IBM System/3
Disk Concepts and Planning Guide

GC21-7571-3
File No. S3-20

IBM System/3
Disk Concepts and Planning Guide

Fourth Edition (December 1975)

This is a reprint of GC21-7571-2, incorporating technical newsletter GN21-5293 dated 17 April 1975.

Information concerning inquiry and dual programming for the Model 10 Disk System has been removed from this manual and can now be found in the *IBM System/3 Disk System Control Programming Reference Manual,* GC21-7512; information concerning inquiry for Model 6 can now be found in the *IBM System/3 Model 6 Operation Control Language and Disk Utility Programs Reference Manual,* GC21-7516; information concerning inquiry and rollout/rollin for Model 15 can be found in the *IBM System/3 Model 15 System Control Programming Reference Manual,* GC21-5077.

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 Bibliography, GC20-8080, 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.

PREFACE

This manual discusses the disk concepts and planning information you need to know to design computer applications for the IBM System/3 Model 6, Model 10 Disk System, and Model 15. The book is intended for programmers who design applications for their company.

The System/3 Model 8 is supported by System/3 Model 10 Disk System control programming and program products. The facilities described in this publication for the Model 10 are also applicable to the Model 8, although the Model 8 is not referred to. It should be noted that not all devices and features that are available on the Model 10 are available on the Model 8. Therefore, Model 8 users should be familiar with the contents of *IBM System/3 Model 8 Introduction*, GC21-5114.

This manual applies to these program products:

- *System/3 Model 10 Disk RPG II (5702-RG1)*
- *System/3 Model 6 RPG II (5703-RG1)*
- *System/3 Model 15 RPG II (5704-RG1)*
- *System/3 Model 10 Subset ANS COBOL (5702-CB1)*
- *System/3 Model 15 ANS COBOL (5704-CB1)*
- *System/3 Model 10 Disk FORTRAN IV (5702-F01)*
- *System/3 Model 15 FORTRAN IV (5704-F01)*
- *System/3 Model 6 Disk FORTRAN IV (5703-F01)*

Differences between these RPG II, COBOL, and FORTRAN programs are noted when applicable, and references are made to related publications.

The chapters of this manual should be read in a specific sequence, as described in *How to Use This Publication* which follows.

You should be familiar with the *IBM System/3 Disk System Introduction*, GC21-7510, the *IBM System/3 Model 8 Introduction*, GC21-5114, the *IBM System/3 Model 6 Introduction*, GA21-9122, or the *IBM System/3 Model 15 Introduction*, GC21-5094, depending on the system you have.

After completing this manual, you should be able to write basic programs with the aid of various reference manuals. 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.

HOW TO USE THIS PUBLICATION

This publication has eight chapters and two appendixes:

- Chapters 1 through 5 discuss the basic characteristics of the IBM 5444 Disk Storage Drive and the IBM 5445 Disk Storage, and describe the following basic file organizations:

  Sequential files

  Indexed files

  Direct files

  Record address files

- Chapters 6 through 8 discuss 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. Information in these chapters is basically the same for the 5444 and 5445, but specific differences are noted.

- *Appendix A* describes the calculations necessary to determine how much disk space a file will require.

- *Appendix B* describes some performance factors to consider when using indexed files.

Chapters 1 through 5 of this manual are for users who need a basic knowledge of how to use disk files. Chapters 6 through 8 can be read after the reader thoroughly understands the basic concepts discussed in Chapters 1 through 5. Appendix A should be read for information about how to calculate file space. Appendix B will help those who plan to use indexed files.

CONTENTS

The IBM System/3 Model 6, Model 10 Disk System, and Model 15 can 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. IBM 5445 Disk Storage, on the other hand, can be attached to the IBM System/3 Model 10 Disk System and the IBM System/3 Model 15 to provide additional storage capacity; no libraries can reside on the 5445.

The major advantages of storing information on disk instead of on cards are:

- Large storage capacity. A 5444 disk can hold as much 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.

IBM 5444 Disk Storage Drive

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 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.

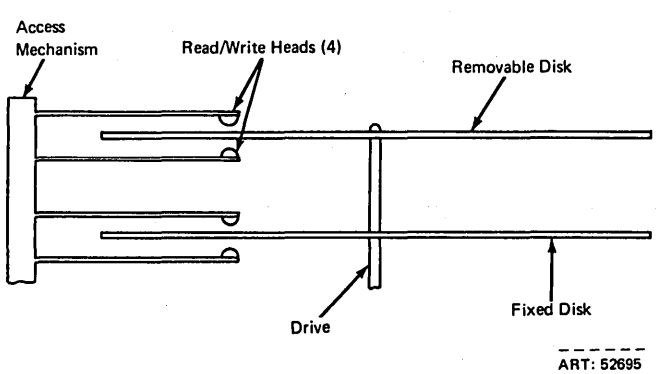
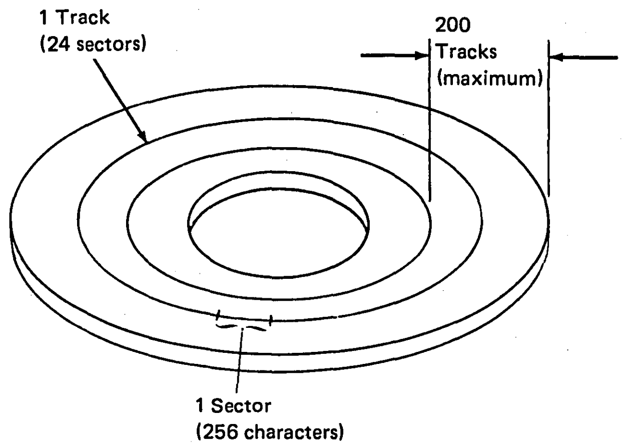
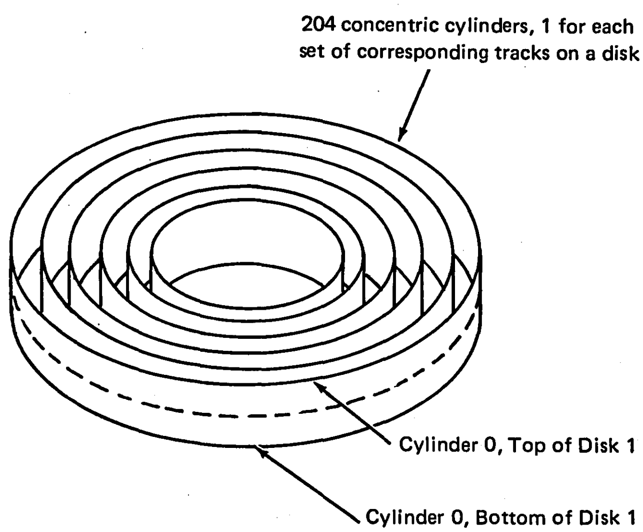


Figure 1. IBM 5444 Disk Storage Drive

Each surface of each 5444 disk provides the user with 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 (bytes) 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.



For this example, cylinders are numbered 0 through 203, beginning with the outer cylinder. IBM customer engineers use cylinder 203 for diagnostic functions, so this cylinder is not available for permanent storage. Tracks in cylinders 1, 2, and 3 are used by IBM programming as alternate tracks whenever tracks in cylinders 1 through 202 are found to be defective; therefore, if IBM programming is being used, cylinders 1, 2 and 3 are reserved for use as alternate tracks. Cylinder 0 is used by IBM-supplied programming support.

Although there are actually 104 or 204 tracks per surface depending on which model you have, only 100 or 200 are available to the user. In this manual and elsewhere, capacity is referred to as either 100 or 200 tracks per surface or 200 or 400 per disk pack.

The IBM 5444 Disk Storage Drive is available in these 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

* Models 6 and 10 only

IBM 5445 Disk Storage

IBM 5445 Disk Storage has one or two drives for the Model 10 Disk System or from one to four drives for the Model 15. Each drive uses a disk pack that 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 2).

Each surface of each 5445 disk contains 200 tracks. Tracks are divided into 20 sectors; each sector has a unique address, and contains 256 characters (bytes) of data.

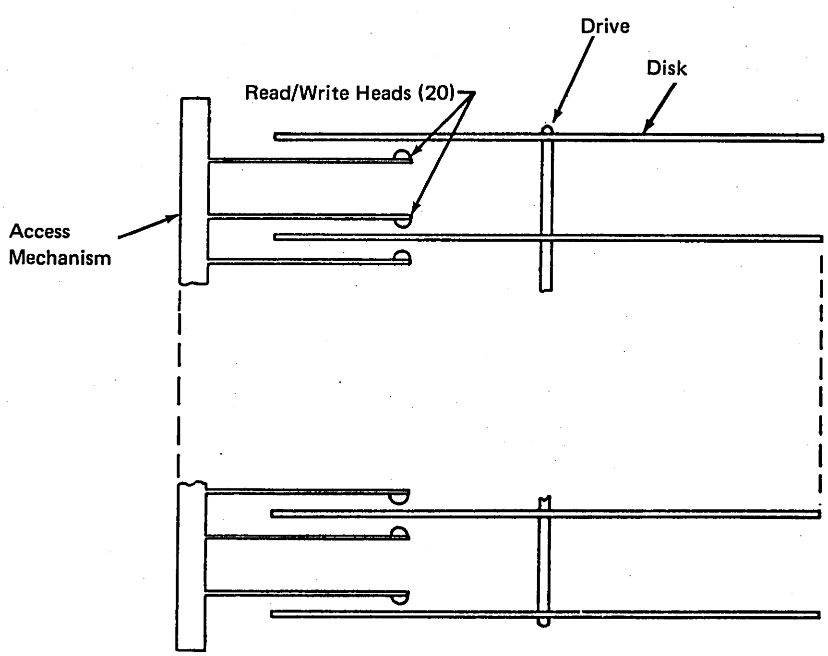


Figure 2. IBM 5445 Disk Storage

A 5445 cylinder consists of all the tracks on a disk pack in one vertical plane (Figure 3). 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 in the cylinder.

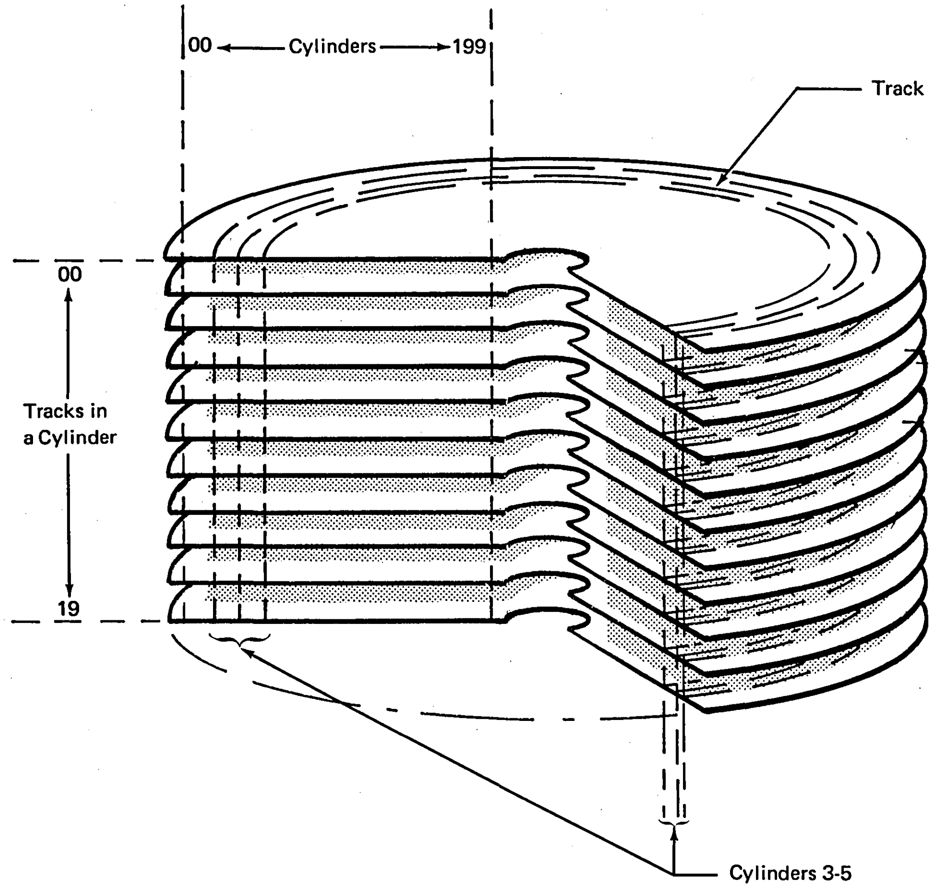


Figure 3. Cylinder Concept on the IBM 5445

Storage Characteristics (5444 and 5445)

Figure 4 shows the relative storage characteristics of the IBM 5444 and IBM 5445 Disk Storage drives.

	5444	5445
Bytes per sector	256	256
Sectors per track	24	20
Bytes per track	6144	5120
Tracks per cylinder	2	20
Bytes per cylinder	12,288	102,400
Cylinders per disk pack	100/200	200
Bytes per disk pack	1,228,800/ 2,457,600	20,480,000
Tracks per disk pack	200/400	4000
Sectors per disk pack	4800/9600	80,000
Maximum number of disk files stored per disk pack	50	50
Maximum number of usable disk surfaces	8	40 (Model 10); 80 (Model 15)
Maximum number of disk drives	2	2 (Model 10); 4 (Model 15)

Figure 4. Characteristics of the IBM 5444 and 5445 Disk Storage Drives

Comparative Access Times (5444 and 5445)

Figure 5 illustrates the access times available on the IBM 5444 Disk Storage Drive (normal and high speed) and the IBM 5445 Disk Storage drive. For more information, see the *IBM System/3 Model 10 Components Reference Manual,* (GA21-9103), the *IBM System/3 Model 6 Components Reference Manual,* GA34-0001, or the *IBM System/3 Model 15 Components Reference Manual* (GA21-9193).

	5444 (normal) *		5444 (high speed)		5445
	100 cyl	200 cyl	100 cyl	200 cyl	
Minimum access time	39 msec	39 msec	28 msec	28 msec	25 msec
Average access time	153 msec	269 msec	86 msec	126 msec	60 msec
Maximum access time	395 msec	750 msec	165 msec	255 msec	130 msec
Data transfer rate	199,000 bytes/sec		199,000 bytes/sec		312,000 bytes/sec
Rotational speed	1500 RPM		1500 RPM		2400 RPM
Average rotational delay	20 msec		20 msec		12.5 msec

* Models 6 and 10 only

Figure 5. Comparative Access Times (5444 and 5445)

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 physical order in which 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 pack. A multivolume file is a file that is contained on more than one disk pack.) 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 6.

Creating a Sequential File

You create a file when you write 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. All tracks in one cylinder are filled first, then all tracks in the next cylinder, and so on until the whole file is placed on the disk.

Figure 6 shows an example of this process using a 5444. In this example, each record is 128 positions (bytes) 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 numbers on the tracks in Figure 6 correspond to the number and position of each record.

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.

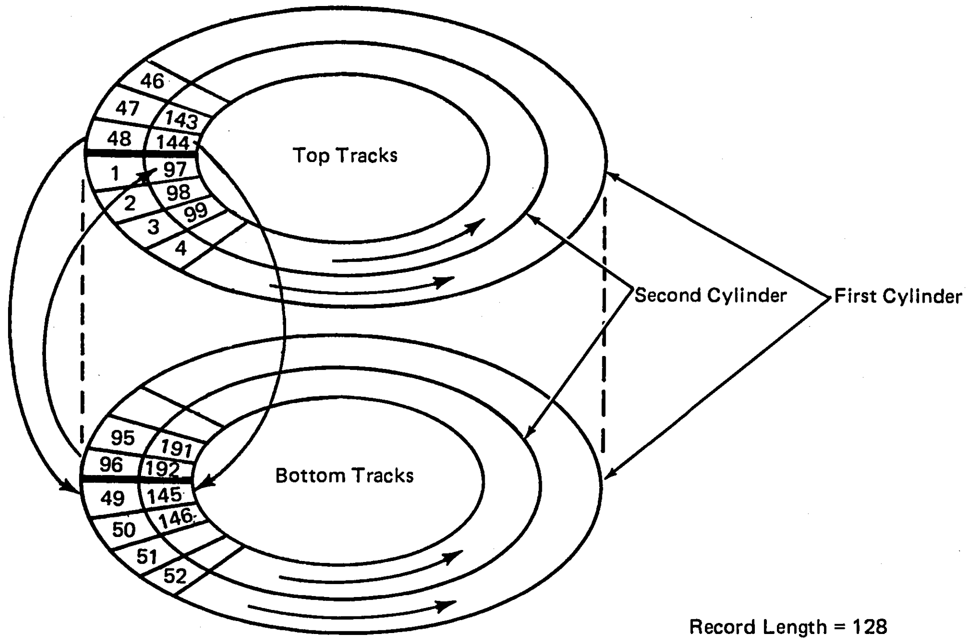


Figure 6. Writing Records on a Disk

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 affect or 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 already in the file. This may be necessary in order to keep the file in a particular order when the control fields of the new records are not higher in sequence than those already in the file. In order to put the new records in the proper sequence, you must sort the file to create a new file containing the added 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.

Note: Adding records to a sequential file is not supported by COBOL. A FORTRAN program must read all existing records first, and then begin writing.

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 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.

In some data processing applications you may not want to process your file consecutively. Consecutive processing is 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.

Note: This chapter and any other discussions of indexed files in this manual do not apply to FORTRAN; indexed files are not supported by FORTRAN.

An indexed file is organized into two parts: an index and the data 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.

