DataGeneral

Software Documentation

AOS INFOS® II System User's Manual

AOS INFOS®II System User's Manual

093-000152-02

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 093-000152

Printed in the United States of America
Revision 02, May 1984

NOTICE

AOS
INFOS®II
System User's
Manual
093-000152

Revision History:

Original Release - June 1978

First Revision - February 1981

Second Revision - May 1984

Effective with:

AOS Rev. 5.00

AOS INFOS®II Rev. 5.00

Much of the material in this manual has been reorganized and presented in a new format. For this reason, we have used vertical bars only to mark technical changes and additions to the previous revision of the manual. The following sections are largely or entirely new to the manual, and do not have change bars: Chapter 11, Appendix D, Appendix E, Appendix F, and Appendix G.

Preface

This manual describes the INFOS® II system as it operates under the Advanced Operating System (AOS). We assume that you have some experience with AOS and that you may or may not have experience with indexed file processing.

Organization of the Manual

We have organized this manual as follows:

Chapter 1	Introduces the INFOS II system, highlighting its features and functions.
Chapter 2	Describes INFOS II file parameters and the commands and procedures you'll use to create INFOS II files.
Chapter 3	Describes the commands you'll use to process INFOS II files. It tells you which arguments are required and which are optional for each command.
Chapter 4	Introduces some basic INFOS II file processing techniques and shows you how to combine commands to perform basic file processing operations.
Chapter 5	Defines and describes the stand-alone utility programs (except for INQUIRE) that are provided to help you create and maintain your INFOS II files.
Chapter 6	Describes the INQUIRE utility program.
Chapter 7	Explains your options for backing up your INFOS II files and recovering them in the event of a system failure.
Chapter 8	Describes how to use your INFOS II system with the FORTRAN 5, FORTRAN 77, DG/L™, and PL/I scientific languages. It also introduces the Business BASIC INFOS II interface.
Chapter 9	Describes the INFOS II assembly language interface.
Chapter 10	Describes INFOS II index and database structures so you can tailor your INFOS II files to suit your needs.
Chapter 11	Describes distributed INFOS II processing and explains how the Remote INFOS Agent (RIA) functions in a distributed INFOS network.
Appendix A	Lists the INFOS II error messages.
Appendix B	Contains INFOS II parameter files for the FORTRAN 5, FORTRAN 77, DG/L, and PL/I languages.
Appendix C	Contains sample FORTRAN 5, FORTRAN 77, DG/L, and PL/I programs that use the INFOS II interface.
Appendix D	Contains a sample COBOL program that uses the INFOS II interface.
Appendix E	Contains a standard ASCII character set.
Appendix F	Explains how to operate the INFOS II process.
Appendix G	Contains some helpful performance tips for INFOS II.

Related Manuals

You will find the following to be helpful companions to this manual:

- *Advanced Operating System (AOS) Programmer's Manual* (093-000120).
- *Programmer's Reference Manual, ECLIPSE®-Line Computers* (015-000024).
- *Software Documentation Guide* (093-000202).
- *AOS & AOS/VS CLI User's Manual* (093-000122).
- *AOS Binder User's Manual* (093-000190).
- *Sort/Merge with Report Writer User's Manual (AOS)* (093-000155).

In addition, you will want to refer to the appropriate reference manual for your programming language. We list these manuals below.

- *FORTRAN 5 Programmer's Guide (AOS)* (093-000154).
- *FORTRAN 5 Reference Manual* (093-000085).
- *FORTRAN 77 Reference Manual* (093-000162).
- *FORTRAN 77 Environment Manual (AOS)* (093-000273).
- *DG/L™ Language Reference Manual* (093-000229).
- *DG/L™ Runtime User's Manual (AOS)* (093-000159).
- *PL/I Reference Manual (AOS)* (093-000204).
- *PL/I-INFOS® II Interface Reference Manual (AOS & AOS/VS)* (093-000165).
- *Business BASIC Commands, Statements, and Functions (AOS/VS, AOS, RDOS, DOS)* (093-705005).
- *COBOL Reference Manual (AOS)* (093-000223).
- *RPG II Reference Manual* (093-000117).

If you plan to use Data General's PRESENT® facility or TPMS with your INFOS II files, refer to the appropriate manual:

- *PRESENT® Information Presentation Facility User's Manual* (093-000168).
- *Transaction Processing Management System (TPMS) Reference Manual* (093-000205).

Reader, Please Note:

We use these conventions for command formats in this manual:

COMMAND required *[optional]* ...

