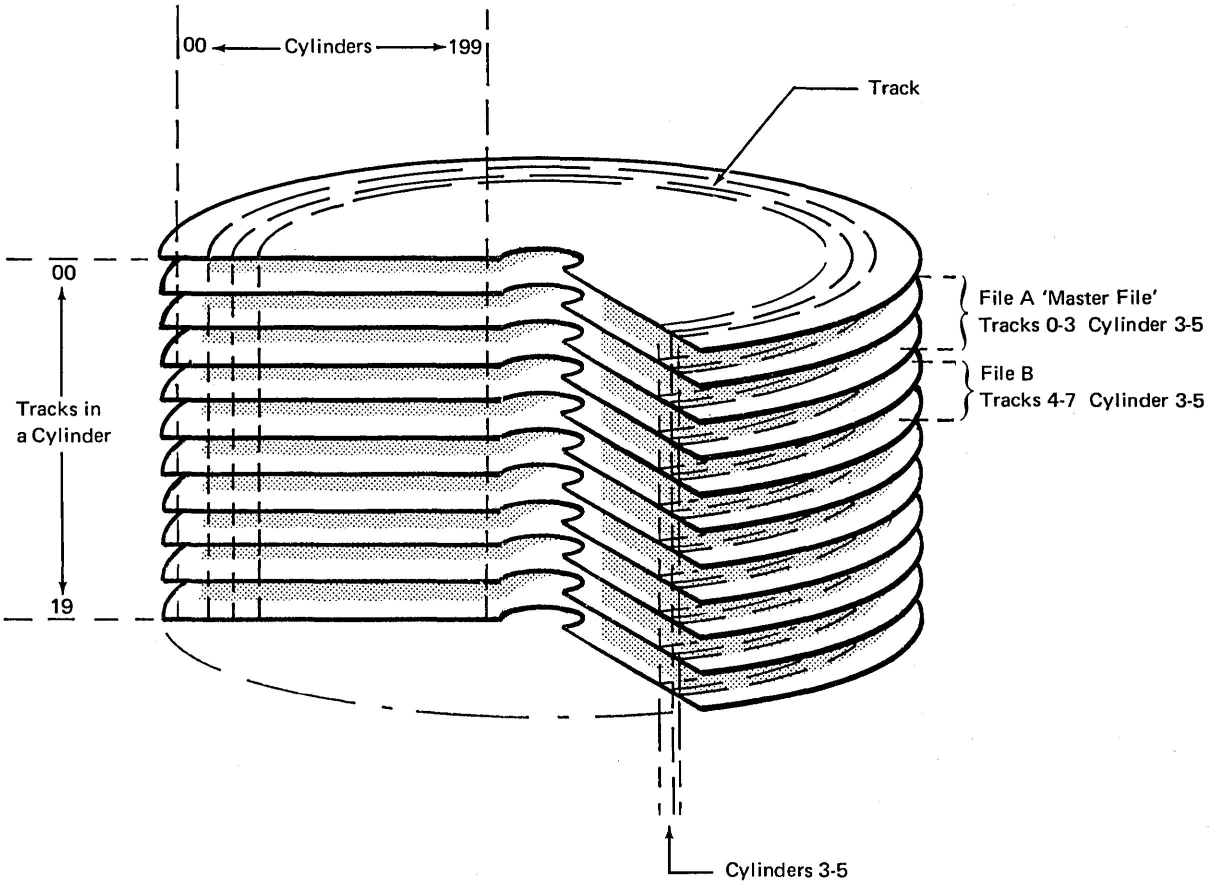

00 ← Cylinders → 199
Track
00
Tracks in a Cylinder
19
File A 'Master File' Tracks 0-3 Cylinder 3-5
File B Tracks 4-7 Cylinder 3-5
Cylinders 3-5

Figure 32. Cylinder Concept on the IBM 5445

FILE PLANNING

The file planning information discussed in Part III of this manual is basically the same for both the IBM 5444 and the IBM 5445. The calculations for determining the size of a disk file (Chapter 9) are different, however, because the 5445 only has 20 sectors per track as compared to 24 sectors per track for the 5444. Also, for an indexed file, the disk address in the index entry is four characters instead of three.