Under certain conditions up to three types of indexes may be used. These index types are given specific names in this manual to eliminate confusion. The first, and most used, index is referred to as the *file index.* In some cases when using the 5445, the system may generate an index (on disk) known as the *disk track index.* Still another type of index, used to improve performance, is the *core index.* For more information on these three indexes, see *Appendix B.*

Each entry in the file index describes a record in the file. There is an entry in the file index for each record in the file. For example, if a file index has 2000 entries, the file contains 2000 records. The first part of the entry contains the record's *key field.* Each entry (key) in 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 file index entry contains the *disk address* of the record. The disk address represents the location on the disk where the record is stored. The file index is arranged in ascending sequence according to the key field in each record.

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 6.

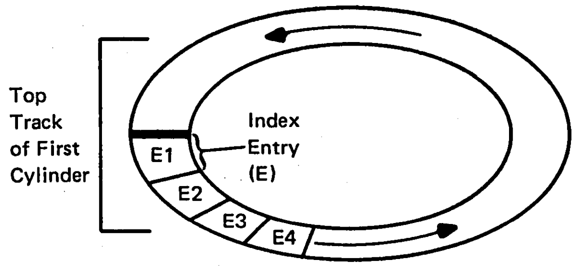
Creating an Indexed File

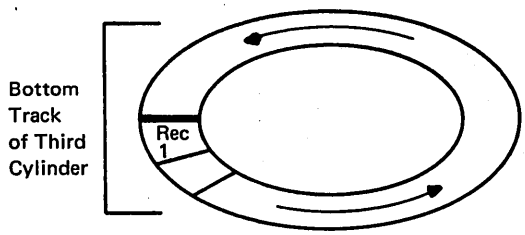
When you create an indexed file for RPG II, the records in the file can be in an ordered or an unordered sequence; when creating an indexed file for COBOL, however, the records must be in ascending sequence, as determined by their keys. An ordered sequence means the records are arranged in order according 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 file index is created as the records are written on disk. If the file is an ordered file, the file index is in the correct sequence when the records are written. If the file is an unordered file, the system automatically sorts the file index into ascending sequence after all the records in the file have been loaded. (The time required for sort can be reduced if the special work file $INDEX44 or $INDEX45 is available.)

The file index area precedes the area where records are placed on a disk. For example, suppose the file index for a certain file requires five tracks. The file index entries would be written on the first five tracks of the file. Records would be written beginning in the first sector of the sixth track. Both the file index area and the record area must start at the beginning of a track.





For indexed files on the 5445, another type of index is created when the file index uses more than 15 tracks. This additional index, which precedes the file index, is known as the *disk track index.* Each entry in the disk track index refers to one track of the file index. The disk track index will be used by the system only if its use will improve performance. See Appendix B for more information on this subject.

Processing an Indexed File

Indexed files are not limited to consecutive processing; they can be processed several ways because the file index provides several ways to find records.

Sequential Processing by Key

When an indexed file is processed sequentially by key, 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 7. Both files contain the same records, and both file 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.

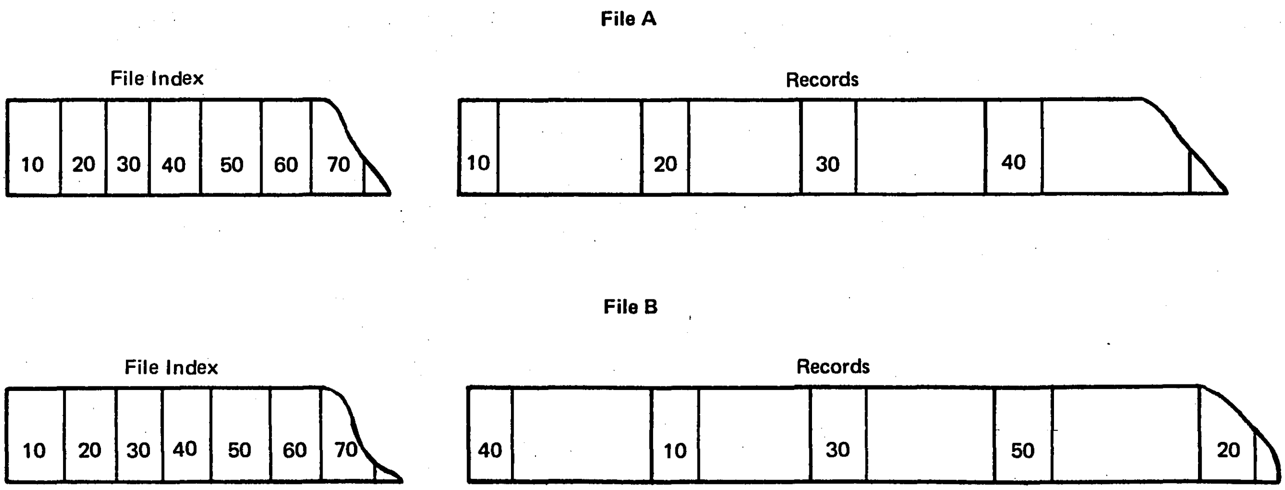


Figure 7. Example of an Ordered and an Unordered File

Sequential Processing Within Limits

Another way to sequentially process an indexed file is sequentially within limits, a method in which records are processed in groups.

Note: COBOL supports starting key (lower limit) processing only. Upper limit processing, if desired, must be provided in your COBOL source program. The limits for an RPG II object program can be supplied by a limits record or the lower limit can be set in your program. For multivolume files, this type of processing applies only to Model 15.

As an example of sequential processing within limits, suppose that a wholesale company prepares monthly statements of each customer'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 customers 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 file 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.

Processing an Indexed File Consecutively

Indexed files can be processed (read) consecutively by defining the indexed file as a sequential input file in the File Description Specifications. When an indexed file is processed consecutively, the file index is bypassed and data records are processed consecutively from the beginning of the file to the end, as if it was a sequential file. Note that indexed files can not be created, added to, or updated consecutively.

An example of using consecutive processing of an indexed file is reading records from an indexed file when the file index is unusable for some reason.

Maintaining an Indexed File

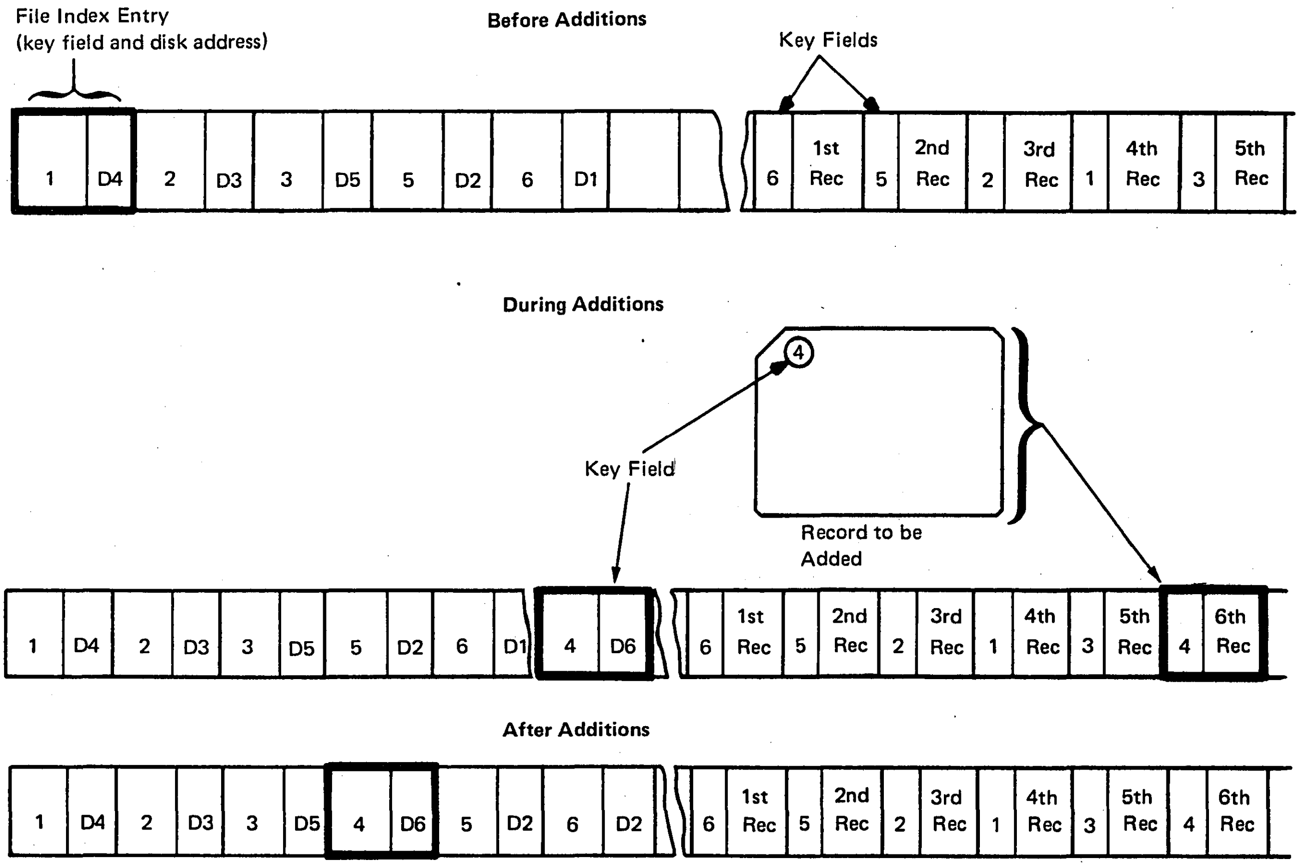
After the file is created, you can use these file maintenance functions to keep the file current for daily processing needs:

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. When records are added *randomly* by key (the records to be added need not be in any particular sequence) or *sequentially* by key, 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 file 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 keys of the added records and the keys of the original records are sorted or merged, so that the keys of all records in the file are in ascending sequence in the file index, as follows:



If many records are to be added to the file, the time required for the index sort/merge can be decreased by allocating a special work file. This requires no special RPG II coding but does require that the //FILE statement be included in the OCL statements, and that the special file name $INDEX44 or $INDEX45 be specified. See the *IBM System/3 Model 10 Disk System Control Programming Reference Manual* (GC21-7512), the *IBM System/3 Model 6 Operation Control Language and Disk Utility Programs Reference Manual* (GC21-7516), or the *IBM System/3 Model 15 System Control Programming Reference Manual* (GC21-5077), for more information concerning these requirements.

Tagging Records for Deletion

Inactive records in an indexed file must be handled like 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; a delete code cannot be put in the key field. When the file is processed, your program can 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 records to be updated are read into storage, changed, and written back on the disk in their original locations. Records in an indexed file can be updated:

1. Sequentially by key

2. Randomly by key

3. Sequentially within limits

Note: COBOL supports starting key (lower limit) processing only; upper limit processing, if desired, must be provided in your COBOL source program. The limits for an RPG II object program can be supplied by a limits file, or the lower limit can be set in your program.

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 file 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 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 file 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. The space previously occupied by the old file can then be used to contain new data.

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 8 shows direct file organization. Records are assigned specific locations, independent of the order they are put into the file. All records 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 (in other words, the entire file must be online). Therefore, multivolume direct files on 5444 disk drives 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). Multivolume direct files on 5445 disk drives are limited to two volumes for the Model 10 or four volumes for the Model 15. For more information on processing considerations when using multivolume direct files, see the discussion on multivolume files in Chapter 6.

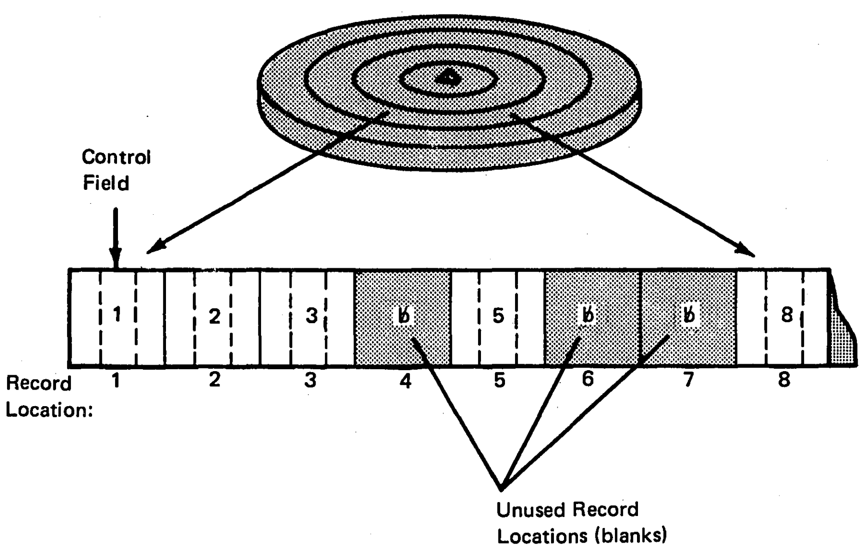


Figure 8. Direct File Organization

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 8, 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 which includes 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:

	74 | 9397	
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 9 shows relative record numbers taken from the student file number being used to update student addresses.

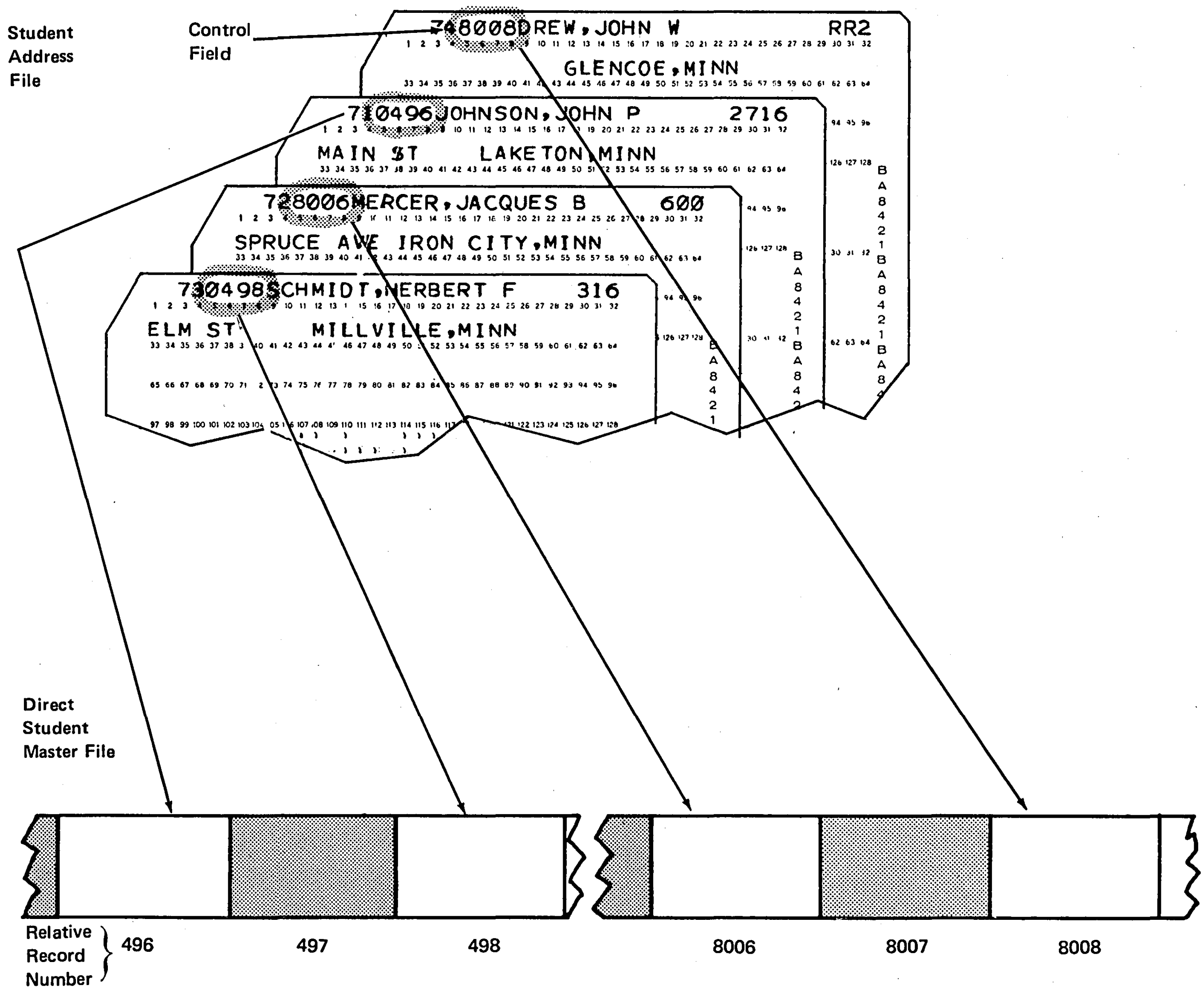


Figure 9. Relative Record Numbers Corresponding Directly to a Control Field

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 must be found for converting the employee numbers to approximately a 1 through 500 range (which would provide 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, but contain different data (see the next section, *Synonym Records*). Your program must allow for synonyms if they are generated.

A 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, there is a possibility of having synonym records. For example, if the numbers 6, 7, and 8 are present, all three become relative record number 2.

Another technique that may produce 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, rather than 50.

A third method would be to divide the employee number by 499 (500 - 1), and use the remainder + 1 as the relative record number.

If there is no sequence to numbers in a control field (such as part numbers), 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 the resulting digits, only four are used as the relative record number for the parts inventory file.

Part number = 468152

$468152 \times 468152 = 2191\boxed{6629}5104$

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 selected 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.

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. Since only one synonym record can be stored in the record location for its relative record number, a different method must be found to store and retrieve the other synonym records.

Chain Technique

One way to handle synonyms is to chain (link) them together so that all can be found by locating the first. 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 (second record) is stored in the first unoccupied record location in the file (a location for which no relative record number had been 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 synonym, and so forth. In Figure 10, 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 done in a separate program. The chain technique is useful when a file is created, but tends to be of less value as records are added to or deleted from a file.