Where	Means
COMMAND	You must enter the command (or its accepted abbreviation) as shown.
required	You must enter some argument (such as a filename). Sometimes, we use: $\left\{\begin{matrix}\text{required}_1 \\ \text{required}_2\end{matrix}\right\}$ which means you must enter *one* of the arguments. Don't enter the braces; they only set off the choice.
[optional]	You have the option of entering this argument. Don't enter the brackets; they only set off what's optional.
...	You may repeat the preceding entry or entries. The explanation will tell you exactly what you may repeat.

Additionally, we use certain symbols in special ways:

Symbol	Means
⤶	Press the NEW LINE key on your terminal's keyboard.
□	Be sure to put a space here. (We use this only when we must; normally, you can see where to put spaces.)

All numbers are decimal unless we indicate otherwise; e.g., 35_8.

Finally, in examples we use

```
THIS TYPEFACE TO SHOW YOUR ENTRY⤶
THIS TYPEFACE FOR SYSTEM QUERIES AND RESPONSES.
```

) is the CLI prompt.

Contacting Data General

- If you have comments on this manual, please use the prepaid Remarks Form that appears after the Index. We want to know what you like and dislike about this manual.
- If you need additional manuals, please use the enclosed TIPS order form (USA only) or contact your Data General sales representative.

End of Preface

Contents

Chapter 1 - Introduction to AOS INFOS II

Chapter 2 - INFOS II File Creation Parameters

Chapter 3 - INFOS II Commands

Chapter 4 - INFOS II File Processing Techniques

Chapter 5 - The INFOS II Utility Programs

Chapter 6 - The INQUIRE Utility

Chapter 7 - INFOS II File Backup and Recovery Options

Chapter 8 - The INFOS II Scientific Language Interface

Chapter 9 - The INFOS II Assembly Language Interface

Chapter 10 - Index and Database File Structures

Chapter 11 - Distributed INFOS II Processing

Tables

Table

Illustrations

Figure

Chapter 1
Introduction to AOS INFOS II

Data General's AOS INFOS® II file management system lets you create, maintain, and use many types of databases in batch and multi-terminal on-line environments. The AOS INFOS II system's data-handling capabilities make it suitable for diverse applications in manufacturing, accounting, payroll, inventory, sales, credit, insurance, and distribution management.

The INFOS II system, operating in conjunction with AOS (as shown in Figure 1-1), not only retains all the AOS features, but also provides a new range of software capabilities. The INFOS II system makes it easy for you to design and process indexed files, using COBOL, Business BASIC, FORTRAN 5, FORTRAN 77, PL/I, RPG II, Idea, the DG/L™ language, or the system's Macroassembler (MASM).

With the INFOS II system software, you retain the AOS facilities to edit, debug, and link your programs, and to collect them into libraries. However, with the INFOS II utility programs, you can also create, delete, rename, dump, and load INFOS II files on an interactive console or in batch, using simple CLI commands. Utility programs allow you to use all available AOS INFOS II processing options, including those that may not be available through your application language.

In addition, you can use AOS INFOS II with a variety of other Data General software products. With the AOS Sort/Merge utility program, you can sort, merge, and copy AOS INFOS II files. The PRESENT® Information Presentation Facility can format the data in your INFOS II file into a report; and if you have the TRENDVIEW® Graphics Charting Package, you can make it into an easy-to-read chart. Data General's Transaction Processing Management System (TPMS) also supports INFOS II.

You can use the AOS INFOS II system in a multiuser environment. Many users can have the same file open at one time, or several users can have several INFOS II files open at one time.