Calculating Record Space

To store your file on disk, you must determine how many tracks will be needed for that file. Since a track on the 5445 contains 20 sectors and a sector contains 256 characters, each track can contain 5,120 characters (20 x 256 = 5120). To calculate the number of tracks the file requires, divide the number of characters in the file by 5120. In the example discussed in Chapter 9, the file contained 6,720 records and each record contained 128 characters. Thus, the number of characters in the file was 860,160. To find the number of tracks this file would require on the 5445, the calculation is:

```
              168      Tracks
 5120 ) 860160
        5120
        34816
        30720
         40960
         40960
```

The calculation results in a quotient of 168 and no remainder. So 168 tracks are needed for the file. When your calculation does have a remainder, always add one more track to the quotient. Otherwise, space is not reserved for the last record.

Calculating Index Space

If your file is indexed, you must also determine the amount of space for the index. Index space is also specified in number of tracks. To find the space needed for the index, you must first find the size of the index entry. The 5445 differs from the 5444 in that the disk address of the index entry for the 5445 is always four characters long. Thus, the size of the entry is the key field length plus 4.

Entry Length = Key Field Length + 4

Thus, if you have a key field, such as a customer number, that is six characters long, the index entry is 10 (6 + 4 = 10).

Next you must determine the number of entries that can be written in a sector. To do this, divide 256 by the entry length. Thus, if the entry length is 10, this calculation is:

```
                         25      Entries in a Sector
Entry Length     10 ) 256
                        20
                         56
                         50
                          6      Remainder
```

The division results in a remainder of 6. Thus, 25 entries can be written in one sector. The last six positions of the sector are not used since a complete entry must be written in a sector. The twenty-sixth entry will be written in the first ten positions of the next sector.

Now you must convert the sector space to track space. To do this, you must perform two calculations. First divide the number of index entries in a sector into the number of records. In our example, this calculation is:

```
                               268      = 269 Sectors
Entries in a Sector     25 ) 6720            Records
                             50
                             172
                             150
                              220
                              200
                               20            Remainder
```

Since this calculation has a remainder, one sector should be added to your answer so that enough sectors will be reserved for all the index entries. Thus, 269 sectors are needed for the index in this example. This result must then be converted to tracks.

There are 20 sectors in a track, so to find the number of tracks required, divide the number of sectors by 20.

```
          13      = 14 Tracks
   20 ) 269
        20
         69
         60
          9          Remainder
```

In this example, there is a remainder of 9. Therefore, you should add one track to your answer. Otherwise, not enough space will be reserved for the index. For this example, 14 tracks are needed for the index.

Index

ƀ–type program 33

activity 51
adding records
  direct file 24
  indexed file 12
  sequential file 6
ADDROUT files
  considerations when using 25
  creating 25
  processing by 26
ADDROUT sort 53
alphameric fields 57
automatic file allocation
  considerations when using 66
  description 66

B–type program 33
batch processing 51
block length 59
blocked records 59

COMPILE OCL statement 76
consecutive processing
  direct file 22
  sequential file 6
conversion method for deriving relative record numbers 17
creating files
  ADDROUT file 25
  direct file
    description 21
    with synonyms 21
    without synonyms 21
  indexed file 9
  with record key limits 27
  sequential file 5
cylinder
  split 78
  5444 4
  5445 78

data file security 67
dedicated system 34
delete code, providing for 57
deleting records (*see* reorganizing files; tagging records for deletion)
designing records 55
direct files
  adding records 24
  creating 21
  description 15
  processing 22
  tagging records for deletion 24
  updating records 24
direct method of deriving relative record numbers 16
Disk File Layout Chart 64
disk storage drive
  5444 3
  5445 77
DPF (*see* dual programming feature)
dual programming feature
  considerations when running System/3 programs
    Alternate Track Assignment 42
    Disk Initialization 42
    Disk Sort 42
    Inquiry 42
  considerations when using 39
  description 37
  executing an RPG II object program 43
  input/output devices 39
  loading programs 43
  main storage 38
  processing time 38