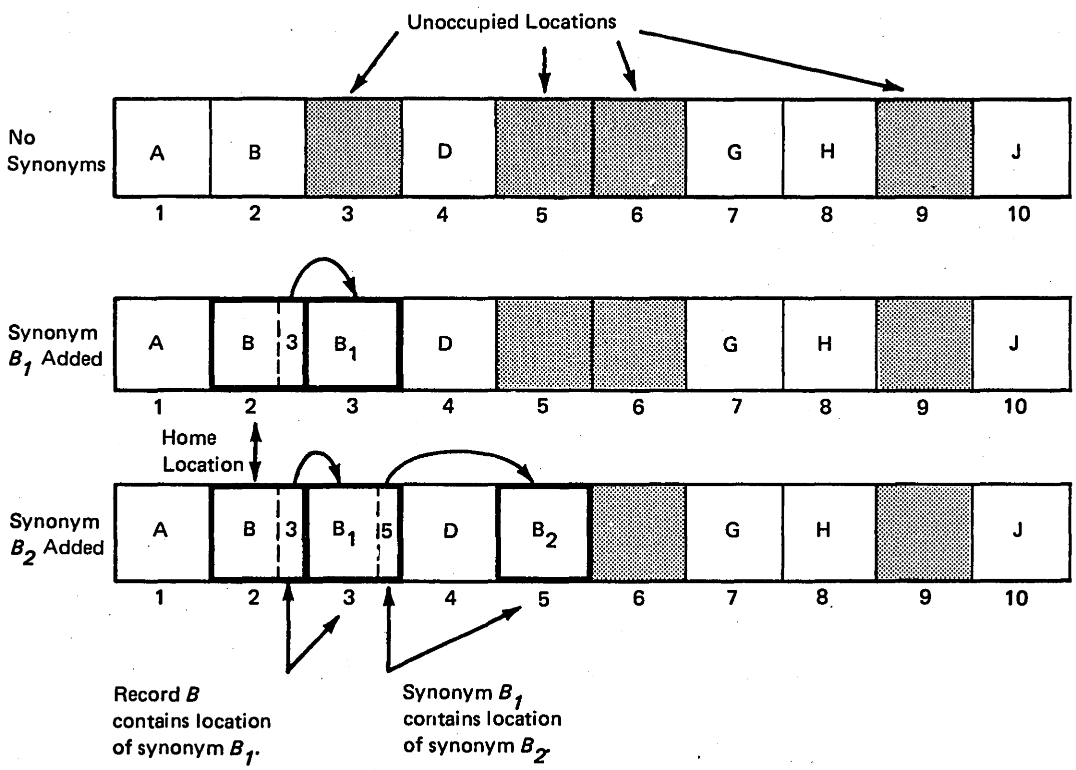


Figure 10. Storing Synonym Records in a Direct File

If a new record is added to the file, but its home location is already occupied by a synonym, for a different record location, the new record must be treated as a synonym for its home location. Figure 11 shows the file that resulted from the addition of synonyms in Figure 10. 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.

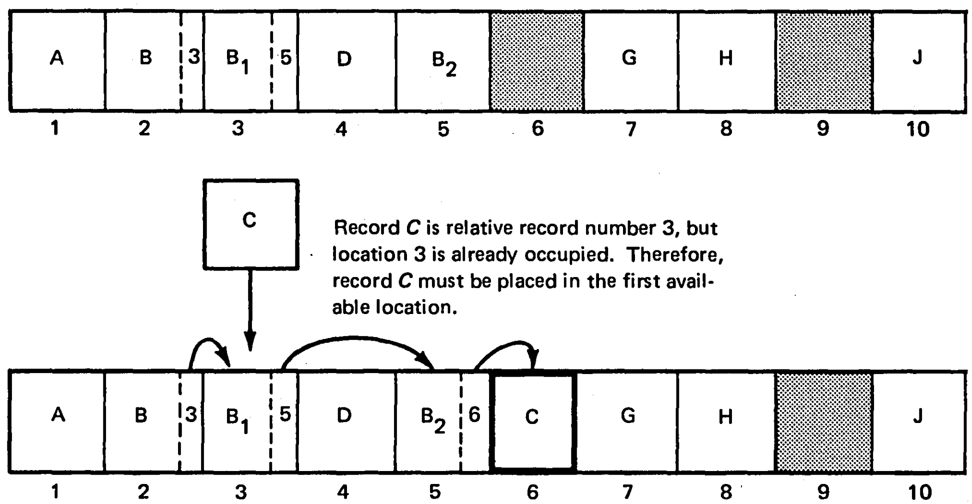
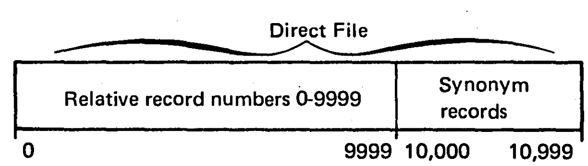


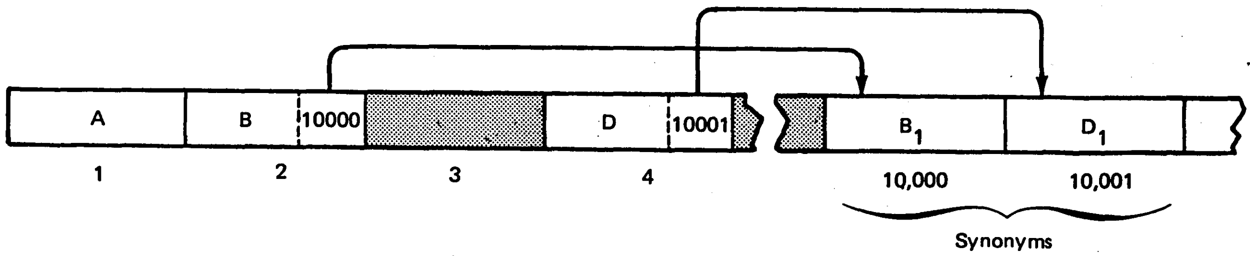
Figure 11. Storing a Record When Its Home Location Is Occupied

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 11, you get record B_1, which is a synonym for relative record 2, which is not the record you want. However, if you check the record retrieved, you find that it is a synonym. You can now chain 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 could eventually 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.



Processing by this method is faster when records must be added to a file because a home location is kept free for every relative record number; only one seek operation is required for records without synonyms. However, this method wastes more file space, because 11,000 locations are used for 8500 records.

Spill Technique

Another method of handling synonym records, the spill technique, uses the home location as a starting point. When the file is first loaded, a counter is set to indicate the maximum number of reads which would be necessary for locating a given synonym record. (For example, the counter would be set to 3 if the maximum number of synonyms for a given home address were 3.) To retrieve a record from the file, you would first need to determine the home record location and read the record from that address. If it isn't the record you want, you read the record in the next location in the file. This process continues until the correct record is selected from the file. If the maximum number of reads (3 in the example, above) is reached, a record-not-found condition exists.

When a record is to be added to a file, you first check the location at the home address. If this location indicates that the home record has a synonym, you increment the relative record number by one, and continue to check for synonyms, until an available space is found. At that point you would add the new record to the file. If the number of times you incremented the relative record number exceeds the count you set up for the maximum number of reads, the count would be incremented by one (in the example, the count would be set to 4).

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

Creating a Direct File

To create a direct file, you must define a disk file as: a chained output file (for RPG II); a random output file (for COBOL); or, a direct access file (for FORTRAN). In this way, the file is uniquely identified to disk system management as a direct file. Disk system management then allocates disk space for the file, and the entire file space is erased to blanks. This action, in effect, creates dummy records whose length is determined by the creating program. Once the file has been cleared, one or more subsequent jobs can be run to read record locations while loading the file. 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. Records are loaded into the file in an update mode by first chaining the record to a given record location according to its relative record number, and then by outputting the new record into that record space. The relative record number is the sequence number of that record within the file. 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 do not have synonyms, you can load records into a direct file in a single pass. In this case, record locations are not 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 *Synonym Records*).

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 (see *Chapter 5. Record Address Files*)

Consecutive Processing

Direct files are often used 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. The direct file has no next available record (EOF) pointer in the label. As a result, consecutive processing will access the entire file space before the last record (LR) condition occurs. Remember that a direct file is cleared to blanks when it is created, and record locations 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 does not depend on 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, retrieving a record by location may require more extensive programming, 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 12 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.

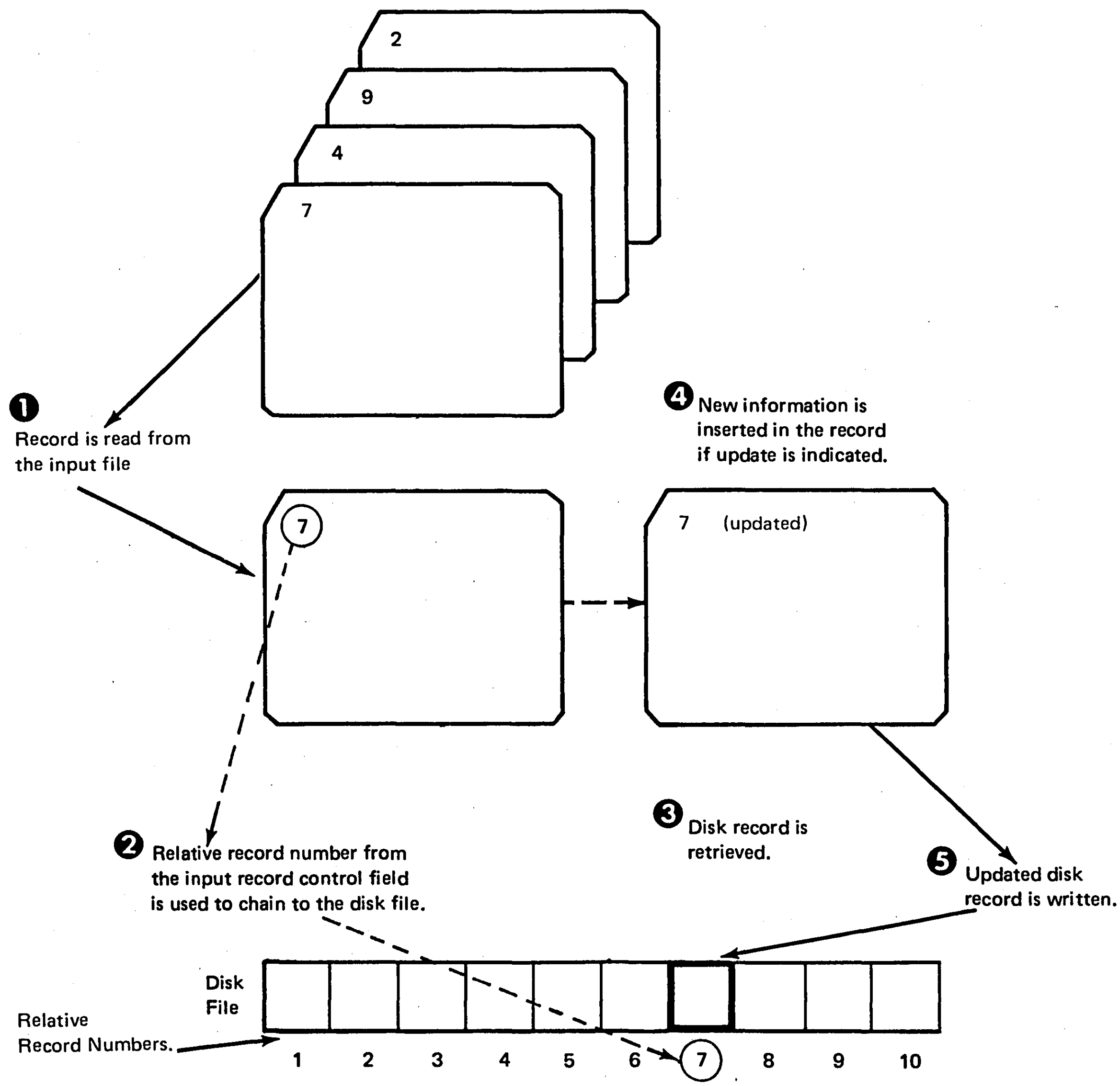


Figure 12. Random Processing by Relative Record Number

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, or it 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.

MANIPULATING DIRECT FILE DATA

Direct file organization on the System/3 offers you a flexible tool for data manipulation that is not available in the other organization methods. With direct organization, you can:

- Access a file consecutively more than once in the same program.
- Load a file, then retrieve the records in the same program.
- Tie together strings of related records so they can be retrieved as a group when they are not necessarily stored together in the file.
- Build and retrieve message queues in a communications system.
- Use a direct file for large arrays.

Using the techniques discussed in this section, a direct file can be used over and over without being re-created; existing records are re-written when the file is used. Consequently, it is usually convenient to create the file with a program that does not load any data. Then all of the accessing programs can define the file as an update, chained, direct, or random file. The examples in this section assume a previously created file.

The techniques described normally require that records be placed in the file in consecutive record locations. The programs will use one or more counters (numeric total fields) to keep track of the next relative record number.

Accessing a File Consecutively

To access a file consecutively more than once in the same program, the program increments the record number counter by one each time a record is accessed, and then chains to the file. This action is repeated until the last record is read. The counter is then reset to zero and the process is repeated. The program recognizes the last record in the file by (1) identifying the last record with a specific code and testing for that code, or (2) by testing for the first block record in the file, or (3) by knowing the record number of the last record.

Loading and Retrieving Records in the Same Program

In update mode, the record number counter is used to load records in consecutive record locations. After records have been loaded, they can be retrieved by record number using the chain operation.

Connecting Strings of Related Records

This technique, known as chaining, requires that each record in the file contain an extra field. That field will contain the record number of the next record in the string. A blank or zero field can be used to identify the last record in a string.

The chaining technique works well in an accounts receivable application. For example, a customer master file is indexed by customer number. Transactions are added consecutively to a direct file as they occur and are applied to a balance field in the customer master record. An inquiry to the master file will cause the balance information and all transactions for that customer to be displayed.

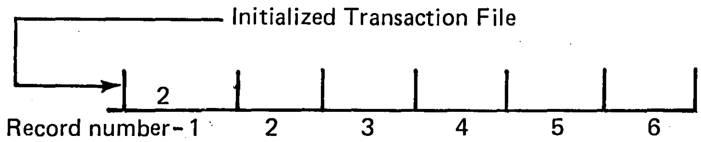
This is accomplished by adding two fields to each customer master record. These fields contain the record numbers of the first and last transaction records (respectively) for that customer in the transaction file. These fields are set to blank or zero at the beginning of the accounting period and remain set at zero until the first transaction is posted for that customer.

Customer Master Record Format

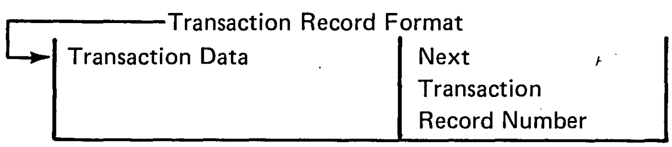
Customer Data	First Transaction Record Number	Last Transaction Record Number

Record 1 in the transaction file is reserved for storing the record number of the next available record space in the file at the time the file is closed. When the file is initialized at the start of the accounting period, record 2 is the next available record.

When transactions are added to the file, record 1 is read at the beginning of the job by the program, to establish where the next transaction will be placed. The value stored in record 1 is increased by one each time a record is added (the new value is written back into record 1 at LR time).



Each transaction record contains a number that is used to locate the next transaction record to the same customer.



Two routines are needed to load transaction records into the file. One loads the first transaction for a customer; the other loads all subsequent records for the customer.

Assuming (1) the transaction file is the primary file, (2) the customer master record has been accessed by a CHAIN operation, and (3) the first transaction record number field is blank or zero, the following is an example of how the first transaction record is loaded and the records set for a customer:

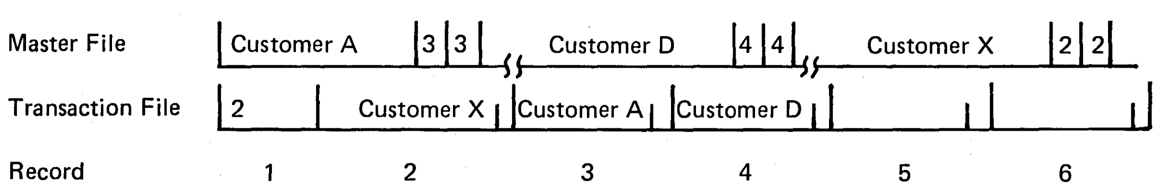
1. Using the next available record number (from record 1) chain to the transaction file.

2. Put the new transaction record out in the record space.

3. Place the next available record number in both the first and last number fields of the master record.

4. Add one to the next available record number.

If one transaction had been loaded for customers X, A, and D, the files would appear as follows:



5 Pointer to next available record (in storage)

The following describes how subsequent records are added:

1. Using the next available record number, add the new transaction to the file.

2. Using the last record number field from the master record, chain to the last transaction for that customer.

3. Update this record by placing the next available record number in its next transaction record number field.

4. Place the next available record number in the last transaction record number field of the master record.

5. Add one to the next available record number.

Assume that one transaction has been added for customer X, one added for customer D, and another added for customer X. The files would then appear as follows:

Master File —

Customer A	3	3	Customer D	4	6	Customer X	2	7

Transaction File —

Record number —	1	2		3	4		5		6	7	
Transaction File —	2	CustX	5	CustA	CustD	6	CustX	7	CustD	CustX	

8 — Next available record (in storage)

Remember that the next available record number will be written into record 1 at LR time.

Message Queuing in a System/3 Direct File

In a communications environment, it is often necessary to store messages as they are received and make them available for processing at a later time. This technique known as *message queuing,* can be readily used with direct files, with the following restrictions:

- Variable length messages must be blocked by the user to fit the fixed length disk record.