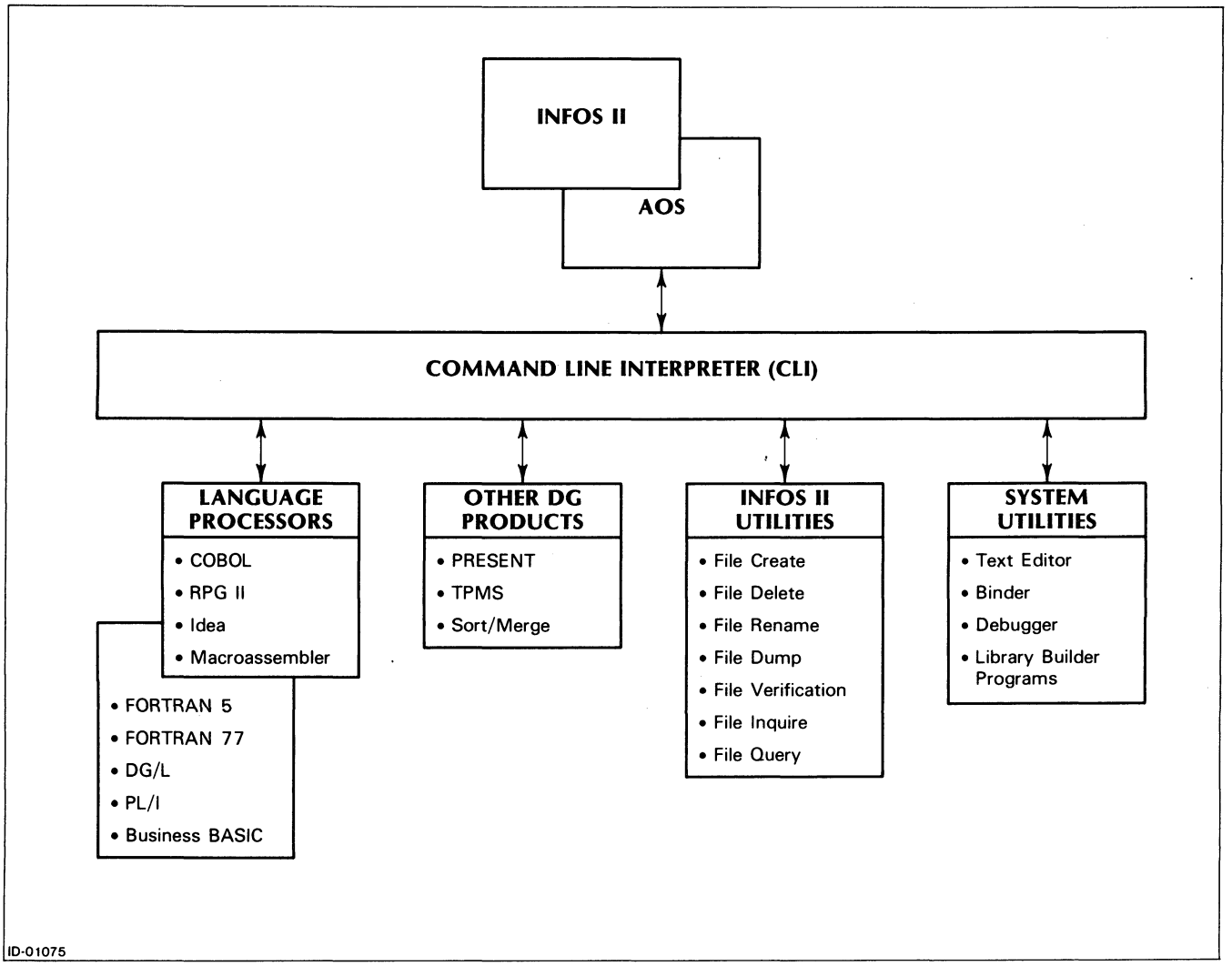


Figure 1-1. A Programmer's View of an AOS INFOS II System

File Organization

The file organization method you choose for any application depends upon the needs of that application. You can organize AOS disk files sequentially or randomly. (See the *AOS Programmer's Manual* for details.) The AOS INFOS II system combines the advantages of both sequential and random access methods in its own file organization method. Before we describe this method, we will summarize the characteristics of these two types of file organization.

Sequential Access Method Files

The system stores records in Sequential Access Method (SAM) files in a physically ordered sequence on magnetic tape or disk. Sequential files are useful if you want to process many data records in an ordered sequence at the same time. The system knows which record it will need to process next and gets the data ready while processing your current request. Records in SAM files can vary in length.

Figure 1-2 shows two SAM files. Each represents a customer file whose records contain customer names, account numbers, addresses, and balances.