field names, rules for 58
field size
  alphameric fields 57
  description 55
  numeric fields 55
file activity 51
file allocation
  automatic 66
  specifying 64
file containing record key limits
  creating 27
  processing 27
file location 64
file organization selection
  file activity 51
  file size 51
  file use 47
  file volatility 50
file planning
  5444 55
  5445 79
file records, determining number of 61
file security 67
file size
  determining 60
  multivolume files 51
  sorting files 52
file space, calculating 61
file storage on removable disks 70
file use 47
file volatility 50

home location 19
home record 19

I/O area, shared 60
I–type program 33
IBM 5406 Processing Unit keyboard console 31
IBM 5444 Disk Storage Drive 3
IBM 5445 Disk Storage Drive 77
IBM 5471 Printer–Keyboard 31
index 9
index space, calculating
  5444 62
  5445 79
indexed files
  adding records 12
  creating 9
  description 9
  processing 10
  reorganizing 13
  tagging records for deletion 13
  updating records 13
input/output area, shared 60
inquiry program 31
Inquiry Request switch 32
interrupt environment 34

keyboard console, 5406 Processing Unit 31

libraries
  description 69
  functions
    naming conventions 70
    object library expansion 70
  locations on disk 71
Library Maintenance program 75
location of the files 64
locations of libraries on disk 71

multivolume files
  description 51
  offline 51
  online 52

naming conventions of a library 70
naming fields 58
number of file records, determining 61
numeric fields 55

object libraries
  description 73
  directory 74
  organization of entries 75
  size 74
  upper boundary 74
object library expansion 70
offline multivolume files 51
online multivolume files 52

printer–keyboard, 5471 31
procedures
  description 69
  storage on disk 69
  storage into libraries 75
processing files
  ADDROUT file 26
  direct file
    consecutive 22
    random by ADDROUT file 23,26
    random by relative record number 22
  indexed file
    random by key 11
    sequential by key 10
    sequential within limits 11,27
  sequential file
    consecutive 6
    random by relative record number 6,22
program storage
  on disk 69
  into libraries 75
  on removable disks 70

random processing
  direct file
    by ADDROUT file 23,26
    by relative record number 22
  indexed file 11
  sequential file 6,22
random updating of an indexed file 13
record address files
  files containing record key limits 27
  files containing relative record numbers 25
record design 55
record layout, documenting 58
record length 58
record space, calculating
  5444 61
  5445 79
records in a file, determining number of 61
relative record number
  deriving
    conversion method 17
    direct method 16
  description 15
removable disks, storing programs and files 70
reorganizing files
  indexed file 13
  sequential file 7
REQ key 31

sector
  5444 4
  5445 77
security, data file 67
sequential files
  adding records 6
  creating 5
  description 5
  processing 6
  reorganizing 7
  tagging records for deletion 6
  updating records 7
sequential within limits (*see* file containing record key limits)
sequential processing of an indexed file
  by key 10
  within limits 11
sequential updating of an indexed file
  by key 13
  within limits 13

shared input/output area 60
size of file, determining 60
source libraries
  description 71
  directory 72
  organization of entries 72
  size 72
split cylinder 78
storage capacity, 5444 4
storing programs and files on removable disks 70
storing programs and procedures
  on disk 69
  into libraries 75
summary sort 53
synonym records 19
system control panel on the Model 6 32
system efficiency, increasing 69

tag-along sort 52
tagging records for deletion
  direct file 24
  indexed file 13
  sequential file 6
track
  5444 4
  5445 77

unblocked records 59
updating records
  direct file 24
  indexed file
    random by key 13
    sequential by key 13
    sequential within limits 13
  sequential file 7

volatility 50

5406 Processing Unit, keyboard console 31
5444 Disk Storage Drive 3
5445 Disk Storage Drive 77
5471 Printer–Keyboard 31

READER'S COMMENT FORM

IBM System/3
Disk Concepts and Planning Guide

GC21-7571-0

YOUR COMMENTS, PLEASE...

Your comments concerning this publication will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

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

YOUR COMMENTS, PLEASE...

Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Cut Along Line

Fold   Fold

FIRST CLASS
PERMIT NO. 387
ROCHESTER, MINN.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Development Laboratory
Rochester, Minnesota 55901

Attention: Publications, Dept. 245

Fold   Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

GC21-7571-0

IBM System/3 Printed in U.S.A. GC21-7571-0

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)