- Queued messages will be processed on a first in-first out basis within a given queue. Records (messages) are placed in the queues in the same manner as transactions were placed in the transaction file in the accounts receivable example presented earlier in this section.

- Three pointers (record numbers) are normally required for each queue in the file: a pointer to the first record in the queue, a pointer to the last record in the queue, and a pointer to the next record in the queue to be processed.

Queue 1	First Record Pointer_1	Last Record Pointer_1	Next Record Pointer_1
Queue X	First Record Pointer_X	Last Record Pointer_X	Next Record Pointer_X

These pointers are usually maintained in arrays, with the queue numbers used for subscripts. Besides the three pointers previously mentioned, a pointer is required to the next available record in the file. When the file is closed, all pointers are stored in a reserved record in a file.

The next record pointer allows the processing program to retrieve records consecutively from a given queue. This pointer is initially set equal to the first record pointer, and is then changed each time a record is retrieved from the queue. This pointer may be maintained within the processing programs instead of in the file, to allow multiple processing programs to access the same queue. Each using program would keep track of its own processing position within a queue.

Using a Direct File for Large Arrays

Arrays that are too large to be held in main storage may be stored on disk as a direct file. The subscript value becomes the record number of the data stored in the file. There is no minimum record size in System/3 disk files. Data fields in an array may be stored as individual records in a direct file.

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 (address out) 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 when 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 than the output file of a tag-along sort because the output records of the ADDROUT file are only three bytes long (see *sorting a file,* in Chapter 6).

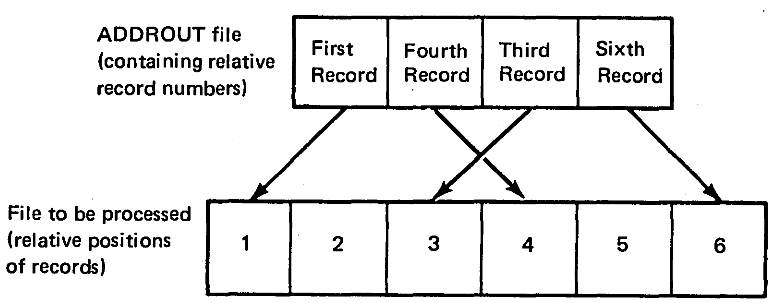
3. If an ADDROUT file is used to process a multivolume file (RPG II and COBOL only), all volumes of that file must be mounted during processing because the next record required may be on any volume.

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. For RPG II, 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 13 shows an ADDROUT file used to process a disk file.

Note: COBOL uses only direct file organization for this application.

A different approach is needed when using FORTRAN and COBOL. You would define the ADDROUT file as an input file, and the corresponding direct file as another input file. Your program would then read from ADDROUT and put the input data into the associated variable (specified in the file definition statement) for the direct file. Execution of a READ statement would then retrieve the desired record from the direct file. You may terminate reading from ADDROUT either at its EOF or prior to its EOF. You must logically determine EOF for your own situation (for example, by a record count).



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 four. Since all records are not read, processing by ADDROUT file is random processing.

Figure 13. 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.*

Note: COBOL supports starting key (lower limit) processing only; upper limit processing, if desired, must be provided for in your COBOL source code. The limits for an RPG II object program can be supplied by a record, or the lower limit can be set in your program.

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, used one at a time.

For instance, in the example explaining sequential within limits in Chapter 3, the customers were divided into four regions. If you wanted to process only 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 (RPG II and COBOL only) 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.

For Model 6, Model 10 Disk System, and Model 15, single volume indexed files may be processed using limits. In addition, on the Model 15, a multivolume file may be processed using limits.

CHAPTER 6. CHOOSING A FILE ORGANIZATION

Chapters 1 through 5 of this manual described several disk file organizations that can be used with the IBM System/3 Model 6, Model 10 Disk System, and Model 15, and 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 more thought 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 the 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 (frequency of additions and deletions) 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. An example of such a file is a customer file. 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 (these changes are made with a transaction file), the record remains in the file as long as the customer does business with the company. 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.

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.

Note: FORTRAN does *not* support indexed file organization.

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. Random 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 either indexed or direct is a better type of organization than is sequential 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 it to be 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?

System/3 provides 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 response must be, ask yourself the following question: *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, can be either sequential, indexed, or direct, depending on other processing needs.

If the inqiury cannot wait, another question 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.

Volatility of the File

The number of records added to or deleted from a file is another important consideration in choosing the type of file organization to use. *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 having to allow 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 until 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 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.

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 would justify the use of a sequential file with consecutive processing or an indexed file processed sequentially by key. Low activity would warrent use of 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 (RPG II and COBOL Only)

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.

Note: Model 10 COBOL supports only multivolume sequential or direct file organization; Model 15 COBOL supports multivolume indexed file organization in addition to multivolume sequential or direct file organization.

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. For example, 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.

An indexed file can be processed randomly using an offline multivolume file, but only if the file was created with this technique 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.

The table that follows shows the valid devices and file organizations for the files used by the System/3 Disk Sort program.

	Devices	**File Organization**
Input files	5444, 5445	Sequential Indexed Direct
	Tape	Sequential
Work files	5444, 5445	Sequential
Output files	5444, 5445	Sequential
	Tape	Sequential

All volumes of a given input, work, or output file must be of the same device type. Input and output files can be single volume or multivolume (online or offline); work files can be single volume or online multivolume only. For more information, see the *IBM System/3 Disk Sort Reference Manual* (SC21-7522).

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 they can be the entire input record. The output for a tag-along sort is a file of sorted records that 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 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:

X376	3	A500	5
Cat. No.	No. Sold	Cat. No.	No. Sold
X376	4	A500	2
Cat. No.	No. Sold	Cat. No.	No. Sold
X376	10		
Cat. No.	No. Sold		

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:

X376	17	A500	7
Cat. No.	No. Sold	Cat. No.	No. Sold

The output for a summary sort is a file of sorted records that 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

An alternative to tag-along or summary sort is the ADDROUT sort. An ADDROUT sort produces a file of relative record numbers. The relative record number can be used by an RPG II or COBOL 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 or COBOL program that processes the records in the desired sequence.

The ADDROUT sort offers two advantages over the other sort types:

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

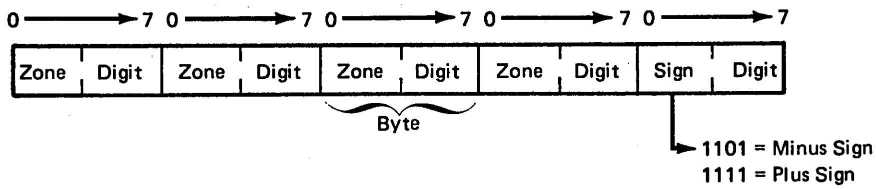
Determining Field Size

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.

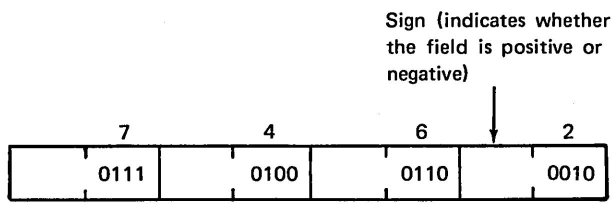
Numeric Fields

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

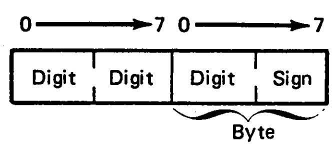
Unpacked decimal format means that each byte of storage, whether on disk or in the computer, can contain one character. (That character may be a decimal number or it may be an alphabetic or special character.) In the unpacked decimal format, each byte of storage is divided into a 4-bit zone portion and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



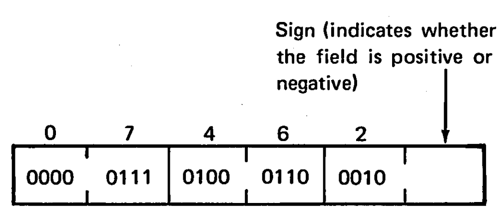
The zone portion of the rightmost byte indicates whether the decimal number is positive or negative. In unpacked decimal format, the zone portion is included for each digit in a decimal number; however, only the zone over the rightmost digit serves as the sign. The unpacked decimal format for decimal 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 24 shows the number of bytes needed for a specified number of characters in a packed field as compared to the number of bytes needed for that number of characters in 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 24. Number of Bytes needed for Specified Numbers of Characters in 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. Thus, a 1-position field is included to provide for a delete code.

Providing Extra Space

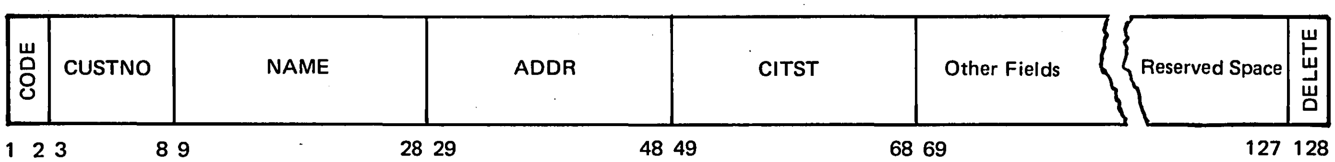
At this stage in planning, it is often desirable to allow for data to be added to a record. For example, suppose the name and address file were created with the fields described, but at a later time each customer's zip code is needed. If all positions in the record are used, there is no place to add 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 25 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 and abbreviations are often necessary, care should be taken to chose a meaningful field name. For example, the word *address* has seven letters; it is shortened to ADDR in Figure 25. 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 25. Layout of Customer Master Record

Documenting Record Layout

When record layouts are documented, your programs are easier to write. Figure 25 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

Although field lengths within a record may vary, the field lengths for the same fields in each record in a file should be the same, and 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 initial example in this section, the sum of the fields was set at 116 positions. However, record length (Figure 25) was established at 128, to reserve 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 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. 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.* Transferring blocks of records can decrease the time required to perform a job, because when records are transferred one at a time, access time is required for the disk access arm to locate each record, and when several records are transferred at a time, access time is usually less.

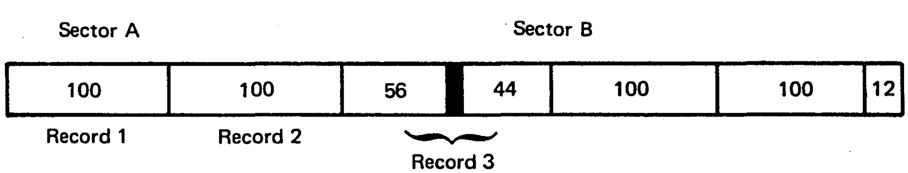
You may want to use unblocked records when a program takes a large amount of storage. Total time to do the job may incerase, but your program will fit in storage.

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 to be 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). See Figure 26 for examples of how record length affects computed block length.

You *can*, however, 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.

Record Length	Input/Output Area Allocated by RPG II**		Number of Records per Block	
	Group A	Group B*	Group A	Group B
32	256	256	8	8
60	256	512	4	8
64	256	256	4	4
80	256	512	3	6
96	256	512	2	5
128	256	256	2	2
256	256	256	1	1
512	512	512	1	1

*Files in Group B can require a larger input/output area than files in Group A.

Group A Files

Consecutive Output
Consecutive Input
Indexed Input without Add or Update, Processed Sequentially (Models 6 and 10)
Indexed Output

Group B Files

Consecutive Update
Indexed Input with Add or Update
Indexed File, Processed Randomly (Model 15)
Direct File

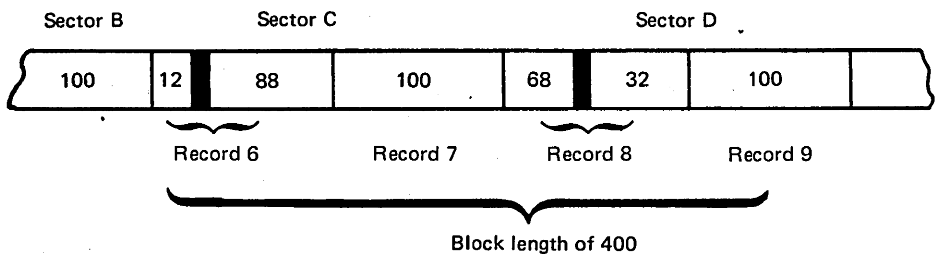
**These entries represent the number of bytes of I/O area that RPG II will use, assuming that the block length you have specified is less than or equal to the values shown in this figure, and that the block length is a multiple of record length. If the specified block length is greater than the values shown, RPG II will round the block length so that the computed size is a multiple of 256.

Note: This figure applies to: 5444 and 5445 files, single I/O areas for data only, single volume files only.

Figure 26. Size of Input/Output Area Computed by RPG II for Disk Files

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 100 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 400. Four 100-character records *can* span three sectors. To process your records in this case required 768 characters (three sectors) in main storage.



The block length for disk records is specified on an RPG II File Description Specifications sheet, and can be from 1 to 9999 bytes for disk files. The block length in a given program does not have to be the same as the block length specified when loading the file. Block length does not affect the way that records are written on disk, but is used to specify the amount of core to be used for the I/O area in the processing program. Block length can be as large or as small as the given program will allow; with a large block length, more records are available (in core) at a given time than if no blocking is specified. In RPG II, if block length is specified as equal to record length, the compiler will assign an efficient block length, to take advantage of the fact that the I/O area must be a multiple of the sector size (256 bytes).

Blocking can be an advantage if you are likely to process multiple records in the block – sequential processing, for example. However, if you are processing sequentially *with additions,* blocking may have an adverse affect on performance for Models 6 and 10; blocking does not affect performance for Model 15.

When processing randomly, you shouldn't specify a large blocking factor unless you are certain that the system will process more than one record in a block before getting another block.

Shared Input/Output Area for Model 6 and Model 10 Disk System — RPG II or COBOL and 5444 Only

Usually a program uses one input/output (I/O) area for each file. However, if you are using the 5444, and 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 5444 disk files in the program share a single I/O area. However, since a shared I/O area increases the time required to process your program, you should not use shared I/O areas unless your program is too large to fit into main storage. In COBOL, the SAME AREA clause is used to share an I/O area. Shared I/O is not available on the Model 15.

To determine the total I/O area needed when each file has its own I/O area, you find the block lengths assigned to each file and add them together. Determining the block length for RPG II is discussed under *Block Length* earlier in this chapter. For a discussion of this capability in FORTRAN, see *Sharing Buffers* in the *IBM System/3 FORTRAN IV Reference Manual,* SC28-6874; for a discussion of this capability in COBOL, see *Same Area Clause* in the *IBM System/3 Subset American National Standard COBOL,* GC28-6452.

Shared I/O does not allow for record blocking. 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. Shared I/O areas cannot be specified in a program if that program also specifies a 5445 file.

Buffered I/O

For certain types of processing (such as consecutive input or output), you can specify an extra I/O area. When this process, called buffering, is specified, an extra area is reserved so that the records being processed are directed first to one area, then to the other. Although specifying an extra I/O area allows the processing operations being performed to be overlapped, extra main storage is required, which reduces the amount of main storage available to the program. Use of dual I/O areas in an RPG II program may cause overlays that might not otherwise have been generated.

Determining Size and Location of a Disk File

Another aspect of the planning stage is determining (1) how much disk space a file requires and (2) where the file will be located 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 the disk in this example contains 200 cylinders, this file has too many cylinders to be contained on this disk (155 + 55 = 210). The file must be written on another disk.

Determining the Size of a Disk File

Appendix A contains examples of the calculations necessary to determine how much space a disk file requires. The following factors are discussed in Appendix A:

- Determining number of records in a file
- Calculating record space
- Determining number of tracks needed (5444 and 5445)
- Calculating index space (5444 and 5445)
- Calculating space for disk track index (5445 only)

Note: The file planning information discussed in this section is basically the same for the IBM 5444 and the IBM 5445. The calculations for determining the size of a disk file (Appendix A) are different, however, because: the 5445 has only 20 sectors per track as compared to 24 sectors per track for the 5444; for an indexed file, the disk address in the index entry is four characters in the 5445 instead of three in the 5444; and, a disk track index may exist for a 5445 file, but not for a 5444 file.

Deciding Where the File on Disk is to be Located

After you determine the amount of space the file requires, you can decide where the file should be located on the disk. Since the number of files a disk can contain depends on the size of the files, it is a good practice to document which files are on which disk.

The Disk File Layout Chart (Figure 27) is available for this purpose. The Disk File Layout Chart shows space available on the fixed and removable 5444 disks. There are 406 positions (0-405), represented on the chart. Each position corresponds to a track. In Figure 27, 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 File Layout Chart. It is also helpful to indicate the name of the file on the chart.