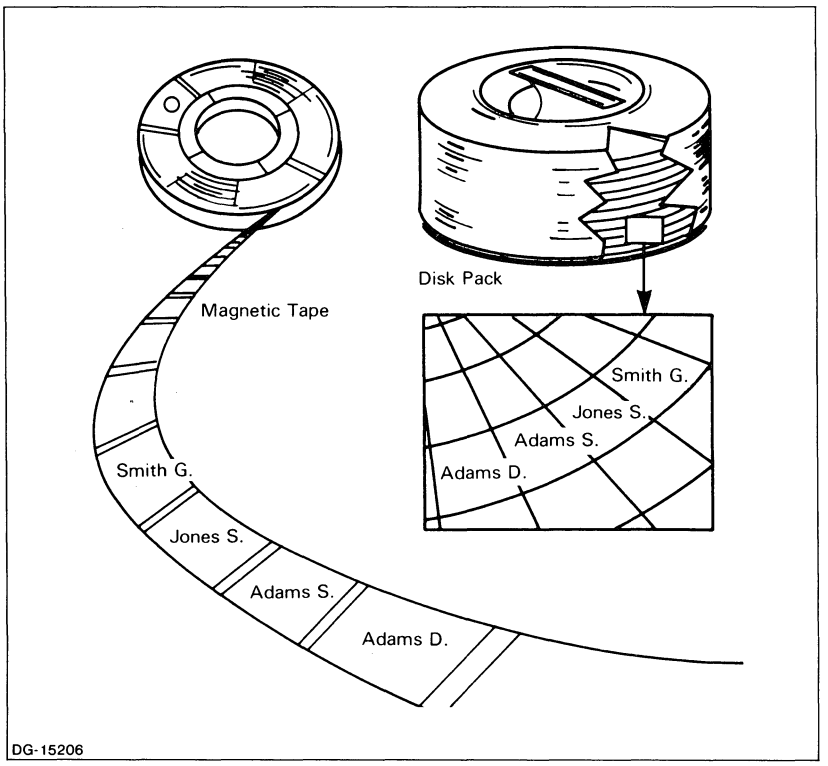


Figure 1-2. SAM Files

Sequential file organization plays an important role in a data-processing shop, because so much of the data that the computer takes in for processing and puts out as reports resides on sequential access devices.

But you can use SAM files only in a limited number of applications. The physical characteristics of magnetic tape, for example, dictate that you read records sequentially, starting at the first record in a file, and reading one after another until the end of the file.

For example, suppose you want to read G. Smith's record, which happens to be the 100th record in the file. You would have to read sequentially from the 1st record through the 99th to get to it. In addition, you can add records to a SAM file one after another, but you cannot insert a new record between two existing ones. You must create a new master file in order to insert a new record or to delete an old one from the middle of the file. This can be awkward and time consuming.

Thus, sequential organization probably would not be the best way to store your records if you wanted to process only a few records at a time and if you didn't have them arranged in sequence.

Random Access Method Files

AOS Random Access Method (RAM) files are more flexible than SAM files. The system stores the records in RAM files in any order on a disk and locates them with a unique system-assigned identification number. Because you don't have to gain access to each preceding record before finding the one you want, processing RAM files can be very fast. You can write and retrieve data records in any order. And in a RAM file, you can easily insert a new record between two existing records.

Figure 1-3 shows a sample RAM file. It is the same customer file you saw in Figure 1-2, except we used the random access method instead of the sequential access method.

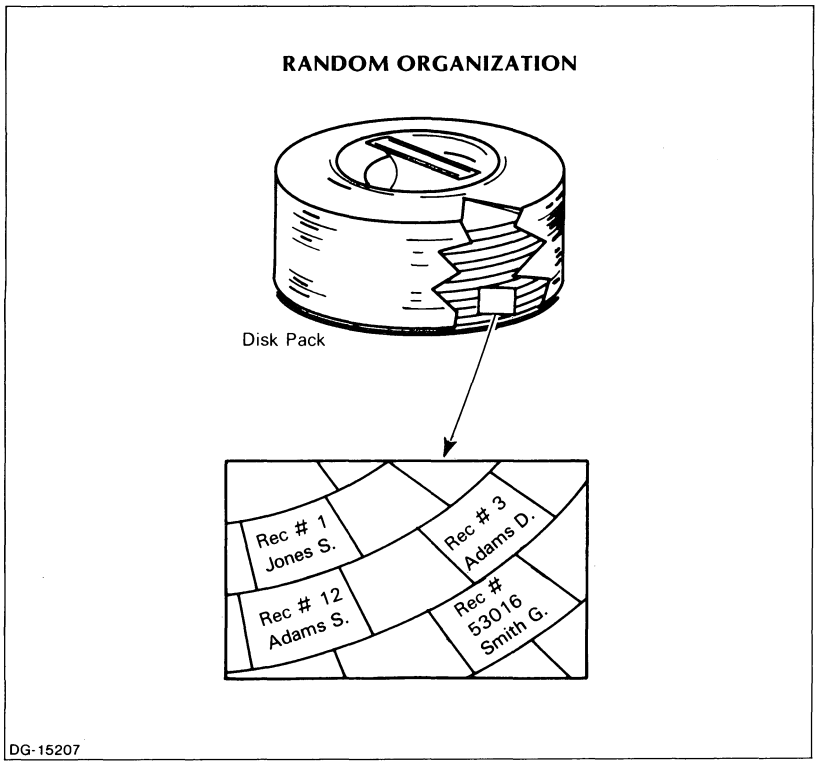


Figure 1-3. RAM File

For some data-processing applications, the ability to gain access to data records randomly is a significant advantage over the sequential access method. But some random file characteristics impose limitations on processing capabilities.