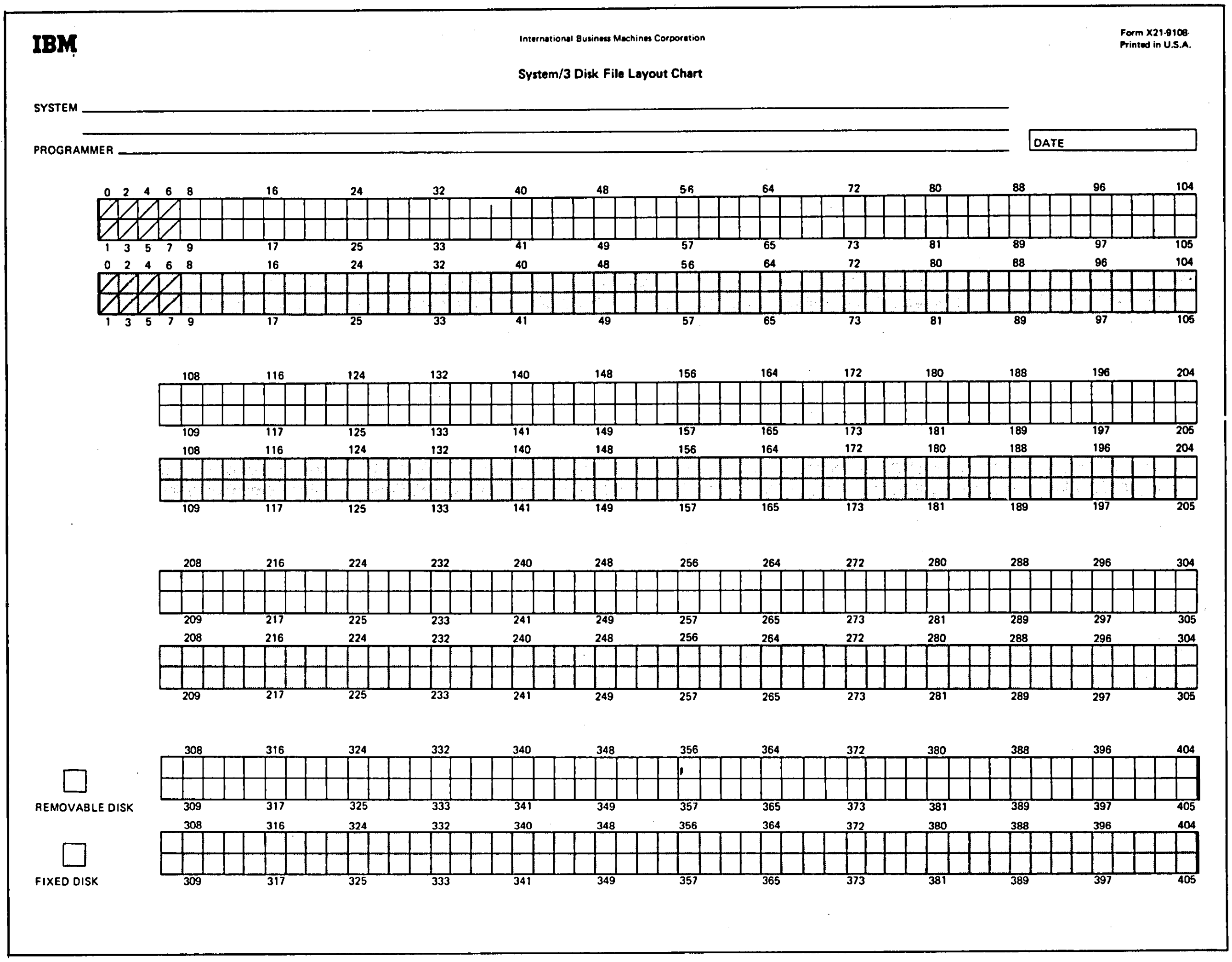


Figure 27. Disk File Layout Chart

Figure 28 shows the space and location of the name and address file using the indexed method. The calculations to determine the amount of disk space required can be done on the back of the chart.

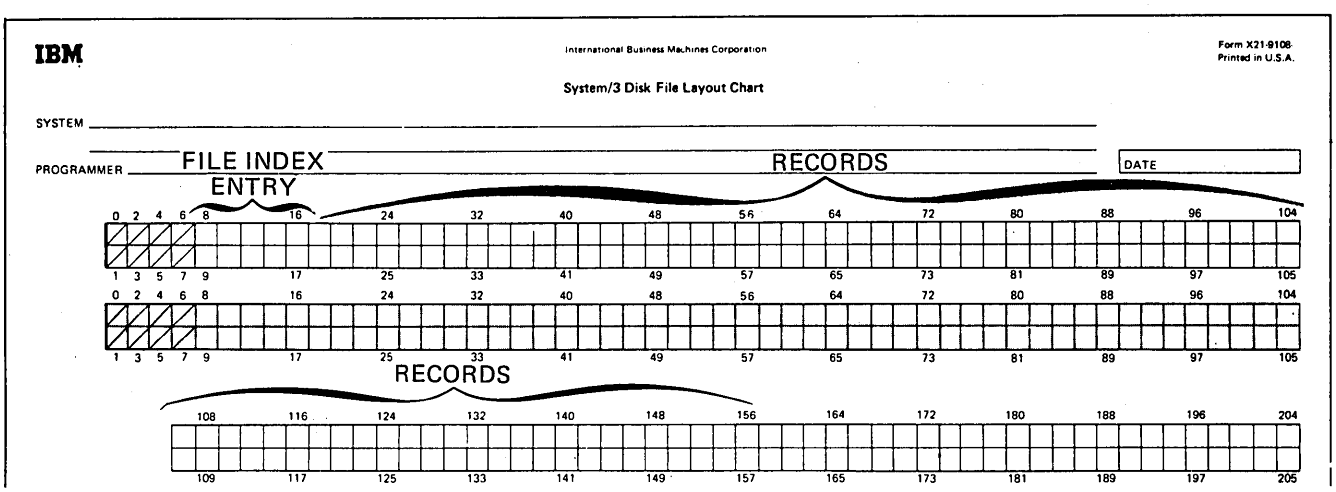
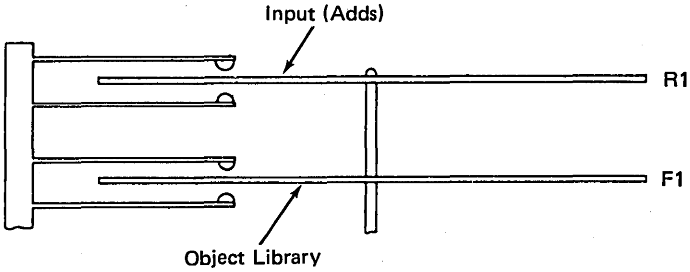
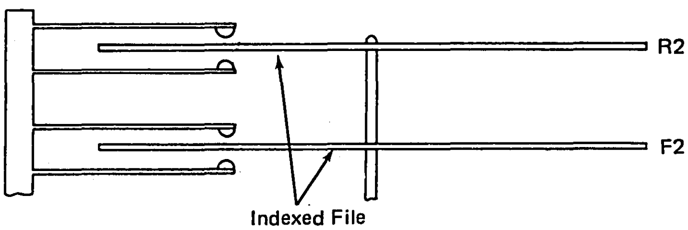


Figure 28. Disk File Layout for an Indexed File

Placement of files in relation to each other also has an effect on the performance achieved when processing them. For example, when adding records to a file, it is desirable to have the input on one disk drive and the file on another drive. In this way, the files can be located as follows for a program that processes an indexed file and adds records to it:





If the program used requires overlays, it might be desirable (depending on your application) for the input file to be located close to the object library to reduce arm movement on drive 1. In each RPG II cycle, it might be necessary for the arm to go to the input area for records to be added, and then to the object library for overlays.

Consideration might also be given to placing the input close to the index of the file, or near the midpoint of the file, or even near the end of the file, depending on the expected distribution of added records.

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-20,LOCATION-8
```

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.

```
1  4   8   12  16  20  24  28  32  36  40  44  48  52
   FILE NAME-FILEA,PACK-VOL1,UNIT-R1,TRACKS-20
```

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. This allows you as many available tracks as possible next to the object library (the object library is located at the lowest numbered tracks), so that 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, as shown below:

Part A	Permanent File	New Permanent File	Available Tracks	Scratch File

Part B	Scratch File	Available Tracks	New Permanent File	Permanent 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).

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 place 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 is greater than or equal to the number of tracks in the permanent file.

Considerations for Using Automatic File Allocation

It is easier to let disk system management allocate file space, but there are some considerations to make in determining whether or not to use automatic file allocation. After you have gained experience, you should be able to place a file on disk more efficiently than can 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.

If you plan your own files and keep your layout chart up-to-date, you can determine where files are located by checking the Disk File Layout Chart. If you allocate space for some files automatically and then want to place a file on disk 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 Model 10 Disk System Control Programming Reference Manual,* GC21-7512, the *IBM System/3 Model 6 Operation Control Language and Disk Utility Programs Reference Manual,* GC21-7516, or the *IBM System/3 Model 15 System Control Programming Reference Manual,* GC21-5077, 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 therefore be 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.

Using the automatic work file allocation function (auto-allocate) when running the Disk Sort program generally increases the time needed to run a sort job; auto-allocate does not always provide the work file arrangement needed for a fast sort run. If you are concerned with minimizing sort run time, use a well planned work file and work file statement, rather than auto-allocate. An advantage of using auto-allocate is that if sufficient contiguous space is not available, the system will find work space that may be located in different areas of the same pack or on different packs.

Automatic file allocation provides for effective use of file space, but not for file usage; it does not provide 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 placed near one another, processing time may be improved.

Split Cylinder Capability (5445)

The 5445 has a split cylinder capability for sequential or direct files (see Figure 29). This means that two or more sequential or direct 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.

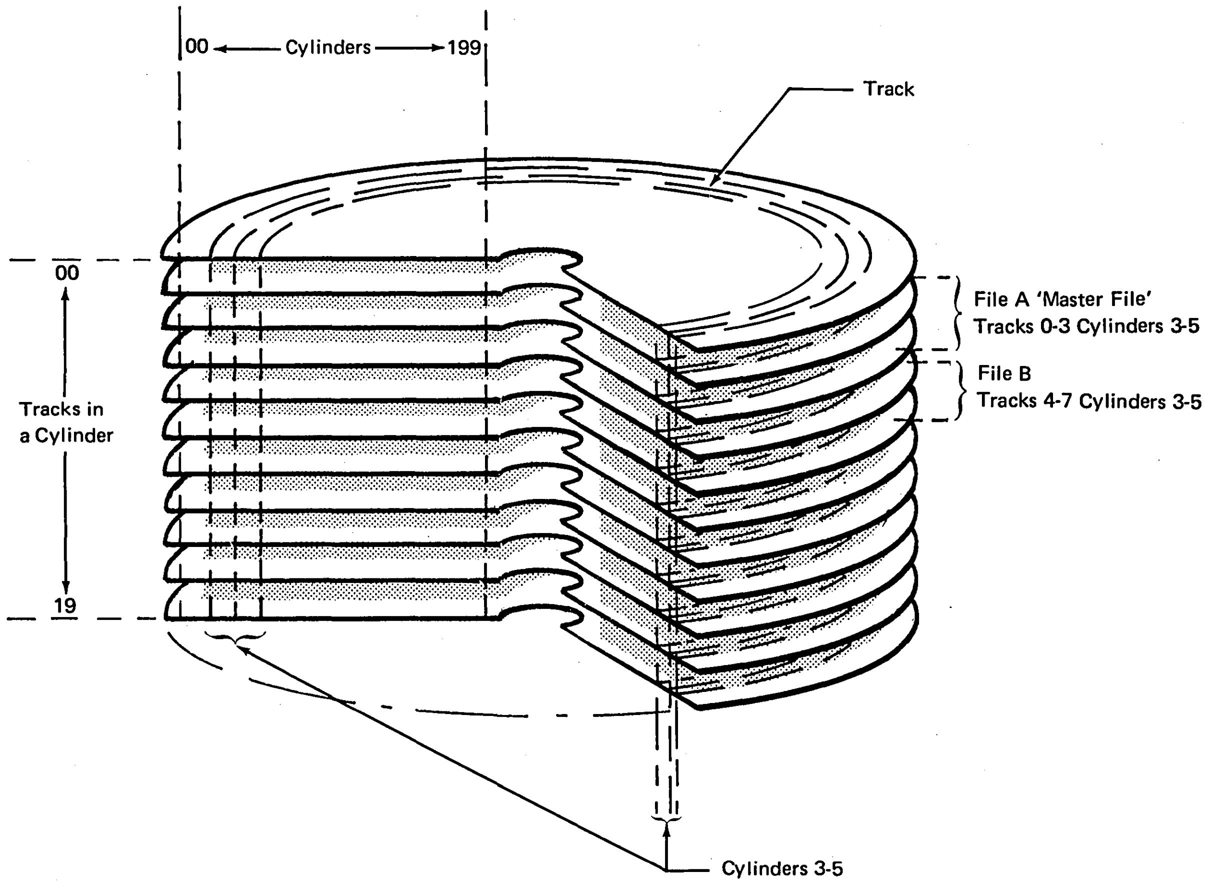


Figure 29. Cylinder Concept on the IBM 5445 Showing Split Cylinder Capability

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 these problems, the 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 until the file label is checked.

The volume label defines the characteristics of the volume. Some typical fields in the volume label are the volume serial number, owner identification, and (for 5444 only) available tracks.

To use a particular disk file required in a program, the operator must use OCL statements to provide information that the system uses to verify that the correct pack is mounted and that the required disk file or disk area is available.

CHAPTER 8. STORING PROGRAMS AND PROCEDURES ON DISK

In the IBM System/3 Model 6, Model 10 Disk System, and Model 15, programs and OCL statements can be stored on an IBM 5444 Disk Storage Drive and transferred as needed into main storage. (This chapter does not apply to IBM 5445 Disk Storage, which can not be used to store programs of OCL statements.)

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 and routines; 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.

Two library functions 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.

A COPY control statement is coded as follows:

```
// COPY FROM-R1,TO-F1,LIBRARY-ALL,NAME-PAY.ALL
```

Object Library Expansion: Object libraries can be expanded for temporary entries. When you copy an object program to 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 expansion capability 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, 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 removable disks 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 you are processing multivolume files. The pack containing a program cannot be removed until the program run is completed.

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 one 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 follow the object library (Figure 30). *Note:* The Library Maintenance program will move the object library to allow space for the source library which must precede 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.

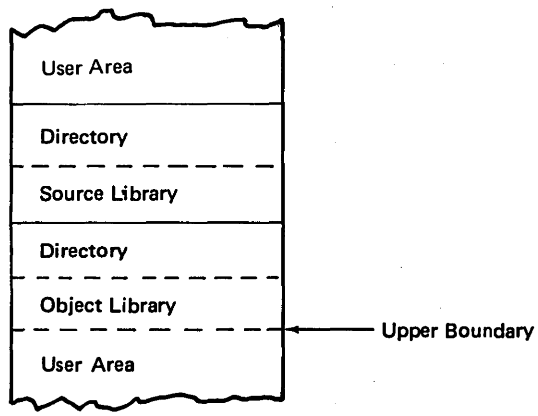


Figure 30. Relative Positions of Libraries on Disk

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 31 shows the format of the source library.

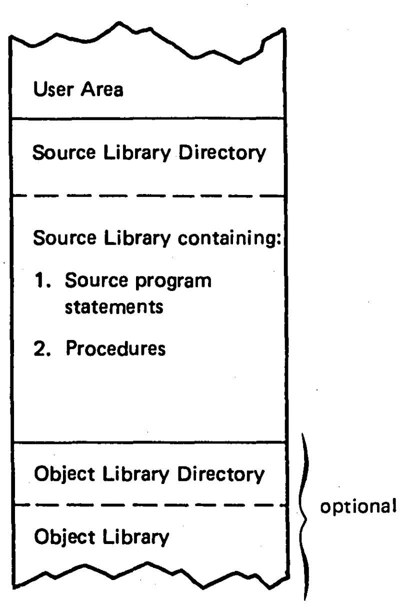
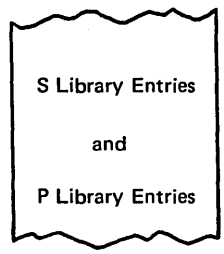


Figure 31. Format of the Source Library

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 32 shows the logical entries within the source library.

Source Library



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

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

Figure 32. 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 31. 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 (programs and procedures) within the source library need not be stored in consecutive sectors. An entry can be stored in widely separated sectors. Within each sector is a pointer to the sector that contains the next part of the entry.

The boundaries 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 used, the duplicate characters are reinserted.

Object Libraries

The object library is a disk area used to store object programs and routines. Object programs (executable rpograms) are programs and subroutines that can be loaded for execution. Routines (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 33 is a sample object library.

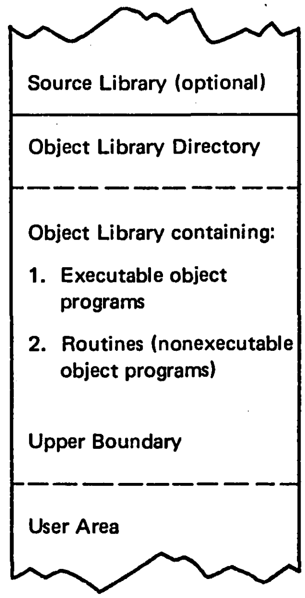
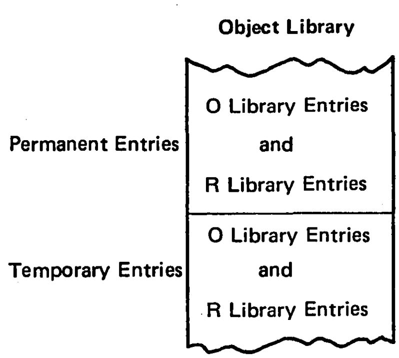


Figure 33. Format of the Object Library

The object library is an area on disk 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 34 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.

Figure 34. Logical Parts of an Object Library

Physical Characteristics of the Object Library

Size: You can build an object library an any 5444 disk pack, but you must have one library online containing the system programs. The minimum size of an object library 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 is not included in the number of tracks you specify for the library; the Library Maintenance program calculates and assigns that additional space for you.

The amount of additional space needed depends on the capacity of your system and whether you have the Roll-Out/Roll-In or Checkpoint/Restart capability, or the dual programming Feature. For Model 6, you may need from two to nine additional tracks; for Model 10, you may need from two to 17 additional tracks; for Model 15, you may need from four to 15 additional tracks. For more information, refer to the appropriate reference manual (as described in the Preface of this manual).

Directory: The Library Maintenance program creates a directory for every object library (Figure 33). 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. The directory size is overidden if the operand specifying the size of the object library directory is coded.

Upper Boundary: The upper boundary of the object library (Figure 33) will automatically expand 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 also place your files toward the end of the pack to leave 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, making more sectors available for use.

Frequent adding, replacing, and deleting of entries may result in unused sectors. You can determine how many sectors are available by printing the system directory using the Library Maintenance program.

Storing Programs and Procedures into Libraries

You can use any of three methods to store programs into libraries: the Library Maintenance program, a specification of the RPG II Control Card sheet, FORTRAN or COBOL Process statement, or the COMPILE OCL statement.

Library Maintenance Program

Depending on your specifications, the Library Maintenance program can:

- Allocate space for a library; 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 or from one library to another (giving new names if requested), from the input device to a library, from a file to a library, from a library to a printer, or from a library to a punch.
- Rename library entries.
- Modify source library entries.

For information on the specifications necessary to perform these functions, refer to the *IBM System/3 Model 10 Disk System Control Programming Reference Manual,* GC21-7512, the *IBM System/3 Model 15 System Control Programming Reference Manual,* GC21-5077, 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 Sheet

You can use RPG II to indicate the type of object program 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. Programs written temporarily in the object library are all overlaid by the next program written permanently in the object library; a single program will be overlaid by the next program of the same name written temporarily in the object library. A program written permanently in the object library is placed in the smallest gap large enough to hold it. A program written temporarily in the object library by RPG II is written at the end of the last temporary entry in the library. The object program is written in the object library that contains 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 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 Model 10 Disk System Control Programming Reference Manual,* GC21-7512, the *IBM System/3 Model 15 System Control Programming Reference Manual,* GC21-5077, 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.

This appendix describes the factors to consider when determining how much disk space a file will require. In some instances, the calculations are different for the IBM 5444 than for the IBM 5445, in which case the calculations are illustrated separately.

Determining Number of Records in a File

To determine the disk space required for a file, 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 also depends upon whether your file organization method is sequential, indexed, or direct. If an indexed file, a sequential file, and a direct file all 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 (the same number of records), 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 file 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
860,160	Total characters in the file

Note: FORTRAN formatted sequential files must have a record length of 16, 32, 64, 128, or 256 bytes. FORTRAN unformatted sequential files have a record length calculated as follows: divide the record length by 248 and round the result up to the next whole number. Multiply that number by 256 to get the storage space required for each record on disk. (The length descriptor for each sector is 8 bytes, which reduces the available data space from 256 bytes — the sector size — to 248 bytes.)

Determining How Many Tracks are Needed — 5444

To store your file on disk, you must determine how many tracks will be needed for that file. Since a track on the 5444 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 calaulation is:

	140	Tracks required
Characters in a track	6144)860160	Characters in the file

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 one or more records.

Determining How Many Tracks are Needed — 5445

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. If the file contains 6720 records and each record contains 128 characters, the number of characters in the file is 860,160. To find the number of tracks this file would require on the 5445, the calculation is:

	168	Tracks required
Characters in a track	5120) 860160	Characters in the file

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 one or more records.

Calculating Index Space – 5444

If the file is indexed, you must also determine the amount of space for the file index.

Note: FORTRAN does not support indexed files.

To find the space needed for the file 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 on the application, but disk addresses are always three characters long. Thus, the size of an index entry is the key field length plus 3.

Index 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 index entry length. For the name and address example (index entry length is 9), this calculation is:

	28	Entries in a Sector
Index 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 that can be contained in a sector into the number of records. In our example, this calculation is:

	240	Sectors
Entries in a Sector 28	) 6720	Records

You must then add one sector to the result; this sector will serve as a delimiter. The result of this calculation (240 + 1 = 241 in this example) specifies how many sectors are needed for the index. If you plan to add to the file at a later time, you must include a minimum of two additional sectors in the final size of the index. One of these sectors is used as a delimiter for the added key area. The other (possibly more than one other) sector is used to temporarily store the added keys, until they are inserted into the original index area at EOF.

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

```
   10+1 = 11 Tracks
24)241     Sectors needed
   240
     1
```

In this example, since there is a remainder, the quotient should be rounded up to the next higher number (11) in order to reserve enough space for the index. Thus, in this example, 11 tracks will be required to contain 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 151 tracks.

140 (records) + 11 (index) = 151

Calculating Index Space — 5445

If your file is indexed, you must determine the amount of space needed for the file index.

Note: FORTRAN does not support indexed files.

Index space, like file space, is 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 index entry is the key field length plus 4.

Index 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 length 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 (the number of characters per sector) by the index entry length. Thus, if the index entry length is 10, this calculation is:

```
                              25      Entries in a Sector
Index 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 that can be contained in a sector into the number of records.

Since this calculation has a remainder, one sector should be added to your quotient so that enough sectors will be reserved for all the index entries.
In our example, this calculation is:

```
                              268 + 1  = 269 Sectors
Entries in a Sector      25 ) 6720              Records
                              50
                              172
                              150
                               220
                               200
                                20              Remainder
```

Then, add one more sector to your total; this sector serves as a delimiter. Thus, 270 sectors are needed for the index in this example. If you plan to add to the file at a later time, you must include a minimum of two additional sectors in the final size of the index. One of these sectors is used as a delimiter for the added key area. The other (possibly more than one other) sector is used to temporarily store the added keys until they are inserted into the original index area at EOF.

There are 20 sectors in a track on the 5445, so to find the number of tracks required, divide the number of sectors by 20. In this example, there is a remainder of 10; therefore, you should add one track to your answer. Otherwise, not enough space will be reserved for the index.

```
       13 + 1   = 14 Tracks
20 ) 270          Sectors needed
     20
      70
      60
      10           Remainder
```

For this example, 14 tracks are needed for the index. For information on how to calculate the disk track index (5445) see Appendix B.

File Size

The file size (number of records in a file), the length of the records in the file, and whether or not a file index is used determine the physical size of the file and whether the file needs to be multivolume. The number of records in a file also affects sequential processing and loading, as well as key sort.

When loading an indexed file, you can specify either the number of records in the file, or the number of tracks. When you specify the number of records, the system determines the number of data tracks, the number of file index tracks, and the number of disk track index tracks by computing record storage requirements, and then computing index storage requirements. When you specify the number of tracks, the system determines how the specified space is to be split between data tracks, file index tracks, and disk track index tracks. Figure 35 illustrates how the system splits an area on the 5445, when the TRACKS parameter is used in the OCL statement.

Number of Tracks	Key Length	Record Length	Disk Track Index	File Index	Data	Number of Keys	Number of Data Records
5	5	64		1	4	560	320
5	5	128		1	4	560	160
5	5	256		1	4	560	80
5	10	64		1	4	360	320
5	10	128		1	4	360	160
5	10	256		1	4	360	80
10	5	64		2	8	1120	640
10	5	128		1	9	560	360
10	5	256		1	9	560	180
10	10	64		2	8	720	640
10	10	128		1	9	360	360
10	10	256		1	9	360	180
50	5	64		7	43	3920	3440
50	5	128		4	46	2240	1840
50	5	256		2	48	1120	960
50	10	64		9	41	3240	3280
50	10	128		5	45	1800	1800
* 50	10	256		3	47	1080	940
100	5	64		13	87	7280	6960
100	5	128		7	93	3920	3720
100	5	256		4	96	2240	1920
100	10	64	1	19	80	6840	6400
100	10	128		10	90	3600	3600
100	10	256		6	94	2160	1880
500	5	64	1	63	436	35280	34880
500	5	128	1	34	465	19040	18600
500	5	256	1	18	481	10080	9620
500	10	64	1	91	408	32760	32640
500	10	128	1	50	449	18000	17960
500	10	256	1	27	472	9720	9440
1000	5	64	1	125	874	70000	69920
1000	5	128	1	67	932	37520	37280
1000	5	256	1	35	964	19600	19280
1000	10	64	1	182	817	65520	65360
1000	10	128	1	100	899	36000	35960
1000	10	256	1	53	946	19080	18920
2000	5	64	1	250	1749	140000	139920
2000	5	128	1	134	1865	75040	74600
2000	5	256	1	69	1930	38640	38600
2000	10	64	2	364	1634	131040	130720
2000	10	128	1	200	1799	72000	71960
2000	10	256	1	106	1893	38160	37860
3000	5	64	1	375	2624	210000	209920
3000	5	128	1	200	2799	112000	111960
3000	5	256	1	104	2895	58240	57900
3000	10	64	2	546	2452	196560	196160
3000	10	128	1	300	2699	108000	107960
3000	10	256	1	158	2841	56880	56820
3980	5	64	1	498	3481	278880	278480
3980	5	128	1	266	3713	148960	148520
3980	5	256	1	138	3841	77280	76820
3980	10	64	3	724	3253	260640	260240
3980	10	128	2	398	3580	143280	143200
3980	10	256	1	210	3769	75600	75380

Figure 35. Sample Record Capacities of Indexed Files on a 5445 Disk if TRACKS Parameter is Used in an OCL Statement

Note: The smaller of the 'Number of Keys' and 'Number of Data Records' entries for a given example represents the upper limit of the capacity of the file for that example.

* For example, given that TRACKS is specified as 50, the key length is specified as 10, and the record length is specified as 256; then we can see from the underlined portion of Figure 35 that:

- No disk track index is required (because the file index is not more than 15 tracks).
- Of the 50 tracks, 3 are used for index and 47 are used for data.
- The 3 index tracks can accommodate 1080 keys.
- The 47 data tracks can accommodate 940 records.

Figure 36 shows how many keys can be contained in one track of file index. Track capacity depends on key length.

Keylength	Number of Keys Per Index Track 5444	5445
1	1536	1020
2	1224	840
3	1008	720
4	864	640
5	768	560
6	672	500
7	600	460
8	552	420
9	504	380
10	456	360
11	432	340
12	408	320
13	384	300
14	360	280
15	336	260
16	312	240
17	288	240
18	288	220
19	264	220
20	264	200
21	240	200
22	240	180
23	216	180
24	216	180
25	216	160
26	192	160
27	192	160
28	192	160
29	192	140

Figure 36. Keys per Index Track

Figure 37 shows the number of tracks needed to store a given number of records, using various record lengths. This information may prove useful in planning file requirements.

Disk Requirements for Data Records (Number of tracks required; does not include indexes)

Number of	Rec-Lth – 50		Rec-Lth – 64		Rec-Lth – 100		Rec-Lth – 128		Rec-Lth – 256	
Records	5444	5445	5444	5445	5444	5445	5444	5445	5444	5445
500	5	5	6	7	9	10	11	13	21	25
1000	9	10	11	13	17	20	21	25	42	50
1500	13	15	16	19	25	30	32	38	63	75
2000	17	20	21	25	33	40	42	50	84	100
2500	21	25	27	32	41	49	53	63	105	125
3000	25	30	32	38	49	59	63	75	125	150
3500	29	35	37	44	57	69	73	88	146	175
4000	33	40	42	50	66	79	84	100	167	200
4500	37	44	47	57	74	88	94	113	188	225
5000	41	49	53	63	82	98	105	125	209	250
5500	45	54	58	69	90	108	115	138	230	275
6000	49	59	63	75	98	118	125	150	250	300
6500	53	64	68	82	106	127	136	163	271	325
7000	57	69	73	88	114	137	146	175	292	350
7500	62	74	79	94	123	147	157	188	313	375
8000	66	79	84	100	131	157	167	200	334	400
8500	70	84	89	107	139	167	178	213	355	425
9000	74	88	94	113	147	176	188	225	375	450
9500	78	93	99	119	155	186	198	238	396	475
10000	82	98	105	125	163	196	209	250	417	500
10500	86	103	110	132	171	206	219	263	438	525
11000	90	108	115	138	180	215	230	275	459	550
11500	94	113	120	144	188	225	240	288	480	575
12000	98	118	125	150	196	235	250	300	500	600
12500	102	123	131	157	204	245	261	313	521	625
13000	106	127	136	163	212	254	271	325	542	650
13500	110	132	141	169	220	264	282	338	563	675
14000	114	137	146	175	228	274	292	350	584	700
14500	119	142	152	182	237	284	303	363	605	725
15000	123	147	157	188	245	293	313	375	625	750
15500	127	152	162	194	253	303	323	388	646	775
16000	131	157	167	200	261	313	334	400	667	800
16500	135	162	172	207	269	323	344	413	688	825
17000	139	167	178	213	277	333	355	425	709	850
17500	143	171	183	219	285	342	365	438	730	875
18000	147	176	188	225	293	352	375	450	750	900
18500	151	181	193	232	302	362	386	463	771	925
19000	155	186	198	238	310	372	396	475	792	950
19500	159	191	204	244	318	381	407	488	813	975
20000	163	196	209	250	326	391	417	500	834	1000

Figure 37 (1 of 2). Disk Requirements for Data Records (number of records varies from 500 to 20000)

Disk Requirements for Data Records (Number of tracks required; does not include indexes)

Number of	Rec-Lth — 50		Rec-Lth — 64		Rec-Lth — 100		Rec-Lth — 128		Rec-Lth — 256	
Records	5444	5445	5444	5445	5444	5445	5444	5445	5444	5445
1000	9	10	11	13	17	20	21	25	42	50
2000	17	20	21	25	33	40	42	50	84	100
3000	25	30	32	38	49	59	63	75	125	150
4000	33	40	42	50	66	79	84	100	167	200
5000	41	49	53	63	82	98	105	125	209	250
6000	49	59	63	75	98	118	125	150	250	300
7000	57	69	73	88	114	137	146	175	292	350
8000	66	79	84	100	131	157	167	200	334	400
9000	74	88	94	113	147	176	188	225	375	450
10000	82	98	105	125	163	196	209	250	417	500
11000	90	108	115	138	180	215	230	275	459	550
12000	98	118	125	150	196	235	250	300	500	600
13000	106	127	136	163	212	254	271	325	542	650
14000	114	137	146	175	228	274	292	350	584	700
15000	123	147	157	188	245	293	313	375	625	750
16000	131	157	167	200	261	313	334	400	667	800
17000	139	167	178	213	277	333	355	425	709	850
18000	147	176	188	225	293	352	375	450	750	900
19000	155	186	198	238	310	372	396	475	792	950
20000	163	196	209	250	326	391	417	500	834	1000
21000	171	206	219	263	342	411	438	525	875	1050
22000	180	215	230	275	359	430	459	550	917	1100
23000	188	225	240	288	375	450	480	575	959	1150
24000	196	235	250	300	391	469	500	600	1000	1200
25000	204	245	261	313	407	489	521	625	1042	1250
26000	212	254	271	325	424	508	542	650	1084	1300
27000	220	264	282	338	440	528	563	675	1125	1350
28000	228	274	292	350	456	547	584	700	1167	1400
29000	237	284	303	363	473	567	605	725	1209	1450
30000	245	293	313	375	489	586	625	750	1250	1500
31000	253	303	323	388	505	606	646	775	1292	1550
32000	261	313	334	400	521	625	667	800	1334	1600
33000	269	323	344	413	538	645	688	825	1375	1650
34000	277	333	355	425	554	665	709	850	1417	1700
35000	285	342	365	438	570	684	730	875	1459	1750
36000	293	352	375	450	586	704	750	900	1500	1800
37000	302	362	386	463	603	723	771	925	1542	1850
38000	310	372	396	475	619	743	792	950	1584	1900
39000	318	381	407	488	635	762	813	975	1625	1950
40000	326	391	417	500	652	782	834	1000	1667	2000
41000	334	401	428	513	668	801	855	1025	1709	2050
42000	342	411	438	525	684	821	875	1050	1750	2100
43000	350	420	448	538	700	840	896	1075	1792	2150
44000	359	430	459	550	717	860	917	1100	1834	2200
45000	367	440	469	563	733	879	938	1125	1875	2250
46000	375	450	480	575	749	899	959	1150	1917	2300
47000	383	459	490	588	765	918	980	1175	1959	2350
48000	391	469	500	600	782	938	1000	1200	2000	2400
49000	399	479	511	613	798	958	1021	1225	2042	2450
50000	407	489	521	625	814	977	1042	1250	2084	2500
75000	611	733	782	938	1221	1465	1563	1875	3125	3750
100000	814	977	1042	1250	1628	1954	2084	2500	4167	5000
125000	1018	1221	1303	1563	2035	2442	2605	3125	5209	6250
150000	1221	1465	1563	1875	2442	2930	3125	3750	6250	7500
175000	1425	1709	1823	2188	2849	3418	3646	4375	7292	8750
200000	1628	1954	2084	2500	3256	3907	4167	5000	8334	10000

Figure 37 (Part 2 of 2). Disk Requirements for Data Records (number of records varies from 1000 to 200,000).

Calculating Disk File Sizes — Summary

This section contains step-by-step explanations of some common calculations.

Determining the Number of Tracks in a Sequential or Direct File (5444)

1. number of records x record length = number of characters

2. $\frac{\text{number of characters (from step 1)}}{\text{6144 (number of characters/track)}}$ = number of tracks (round to the next higher whole number)

Determining the Number of Tracks in a Sequential or Direct File (5445)

1. number of records x record length = number of characters

2. $\frac{\text{number of characters (from step 1)}}{\text{5120 (number of characters/track)}}$ = number of tracks (round to the next higher whole number)

Determining the Number of Tracks in an Indexed File (5444)

To determine the number of data tracks in an indexed file, the following two steps should be used:

1. number of records x record length = number of characters

2. $\frac{\text{number of characters (from step 1)}}{\text{6144 (number of characters/track)}}$ = number of data tracks (round to the next higher whole number)

The following four steps should then be used to determine the number of file index tracks in an indexed file:

1. key field length + 3 = index entry length

2. $\frac{\text{256 (number of characters/sector)}}{\text{index entry length (from step 1)}}$ = number of entries per sector (drop remainder)

3. $\frac{\text{number of records}}{\text{number of entries per sector (from step 2)}}$ = number of sectors (round to the next higher whole number; then, add one sector for a delimiter, and two or more additional sectors if you plan to add records to the file later)

4. $\frac{\text{number of sectors (from step 3)}}{\text{24 (number of sectors/track)}}$ = number of index tracks (round to the next higher whole number)

Determining the Number of Tracks in an Indexed File (5445)

To determine the number of data tracks in an indexed file, the following two steps should be used:

1. number of records x record length = number of characters

2. $\frac{\text{number of characters (from step 1)}}{\text{5120 (number of characters/track)}}$ = number of data tracks (round to the next higher whole number).