First, you can gain access to records only by byte, record, or block number. The record number has no logical connection to the record's contents. For example, you may find it difficult to remember that the system recognizes G. Smith as 53016. You have to establish your own association between the record and its number. If your products are known by their names, then you would probably have to maintain some sort of external list, equating a product's name to its record number.

Second, a problem arises as you discontinue making certain products and begin making new ones. If you decided to reuse the record numbers of the discontinued products for the new ones, you would be constantly changing your product name/record number list and probably have a lot of confusion. So instead, you might decide to add all the new product profile records to the end of the file, giving each new record the next sequential number. But in that case, your RAM file would eventually get very unwieldy and contain a lot of wasted space.

Third, to gain access to a record by its number, all the records have to be the same length, which isn't always the case. In a product profile file, for example, where each record is a description of one of your products, no two records are apt to be the same length. Therefore, to put these records in a random file, you must pad all the profile records to the size of the largest record. Here again, you end up wasting a lot of space.

Fortunately, as file designers discovered the above problems, they also found that they had the rudiments of a file index in the way the random access system equated a record's sequential number with its actual physical location. Such an equation is generally called a *sector address map,* and it opened the way to indexed sequential access method files.

Indexed Sequential Access Method Files

The AOS INFOS II system uses the Indexed Sequential Access Method (ISAM). ISAM files let you gain access to records sequentially or randomly, using a *key* you create to identify each record. A key is an 8-bit character string used to identify records in the database.

ISAM files bring the concept of *keyed access* to data processing. Just as the COBOL and PL/I languages remove you from the bit flipping mode of programming, keyed access frees you from the limitations of sequential file organization, and eliminates the need to know a record's location to gain access to it randomly.

You can probably already see certain fundamental advantages to an ISAM file. For example, if the product profile file mentioned earlier were an indexed sequential file, you could obtain a product description record by the name of the product the record describes. This is a more natural way to gain access to these records, rather than using numbers totally unrelated to the product. The INFOS II system's ISAM gives you these advantages and more.

Whether you are familiar with ISAM files or not, you'll want to read the remaining sections of Chapter 1. These sections highlight the basic concepts and features of the AOS INFOS II ISAM files.

The INFOS II Indexed Sequential Access Method

Although ISAM is not unique to the AOS INFOS II system, INFOS II ISAM provides you with system-unique options and features, which we will describe in the following sections of this chapter and in other chapters.

File Structure

An AOS INFOS II ISAM file is composed of two independently defined files: an *index file* and a *database file.* The AOS INFOS II system links these two files into a single processible unit at the time of creation. You define each structure according to the characteristics you want it to have. You should be concerned with the features that will allow you to perform the functions you want.

File size isn't likely to be a problem, since the INFOS II system imposes no practical limit on it. The largest ISAM file you can have depends on how many disk drives you have. You can have the index and database on a single disk, or you can have the index on one or more disks and the database on one or more other disks.

Multiuser Support

In addition to handling all the physical aspects of file creation and maintenance, the INFOS II system allows many different file users to gain access to an ISAM file simultaneously.

In modern data processing, it is common to have an on-line application in which many users simultaneously gain access to one file. Entering orders and controlling inventory are two examples of this. Another modern application involves real-time process control using a multitask program. A universal problem with situations such as these has been how to keep the file users separated so that one doesn't interfere with another.

The AOS INFOS II system doesn't have this problem. It lets as many as 256 separate file users simultaneously gain access to an INFOS II file while automatically keeping each user distinct from all others. Each person who gains access to an ISAM file can also lock the accessed data record, if desired, so that no other user can process that record until he or she unlocks it and returns it to general use. Moreover, any user can gain exclusive use of an index, if no one else is currently using the file.

Characteristics of an INFOS II Index

An AOS INFOS II index contains keys. As we already mentioned, keys are character strings that the system uses to identify records in the database. The system sorts these keys sequentially, according to their binary values. It automatically maintains the association between the index and the database, and ensures that at least one key points to every data record.

Think of a key as a data record's logical identifier. When you want to gain access to a record in an ISAM file, you just supply its key. You can gain access to a subset of records by using keyed access to obtain the first record randomly, and then supplying the keys for the subset in sequential order. You can gain access to any record randomly merely by supplying its key.

Since keys are usually made up of ASCII characters, a key can be letters that form a word (SMITH, JONES, ADAMS); digits and spaces that conform to some number, like a social security number (123 45 6789); or a combination of letters and numbers that might, for example, form a part or invoice number (1P202A). A key can be any combination of 8-bit characters from 1 to 255 characters long. (See Appendix E for a list of ASCII characters.)

An important characteristic of an index is that, regardless of the order in which you enter keys, the system stores them in sequence, based on their binary value. The INFOS II system determines a key's sequential location by looking at it as a string of 8-bit bytes. It compares two keys byte by byte, then stores the key with the lower value in front of the one with the higher value. Thus, if you are updating an ISAM file and enter records for the keys CAT, 123, BAKER, DOG, and ALBERT in that order, when you process the file sequentially, the record for 123 will be accessed first. The system will store the keys in the sequence 123, ALBERT, BAKER, CAT, and DOG — that is, sequentially by binary value.

Notice that, in the preceding example, the keys are not the same length, nor do they conform to any system-imposed format. Remember that a single key can be any combination of characters you want, from 1 to 255 characters long. Also, a key may or may not represent some field in a data record. For example, in a payroll file that includes the employees' names as fields in the data records, you could use names as the keys. But you could also use employees' social security numbers as keys, although they might not be a field in the payroll records.

Normally, one key in the index will correspond directly to one record in the database, but this is not always the case. You can construct a file with more keys than data records. This could be useful, for example, in a data-gathering application that will extend over several weeks. If you know when you create the file exactly what statistics you'll eventually collect, you can construct an index with a key for each expected record, even though some data won't be available until later in the life of the file. We explain how to do this in Chapter 3.

Characteristics of an INFOS II Database

An AOS INFOS II database consists of records containing fields of information, stored in the form of binary characters. The types of information you store will depend on your application. For example, you might want to store employee payroll records, accounts receivable records, or accounts payable records.

The AOS INFOS II system stores records randomly; the sequentially ordered keys point to the locations of their records. The lengths of INFOS II ISAM records can vary depending on how much information you want to store in them.

Figure 1-4 shows an ISAM version of the RAM file in Figure 1-3.

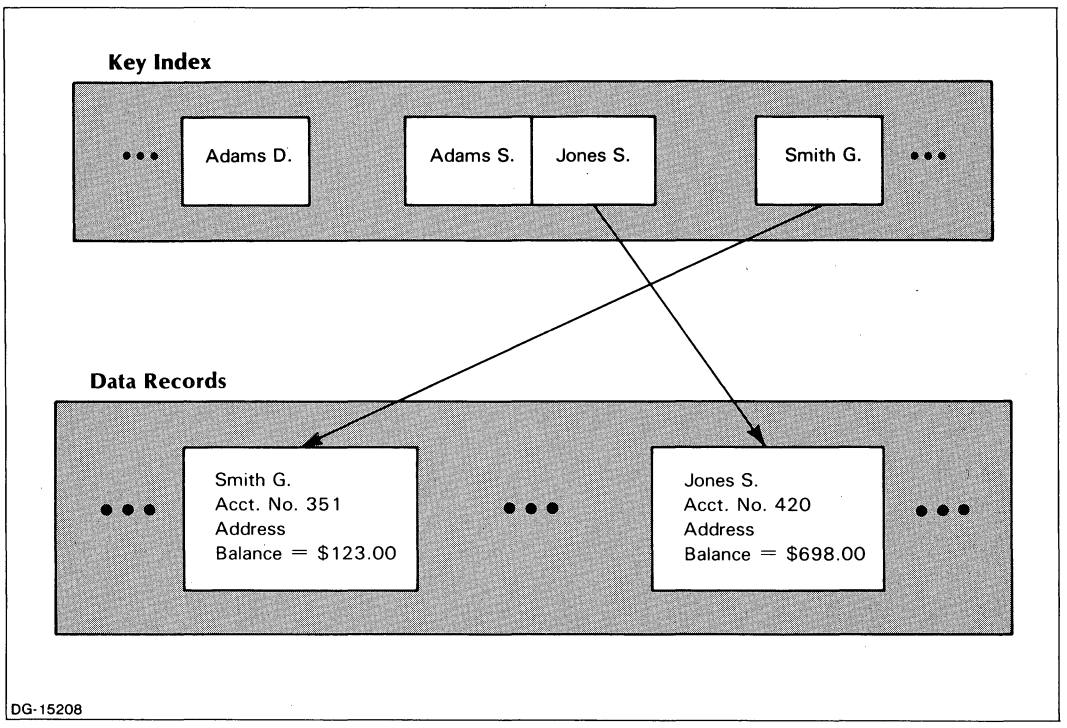


Figure 1-4. ISAM File

Defining Database Characteristics

You directly control the four primary characteristics of your INFOS II database:

- Size of the database.
- Length of a physical data transfer.
- Data record length.
- Space management.

As you learned earlier, the database resides on one or more disks. When you define a database, you can specify how many disks your data records, keys, and partial records will require. Also, you can specify the physical size required on each disk. Then, the INFOS II system will not attempt to place more records on that disk than will fit in the size you specified. Alternatively, you can let the system find as much room on a disk as it can. In any case, don't be concerned about the location of any data record because if you gain access to a record on one disk and the next record you want is on another disk, the INFOS II system will automatically find the second record no matter what disk it's on.