The following four steps should then be followed to determine the number of file index tracks in an indexed file:

1. key field length + 4 = index length

2. $\frac{\text{256 (number of characters/sector)}}{\text{index entry length (from step 1)}}$ =number of entries per sector (drop remainder)

3. $\frac{\text{number of records}}{\text{number of entries per sector (from step 2)}}$ = number of sectors (round to the next higher whole number; then, add one sector for a delimiter, and two or more additional sectors if you plan to add records to the file later)

4. $\frac{\text{number of sectors (from step 3)}}{\text{20 (number of sectors/track)}}$ = number of index tracks (round the next higher whole number)

Determining the Number of Tracks of Disk Track Index (5445)

If an indexed 5445 file has more than 15 index tracks (from step 4 above), the file will have a disk track index in addition to the file index. The following two steps should be used to determine the number of tracks needed for the disk track index:

1. $\frac{\text{number of index tracks (greater than 15)}}{\text{number of entries per sector (from step 2 above)}}$ = number of sectors (round to the next higher whole number)

2. $\frac{\text{number of sectors (from step 1)}}{20}$ = number of disk track index tracks (round results to the next higher whole number)

The total number of tracks in a 5445 indexed file can be determined by adding the number of data tracks, the number of file index tracks, and the number of disk track index tracks.

Converting Cylinder/Track to Track Number

To convert cylinder/track to track number, multiply cylinder number by the number of tracks on each cylinder and add track number.

EXAMPLES:	*5444*	*5445*
	6/1 = cylinder track	5/3 = cylinder/track
	6 x 2 + 1 = 13	5 x 20 + 3 = 103
	13 = track number	103 = track number

Converting Track Number to Cylinder/Track

To convert track number to cylinder/track, divide track number by the number of tracks on a cylinder. The quotient is the cylinder and the remainder is the track.

EXAMPLES:	*5444*	*5445*
	13 = track number	103 = track number
	13 ÷ 2 = 6 (remainder 1)	103 ÷ 20 = 5 (remainder 3)
	6/1 is the cylinder track	5/3 is the cylinder/track.

APPENDIX B. PERFORMANCE CONSIDERATIONS FOR PROCESSING INDEXED FILES

Many factors affect the performance of a program that processes indexed files using the System/3 Disk Systems, Model 6, Model 10, or Model 15.

Note: In this section, references to the IBM 5444 Disk Storage Drive apply to Models 6, 10, and 15 unless specifically noted otherwise; references to IBM 5445 Disk Storage apply only to the Models 10 and 15.

Since you can control most of the factors discussed in this appendix, with proper planning you can obtain optimum results. However, no single approach will produce optimum results for all users. An understanding of the factors presented in this appendix will help you adapt your processing techniques for maximum throughput.

Figure 38 describes a sample program run a number of times using different combinations of some of the performance factors. This example reflects performance of a program that randomly adds records to an indexed file, using the 5445 on a System/3 Model 10 Disk System. Figure 39 describes several other performance factors that remained stable (as specified) for the runs described in Figure 38. These factors which should be considered when planning for optimum performance, are discussed later in this appendix.

	Run 1	Run 2	Run 3	Run 4	Run 5
Disk Track Index (22-byte core index) Used:	No	Yes	Yes	Yes	Yes
Work File for Key Sort/Merge:	No	No	Yes	No	Yes
Pre-Sorted Input:	No	No	No	Yes	Yes
Total Job Time (in minutes)	72	50	40	24	13

Figure 38. Performance Achieved with Sample Program Under Various Conditions.

Programming Considerations

- Buffered I/O: not used
- Shared I/O: not used (cannot be used with 5445 files)
- Type of processing: random update with additions, using CHAIN
- Highest added key save area used: yes
- Other data: no overlays; minimal processing; version 7 of Model 10 Disk System SCP and RPG II; minimal printing; 24K dedicated system; total time includes OCL processing; 79 RPG II source statements, including 19 detail calculations specifications

File Considerations

- Key length: 10 bytes
- Record length: 96 bytes
- Block length: 384 bytes
- File size: 25,000 records
- Location of files: indexed file on D1; work file for key sort ($INDEX45) on D2; added records on MFCU (Model 2; 500 cards per minute)
- Number of records added: 1500 (from 1500 cards)
- Distribution of added records: evenly throughout the file

Figure 39. Characteristics of Environment for Performance Test

Indexes

Indexes are defined as follows:

- The *core index* is located in main storage. The length of the core index is specified by the programmer.
- The *disk file index* (or simply the file index) is located on the disk storage device, and precedes the data records (see Chapter 3 for more information).
- The *disk track index* is located on an IBM 5445 Disk Storage drive, immediately preceding the file index. A disk track index is generated by the system when an indexed file with more than 15 tracks of file index is loaded.

Figure 40 shows the relationship between these index types when using the 5445.

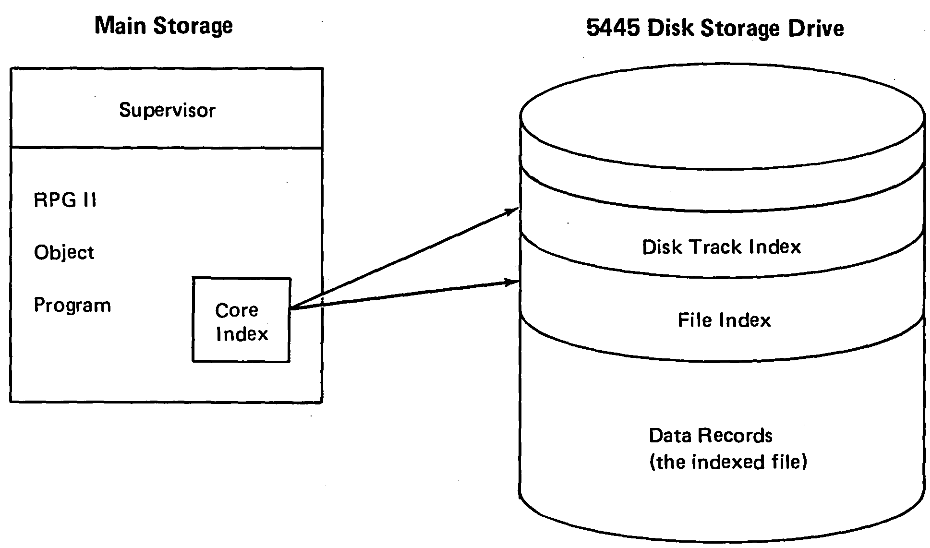


Figure 40. Relationship of Indexes

Core Index

The core index is a table containing entries for tracks in the index portion of a data file. Each entry contains a track address and the lowest key field associated with the next track. Figure 41 shows the layout on disk of the index for the indexed file, INDEXT, which contains 1000 records. Since all index entries are contained on three tracks, the core index for INDEXT shown in Figure 42 contains only three entries, one per track. Each core index entry contains the low key on the next track and the track address.

Columns 60-65 of the RPG II File Description Specifications sheet are used to specify the number of bytes you want to reserve for the core index and a highest added key save area (discussed later in this section). Using the amount of core storage you specify, the system builds the most efficient core index it can. The core index is built immediately before your RPG II program is executed. A core index can be specified for more than one file used in a program; note, however, that core index cannot be used with shared I/O.

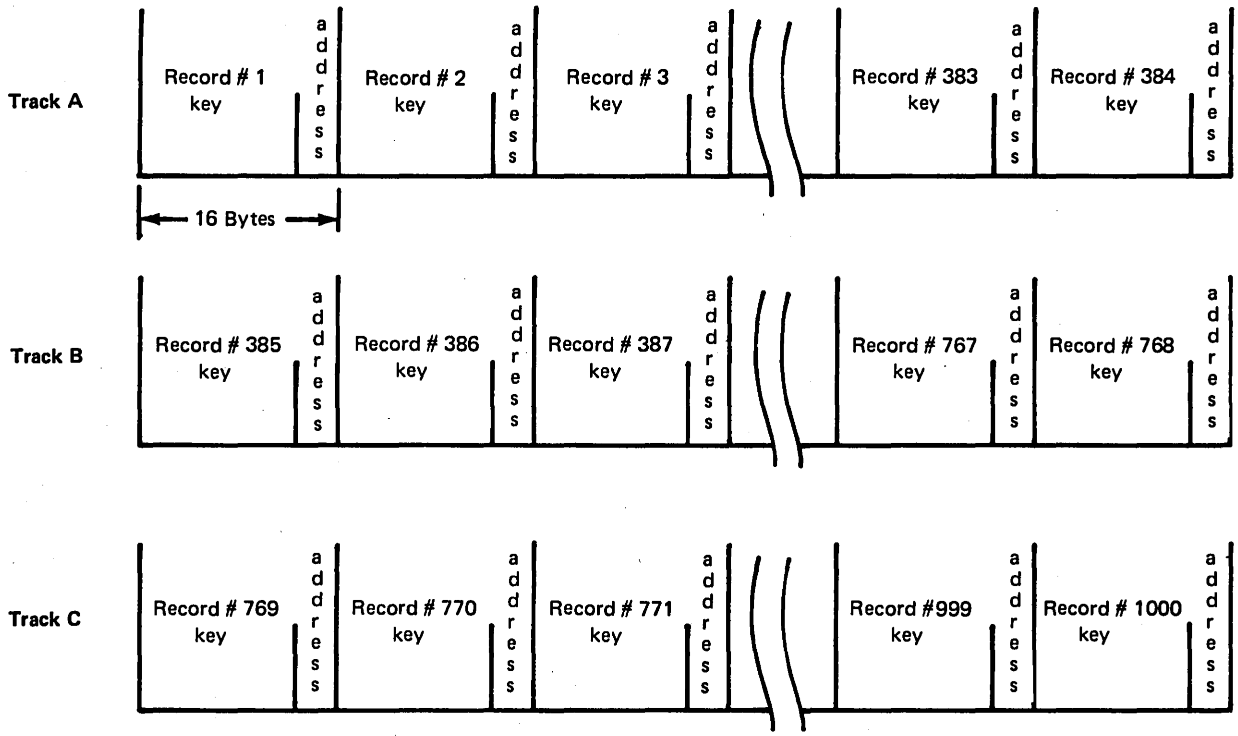


Figure 41. Disk Layout of the Index for INDEXT

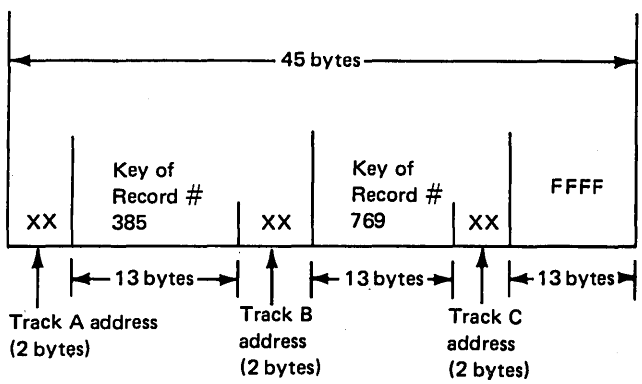


Figure 42. Core Index for INDEXT

Use of the core index can significantly reduce the amount of time needed to process an indexed file because it enables the system to go more directly to the specific record you want. With the core index, the system can find a specific record by searching only a small part of the file index.

Without the core index, if the next key is lower than the last key, all index entries that precede the desired record must be searched. Using the core index shown in Figure 42, the system finds record 767 in this manner:

1. The core index is searched until the first key field higher than record 767 is located. In this instance the key is 769, on track *C.* Since 769 is the low key on track *C,* key 767 must reside on track *B.*

2. Track *B* in the file index is searched until key 767 is located.

3. Then, the system chains directly to the associated data record.

Figures 43 and 44 show the number of bytes of main storage required for a core index that provides the most efficient random processing of an indexed file (on a 5444 or 5445), using key length and number of records as variables.

	Number of Records (in 1000's)					
Key Length	**2**	**5**	**8**	**10**	**15**	**20**
20	176	418	682	836	1254	1672
19	168	399	651	798	1197	1596
18	140	360	560	700	1060	1400
17	133	342	532	665	1007	1330
16	126	306	468	594	882	1170
15	102	255	408	510	765	1020
14	96	224	368	448	672	896
13	90	210	315	405	600	795
12	70	182	280	350	518	700
11	65	156	247	312	455	611
10	60	132	216	264	396	528
9	44	110	176	220	330	440
8	40	100	150	190	280	370
7	36	81	126	153	225	306
6	24	64	96	120	184	240
5	21	49	77	98	140	189
4	18	36	60	72	108	144

Figure 43. Core Index Sizes for 5444 Single Volume Indexed Files Without Additions

	Number of Records (in 1000's)					
Key Length	**2**	**5**	**8**	**10**	**15**	**20**
20	220	550	880	1100	1650	2200
19	210	483	777	966	1449	1911
18	200	460	740	920	1380	1820
17	171	399	646	798	1197	1596
16	162	378	612	756	1134	1512
15	136	340	527	663	986	1309
14	128	288	464	576	864	1152
13	105	255	405	510	750	1005
12	98	224	350	448	658	882
11	78	195	312	390	585	767
10	72	168	276	336	504	672
9	66	154	242	297	440	583
8	50	120	200	240	360	480
7	45	99	162	198	297	396
6	32	80	128	160	240	320
5	28	63	105	126	189	252
4	24	48	78	96	144	192

Figure 44. Core Index Sizes for 5445 Single Volume Indexed Files Without Additions

Note: To adapt this figure to apply to processing with additions, add one keylength to the computed core index sizes (Model 10 only).

Figure 45 shows the relative number of tracks required when the record length and number of records are variables.

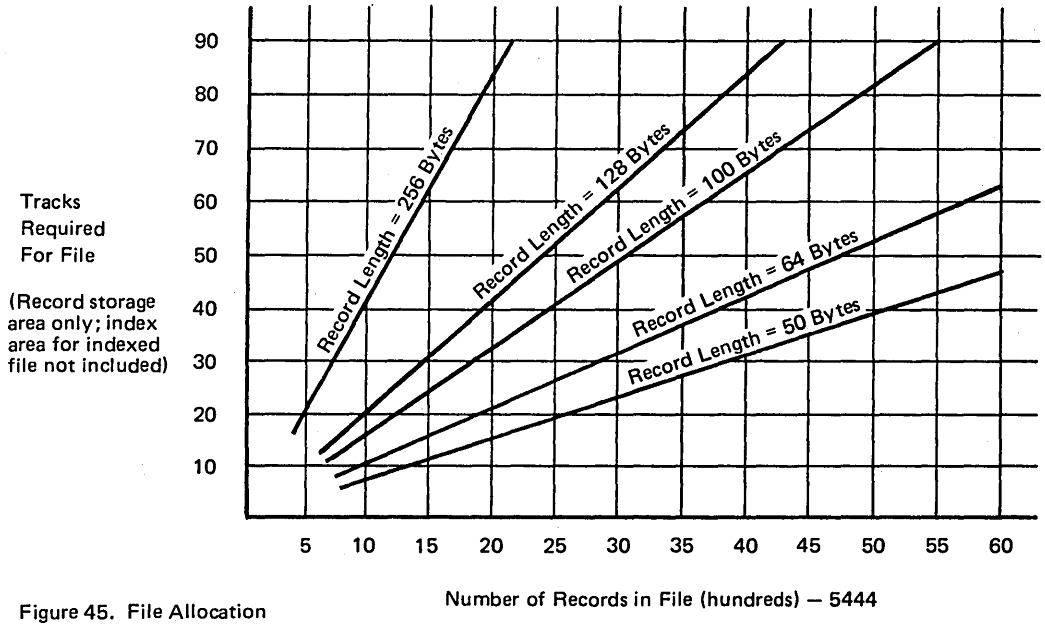


Figure 45. File Allocation

Core Index Utilization

A core index entry (for either 5444 or 5445 files) contains a track address and the lowest key field associated with the next track. The format of a core index entry is:

C	H	Key field

Where C is the cylinder number (one byte)
H is the head (track) number (one byte)

The address (C-H) points to a track in the file index or (for 5445 files) to a track in the disk track index. The system analyzes the index (on disk) to determine which kind of index it is.

The core index is constructed before execution of the object program. The number of entries the core index contains depends on factors such as keylength and number of tracks in the file index and/or disk track index. (The term *keylength* refers to the number of bytes in the key associated with the indexed file.) When the system analyzes the core index area to determine its optimum use, it looks at the logical file size rather that at the physical file size specified.

In the following section is a discussion of the most efficient core index size and the smallest usable core index. Since the user is not required to provide a core index entry, for single volume files, the smallest core index is 0 entries. Multivolume files will always default to the minimum core index size. In the following discussion, smallest core index refers to the smallest usable core index that can still provide a performance advantage, as specified in your program. Core index utilization is discussed in this section.

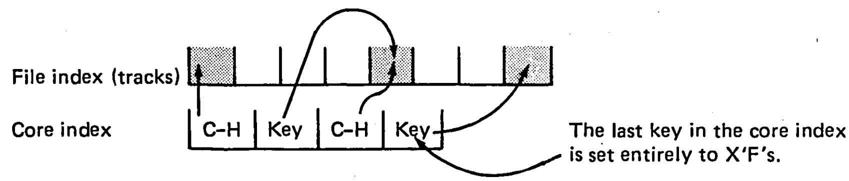
Note: FORTRAN does not support indexed files; Model 10 COBOL does not support multivolume indexed files.