You can choose between two different physical *transfer lengths* for your database. Transfer length refers to the number of bytes that are physically moved between the computer and a storage device in one operation. You can elect to have either 2048 or 4096 bytes. We call this the *page size* of the database and explain it in greater detail in Chapter 10 of this manual.

Your data records can be whatever length is appropriate to store the necessary information, up to 4088 bytes. In addition, when you create an INFOS II file, you can specify that you want *space management* for the database. This means that the INFOS II system will automatically manage the space allocated for the database. It will recover space that is freed when you physically delete or relocate records, and make it available for reuse. Thus, you never have to reorganize your file to recover unused space. We discuss space management in more detail in Chapters 2 and 10.

Also note that each time you open an INFOS II file, you can request exclusive use of the database. If no other user has the file open when you request exclusive use, the system ensures that for as long as you have the file open you are the only INFOS user permitted access to the database.

Inversion

Inversion is an AOS INFOS II processing technique that allows you to make multiple references to a single data record. There are two types of inversion: database inversion and file inversion. With database inversion, you link multiple keys in the same index file to a single database record. With file inversion, you create a second index file for an existing database that already has an index. These techniques are useful if you have a file that contains several types of information and you want different sets of keys to correspond to the same information in the database.

For example, the database records in Figure 1-4 contain customer names, account numbers, addresses, and balances. This database has only one index, which contains customer name keys. Using file inversion, you can create an index with keys that contain account numbers, addresses, balances, or any other kind of information, and link them to the records in the same database. Figure 1-5 shows an inverted database. We discuss file inversion and database inversion in more detail in Chapter 4.

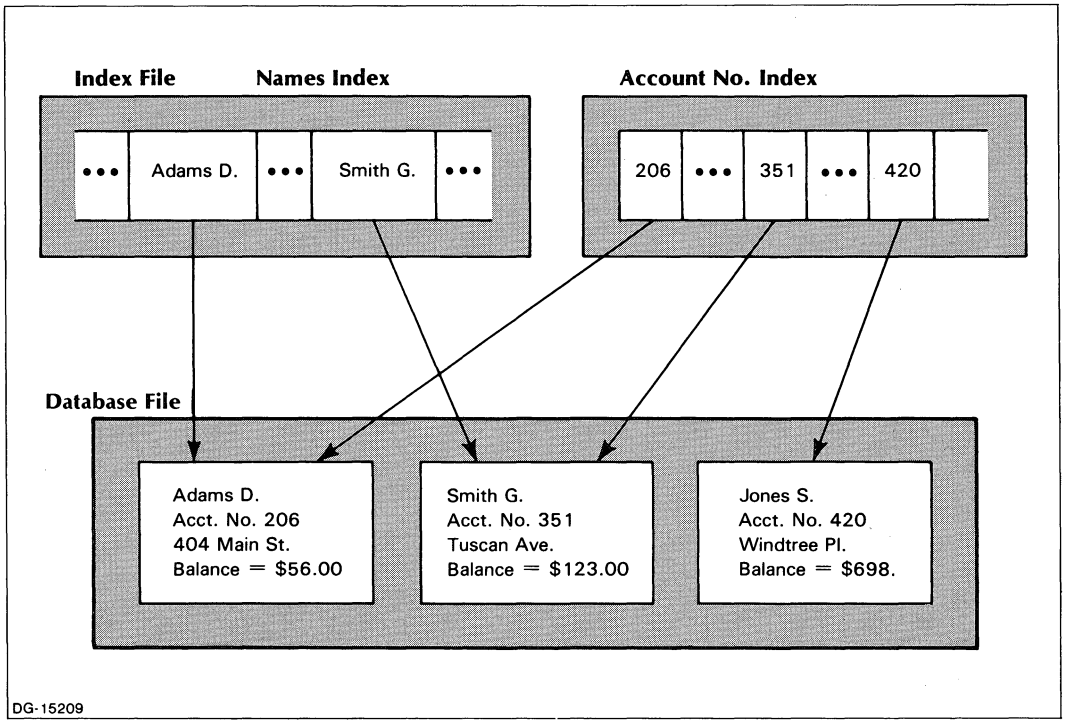


Figure 1-5. ISAM File with Database Inversion

Record Use Counts

The AOS INFOS II system keeps track of how many keys are linked to a single data record. This figure is called the record's *use count*. Each time you link a key to a record, the record's use count increases by one. When you delete a key, the use count of the record it points to is decreased by one. If a record's use count is reduced to 0, the system deletes the record. The system will never delete a record whose use count is greater than 0.