Processing 5444 Single Volume Files

The most efficient core index for this type of file would contain one entry for every track of file index. Its size is computed as follows:

(keylength + 2) x (number of tracks in the file index)

Since only one core index entry would provide no advantage for 5444 files (and, for RPG II, the system would not build a core index if there was room for only one entry), the smallest core index you should specify is two entries, one pointing to the midpoint of the logical file index, and the other pointing to the logical end of the file index:



Processing 5444 Multivolume Files — Online

Since all volumes are online for this type of file, all records are available for processing, and the most efficient core index would contain one entry for every track of file index on all volumes. For example, if volume 1 contained 30 tracks of the file index, volume 2 contained 25 tracks of the file index, and volume 3 contained 25 tracks of the file index, then the core index providing the best performance would be computed as follows:

(keylength + 2) x (30 + 25 + 25)

Note that this calcuation is based on the number of tracks of file index actually containing keys, rather than on the number of tracks allocated.

The smallest core index allowed is one entry for each possible online volume (i.e., 4 entries). When using RPG II, at least the minimum number of entries is required and therefore will be supplied, as a default value, if no core index is specified on the RPG II File Description Specifications sheet.

Processing 5444 Multivolume Files — Offline

Since each volume is processed individually, the most efficient core index for this type of file would be one entry for each track of file index contained in the volume which has the most tracks of file index. Its size is computed as follows:

(keylength + 2) x (greatest number of file index tracks in any volume used)

The smallest core index allowed is one entry for each possible online volume (i.e., 4 entries). When using RPG II, at least the minimum number of entries is required and therefore will be supplied, as a default value, if no core index is specified on the RPG II File Description Specifications sheet.

Processing 5445 Single Volume Files — (without additions on Model 10; with or without additions on Model 15)

The most efficient core index for this type of file would contain one entry for every track of file index. Its size would be computed as follows:

(keylength + 2) x (number of tracks)

In this case, the smallest core index you should specify is a single entry (keylength + 2). This minimum size core index will be used if the file index contains 16 or more tracks. The file will have a disk track index, and the single core index entry will point to the first track of this disk track index. If the file index contains fewer than 16 tracks, no disk track index exists and the single core index entry will not be used.

Processing 5445 Single Volume Files — (with additions on Model 10)

The most efficient core index for this type of file would contain one entry for every track of file index, plus one keylength to be used for the highest added key save area (discussed later in this section). This area is computed as follows:

[(keylength + 2) x (number of tracks)] + (keylength)

The smallest core index that you should specify will contain one entry plus one keylength to be used for the highest added key save area, computed as follows:

(keylength + 2) + keylength, or 2(keylength) + 2

The single entry will either be used to point to the start of the disk track index or will not be used at all. The system automatically makes this decision, depending on which approach will provide the best performance.

Processing 5445 Multivolume Files — Online (without additions on Model 10; with or without additions on Model 15)

Since all volumes are online, all records are available for processing. The most efficient core index for this type of file would contain one entry for every track of file index on all volumes, minus 2, computed as follows:

(keylength + 2) x [(total number of tracks of file index on all volumes) — (2)]

For example, if 150 tracks of file index on volume 1 are used, 20 tracks of file index on volume 2 are used, and the keylength is 10, the core index size that you should specify to provide the best performance is computed as follows:

(10 + 2) x [(150 + 20) - (2)] = 2016

Note: A single core index entry is automatically reserved for each volume; the core index size you specify will be in addition to this requirement.

The smallest core index that you should specify for this type of file would contain one entry per volume, computed as follows:

(keylength + 2) x (number of volumes)

Processing 5445 Multivolume Files — Online (with additions on Model 10)

The most efficient core index for this type of file is computed as in the preceding example. Remember that a 'highest added key save area' and a single core index entry are automatically reserved for each volume; the core index size you specify will be in addition to these requirements.

The smallest core index that you should specify will contain one entry for each volume, computed as follows:

(number of volumes) x [(2) (keylength) + 2]

Processing 5445 Multivolume Files — Offline (without additions on Model 10; with or without additions on Model 15)

Since each volume is processed individually, the most efficient core index for this type of file would be large enough to accommodate the volume with the greatest number of file index tracks. The size of such a core index would be computed as follows:

(keylength + 2) x (greatest number of file index tracks, –2)

A single core index entry is automatically reserved for each volume; the core index size you specify will be in addition to this requirement.

For this type of file, the smallest core index you should specify would contain a single entry (keylength + 2). In this case, the core index will be used if the file index contains 16 or more tracks. Under these circumstances, the file would have a disk track index, and the single core index would point to the first track of this disk track index. If the file contains fewer than 16 tracks, no disk track index would exist, and the core index entry would point to the first track of file index, and would contain the 'HIKEY' value.

Processing 5445 Multivolume Files — Offline (with additions on Model 10)

The most efficient and the smallest core indexes for these files are computed as described in the preceding example. The only difference between this example and the preceding one — processing with additions — is that in this example a 'highest added key save area' as well as one core index entry are always reserved for each volume.

File Index

The file index is part of the indexed file that you define using the OCL statement. The file index precedes the data records in the file, and contains an entry for each record in the data file. The formats of the file index entries for 5444 and 5445 files are shown below. Note that the disk addresses shown represent displacements from the start of the data area.

File Index Entry Format — 5444 Files

Key	C	S	D

Where C is the cylinder number (one byte)
S is the sector number (one byte)
D is the displacement within the sector (one byte)

The address (C-S-D) points to a data record in the indexed file.

File Index Entry Format — 5445 Files

Key	C	H	R	D

Where C is the cylinder number (one byte)
H is the head (track) number (one byte)
R is the record number (one byte)
D is the displacement within the sector (one byte)

The address (C-H-R-D) points to a data record in the indexed file.

See Chapter 3 for more information on file indexes.

Disk Track Index

The disk track index can be used only for indexed files on the 5445. If an indexed file on the 5445 has more than 15 tracks of file index, a disk track index will be built by the system when the file is loaded. This index precedes the file index and is part of the file as specified on the OCL statement. The disk track index contains one entry for each track of file index. When processing a multivolume file, if volume 1 has 4 tracks of file index and volume 2 has 50 tracks of file index, a disk track index will be produced only on volume 2.

When processing single volume 5445 indexed files on Model 10, the disk track index is not used unless a core index is specified in the program. When processing single volume 5445 indexed files on a Model 15, the disk track index is used whenever it is more efficient to do so. When processing a multivolume 5445 indexed file, RPG II provides two core index entries; an additional core index entry is used if a core index is specified in the program (see *Core Index*).

Disk Track Index Entry Format — 5445 only

Key	C	H	F	F

Where C is the cylinder number (one byte)
H is the head (track) number (one byte)
FF is a 2-byte-long filler (X'FFFF')

The X'FFFF' tells the program that this is a disk track index entry.
The address (C-H) points to a track in the file index.

The disk track index is used only when the system determines that its use will improve performance. In effect, it is an extension of the core index, and can be used only in conjunction with a core index. If the core index is large enough to contain an entry for every track, or every second, third, fourth, fifth, or sixth track of file index, then the disk track index will not be used. If the core index is large enough to contain an entry for only every group of seven or more tracks of file index, then the disk track index will be used. (See *Core Index* for more information on that subject.)

The size of the disk track index must be at least one track, which should be enough room for most files. The capacity of one track of disk track index varies according to keylength.

Keylength	Number of Entries in Disk Track Index	Capacity — Number of Records
5	560	313,600
10	360	129,600
15	260	67,600
20	200	40,000
25	160	25,600

For example, if your keylength is 10 bytes, a file of 129,000 records will require a disk track index of only 1 track and a file index of 360 tracks. If the file contains more than 129,600 records, a disk track index of 2 or more tracks will be required.

To calculate the number of tracks required for a disk track index, perform these calculations:

$$E = \frac{256}{\text{keylength} + 4} \quad = \text{ number of entries per sector (drop the remainder)}$$

$$N = \frac{\text{number of tracks of file index}}{E} \quad = \text{ number of sectors required}$$

$$T = \frac{N}{20} \quad = \text{ number of tracks required for the disk track index (round up to next whole number)}$$

For example, if your file contains 100,000 records (10-byte keys), the file index requires 278 tracks. The disk track index requires 0.77 tracks, or rounded upwards, 1 track, computed as follows:

$E = 256/(10 + 4) = 18.3$ entries per sector

$N = 278/18 = 15.4$ sectors

$T = 15.4/20 = 0.77$ tracks, rounded upwards to 1 track.

For more detailed information, see *Appendix A. Calculating Disk File Size.*

Type of Processing

The type of indexed file processing used, combined with other factors, greatly affects program performance. Figure 46 shows the different kinds of processing permitted by RPG II for indexed files, and indicates whether the other factors are related to each type of processing. Notice, for example, that core index is used only for random processing or for output with additions, while key sort routines are only used after adding records or after an unordered load.

Type of processing for indexed files	Other performance factors: Core index	Disk track index	Save area	Key sort	Work file/key sort	Location	Distribution	Number of records	Number of adds
Sequential input/update									
• By key, with additions				X	X	X	X	X	X
• By key, without additions						X		X	
• By limits						X		X	
Random input/update									
• By chaining, with additions	X	X	X	X	X	X	X	X	X
• By chaining, without additions	X	X				X		X	
• By ADDROUT	X	X				X		X	
Output									
• Unordered load (see note)				X	X	X	X	X	
• Ordered load						X		X	
• Additions only	X	X	X	X	X	X	X	X	X

X = Performance factor is applicable

Note: Work file/key sort is not used for an unordered load for models 6 or 10.

Figure 46. Applicability of Performance Factors to Type of Processing

Highest Added Key Save Area

Model 6 and 10 (5445 Only)

When a record is added to an indexed file, the file is checked to ensure that the record key being added is not a duplicate of a key already in the file. If the file is being processed randomly, the file index is scanned. (The file index is the portion of the index that existed before the current job was started; it is in sequence from a prior run.) If the new key to be added is not found in this file index, the area that contains keys added in the current run is searched on a key-by-key basis. The keys in this area are not necessarily in sequence, and must be searched by examining each key. If no similar key is found, the record is a legitimate "add" to the file. The number of keys in this "added index area" increases as records are added, and as a result, the time to search this area increases as the job progresses.

This "highest added key save area" is reserved at the beginning of the core index area by the system when 5445 indexed files are being processed randomly with additions (see Figure 46). The save area is equal to one key length. For single volume files, the save area will exist only if the number of bytes specified for core index (RPG II File Description) is equal to or greater than the key length.

If the highest key added to the file by the current job is saved, the search of the "added index area" can be avoided for added records that have keys higher than the previous highest added key. This saving of search time can be considerable if many records are being added in a job and if their keys are in ascending sequence (same sequence as the file).

For multivolume 5445 indexed files processed randomly, there is always a core index, and therefore the highest added key save area will always exist (for additions).

Pre-Sorted Input

When adding records to an indexed file using sequential processing (i.e., matching records in RPG II), the input *must* be sorted in the same type of sequence as the records in the file. When adding records randomly, it is not necessary that the input be pre-sorted. However, by pre-sorting the input for random processing, significant performance improvements are generally realized.

Key Sort/Merge

When adding records to an indexed file, the keys of the added records are held in an area separate from the file index. At the end of job (eg., after LR processing), the added keys are sorted and then merged into the file index. If the input is pre-sorted, the keys don't need to be sorted at end of job, and time can be saved. Also, if a work file is specified in OCL, the key merge time can be further reduced. (See *Work File For Key Sort/Merge*, following.) The amount of main storage also affects the time required for the key merge operation.

Work File For Key Sort/Merge

As we have seen earlier in this appendix, keys of added records are sometimes sorted — and are always merged — at end of job when adding to an indexed file. If disk space is available, you can enhance the performance of this function by specifying a work file for the key merge routine to use. Also, for Model 15, a work file can be specified for the key sort routine to use for an unordered load of an indexed file. The effect of making such a work file available to the key sort/merge is as follows:

	Key Sort/Merge Time (in minutes)		Reduction in Processing Time
	Without work file	With work file	
On 5444 (using $INDEX44):			
• Adding 500 records to 5000	2.7	0.5	81%
• Adding 2500 records to 10,000	22.6	3.9	83%
On 5445 (using $INDEX45):			
• Adding 500 records to 5000	1.9	0.4	78%
• Adding 2500 records to 25,000	36.3	3.1	91%

For this example, the keylength was 10 bytes; the work file for key sort/merge was on a different drive than were the file index and added key areas; and the added keys were placed near the beginning of the file (this distribution may somewhat slant the statistics, but in this example does not alter the point being made).

The work file is used to merge the added keys into the index, and must be large enough to contain all of the keys added to the file. If the program adds records to more than one indexed file, the size of the work file for key sort is computed by determining (for each file) the number of sectors required to contain the added keys. The work file must be able to accommodate the largest number of sectors you have computed.

Model 15 (5444 and 5445)

On the Model 15, there is a "highest primary key save area" as well as a "highest added key save area" (described in the preceding discussion). When a file is opened, the "highest primary key save area" contains the highest key in that file. Using this area, when records are added to the file the system can easily determine if the new record to be added is logically beyond the end of the original file.

Unlike the Model 10, both the "highest added key save area" and the "highest primary key save area" are always used to perform random additions to a file, regardless of the presence of a core index.

If the indexed file is on a 5444 disk, the work file must be named $INDEX44 and must be located on a 5444 disk. If the indexed file is on a 5445 disk, the work file must be named $INDEX45 and must be located on a 5445 disk. To compute the number of tracks required for the work file, use the following calculations:

For the 5444 disk:

$$\frac{256}{\text{keylength} + 3} = \text{Number of index entries per sector (drop the remainder)}$$

$$\frac{\text{Number of adds}}{\text{Number of index entries per sector}} = \text{Number of sectors (round up to next whole number)}$$

$$\frac{\text{Number of sectors}}{24} = \text{Number of tracks needed for work file (round up to next whole number)}$$

For the 5445 disk:

$$\frac{256}{\text{keylength} + 4} = \text{Number of index entries per sector (drop the remainder)}$$

$$\frac{\text{Number of adds}}{\text{Number of index entries per sector}} = \text{Number of sectors (round up to the next whole number)}$$

$$\frac{\text{Number of sectors}}{20} = \text{Number of tracks needed for work file (round up to next whole number)}$$

If the work file is not large enough to contain all of the added index keys, the keys are sorted without using the work file. (For the Model 15, a halt will occur, but you will be allowed to continue without using the work file.) If possible, the work file should be locatd on a different disk drive than the indexed file whose keys are being sorted. If this is not possible, the work file should be as close as possible to the beginning of the file whose keys are being sorted, in order to minimize the disk seek time required.

The work file can be used with multivolume files. However, a work file cannot be located on a pack that contains an offline volume from a multivolume file. The pack that contains the work file must remain online while the job is running.

For small indexed files of 10 tracks or less where sort time is negligible, using the work file will not improve performance and should be avoided.

To use a work file for key sort/merge, it is necessary only to specify the OCL FILE statement; no changes are needed to your source program, and your programs need not be recompiled.

Keylength

Keylength, which is usually determined by the application and is not too flexible, is a major factor in key sort performance as well as being a great determining factor in the size of the file index and the disk track index. For example, assume you have a file of 50,000 records. As shown in the following, the number of tracks required for the file index varies greatly as the keylength changes.

Keylength	File Index Tracks	
	5444	5445
5	66	90
6	75	100
7	84	109
8	91	120
9	100	132
10	110	139

Not only does an increase of one byte in the keylength greatly increase the size of the file index, but it could also result in an increase of 50,000 bytes in the size of the file (an increase of 9 tracks on the 5444 or 10 tracks on the 5445).

Distribution of Added Records

The difference in performance between two separate add runs may be explained by the distribution of added keys. With random additions, program performance can vary according to the distribution of added keys in relation to the existing file. If the added keys are distributed throughout the file, the time for the add run may be longer than if all additions are relatively close together. The reason for the difference in time required lies in the search for duplicate keys. With even distribution of keys throughout the file, more of the file index must be scanned than would be required with limited distribution.

For example, assume your file has keys numbered 00001 to 25000. If you were to add 1000 records with keys spread between 00002 and 24999, the time for this run could take longer than if the added keys were in the range 00002 to 05000, or from 20000 to 24999, or from 25001 to 26000. Other factors (discussed earlier in this appendix) which affect performance when adding records are pre-sorted input, highest added key save area, size of keys, size of index, etc.

INDEX File Description Entry (Model 15 RPG II)

To obtain additional core storage for the file index when processing 5444 or 5445 indexed files, specify this option on the File Description Specification (continuation statement). Normally only one sector of file index is read into core at a time; with this option, you can cause two or more sectors of file index to be read into core at one time.

INDEX

READER'S COMMENT FORM

IBM System/3
Disk Concepts and Planning Guide **GC21-7571-3**

YOUR COMMENTS, PLEASE . . .

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM branch office serving your locality.

Corrections or clarifications needed:

Page *Comment*

Please include your name and address in the space below if you wish a reply.

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

GC21-7571-3

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
Publications, Dept. 245
Rochester, Minnesota 55901

Fold

Fold

IBM®

International Business Machines Corporation
General Systems Division
5775D Glenridge Drive N.E.
Atlanta, Georgia 30301
(USA Only)

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

GC21-7571-3

IBM System/3 Disk Concepts and Planning Guide Printed in USA GC21-7571-3

IBM

International Business Machines Corporation
General Systems Division
5775D Glenridge Drive N.E.
Atlanta, Georgia 30301
(USA Only)

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