File Processing

We mentioned earlier that you can gain access to AOS INFOS II files either sequentially or randomly. If you want random or *keyed* access, you simply enter the key associated with the data record you want. The system then retrieves the data. For example, if you want the address of G. Smith, shown in Figure 1-5, enter the key Smith G. The INFOS II system will automatically find G. Smith's record and return the information in it.

If you want sequential access, you can use *relative-position* processing. This allows you to move sequentially through your index in the order of the keys. We call this movement relative because all movement within the index is in relation to the key you specified when you entered the index. You can start at the beginning of the index, or enter the index with any random key and then move sequentially through it from that point. (We describe how to do this in Chapter 3.) If you want to read the next record in the index, you read forward. If you want to read a record that you have already passed, you read backwards. Figure 1-6 illustrates relative-position processing.

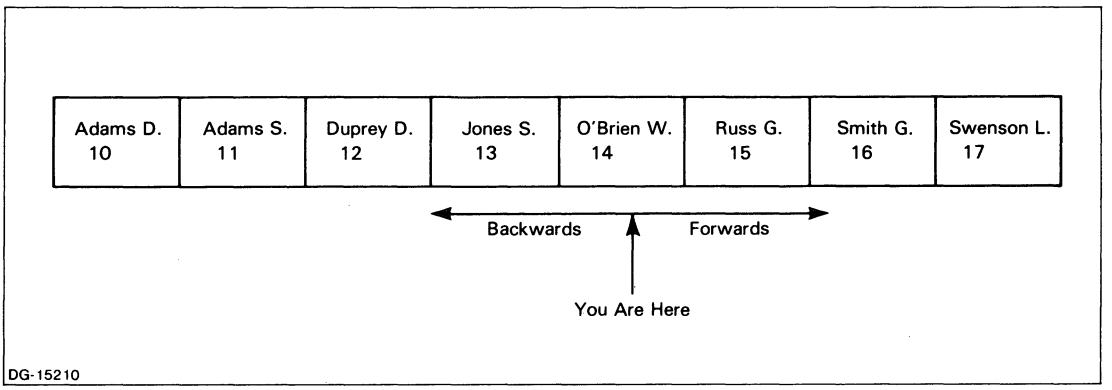


Figure 1-6. Relative Motion in an INFOS II Index

Generic and Approximate Search Keys

Two other features that make it easier to gain access to records in INFOS II files are *generic* and *approximate* search keys. You can use generic or approximate search keys to gain access to a record without specifying its entire key. You specify only the part you know. We discuss generic and approximate search keys in greater detail in Chapter 4.

ISAM Summary

With the AOS INFOS II Indexed Sequential Access Method, you retain most of the advantages of the Sequential Access Method and the Random Access Method without the disadvantages. You have sequential or random access to records, variable length keys and records, and the ability to insert and delete records without creating a new file. With inversion, you can associate more than one key with a single database record, or more than one index with a database. The AOS INFOS II system also provides you with parameters that allow you to tailor your files for specific applications. We describe these parameters in Chapter 2.

The INFOS II Database Access Method

After you've been using ISAM files for a while you'll realize that with a few, additional features, you can expand your processing capabilities. This is what the INFOS II Database Access Method (DBAM) is all about. Like the ISAM file, a DBAM file contains an index file and a database file. However, a DBAM index file can contain up to 32 levels of indexes. The index at the first level is often called the main index. The indexes in the levels below the main index are called subindexes. The keys in the main index and subindexes can be linked to the database records and other subindexes.

Depending on your needs, you can use subindexing alone or combined with other INFOS II accessing features, such as inversion. You can, of course, have a single-level DBAM file, which functions just like an ISAM file. We will look at how you can use different types of DBAM for different applications in the following sections.

Subindexing

You can have as many as 32 levels of indexes. The system numbers these from 0 at the top to 31 at the bottom. Only one main index can exist at level 0. The maximum number of subindexes at any lower level depends on the total number of keys at the next higher level. Therefore, if your main index has 80 keys, you could have 80 unique subindexes at level 1.

Sometimes a main index or subindex is referred to as a *selector* index or subindex. This name was derived from the way your application uses the subindex, not from any special properties or characteristics of the subindex. That is, you use the selector subindex to select one of the lower level subindexes.

Subindexing is useful in many applications. For instance, you might want a single database to contain two or more logically different types of data records. This database could contain accounts receivable, accounts payable, payroll, inventory, purchase order records, etc. For efficient access, you would store the keys for each record type in unique subindexes. Figure 1-7 shows a simplified DBAM file with this type of arrangement.

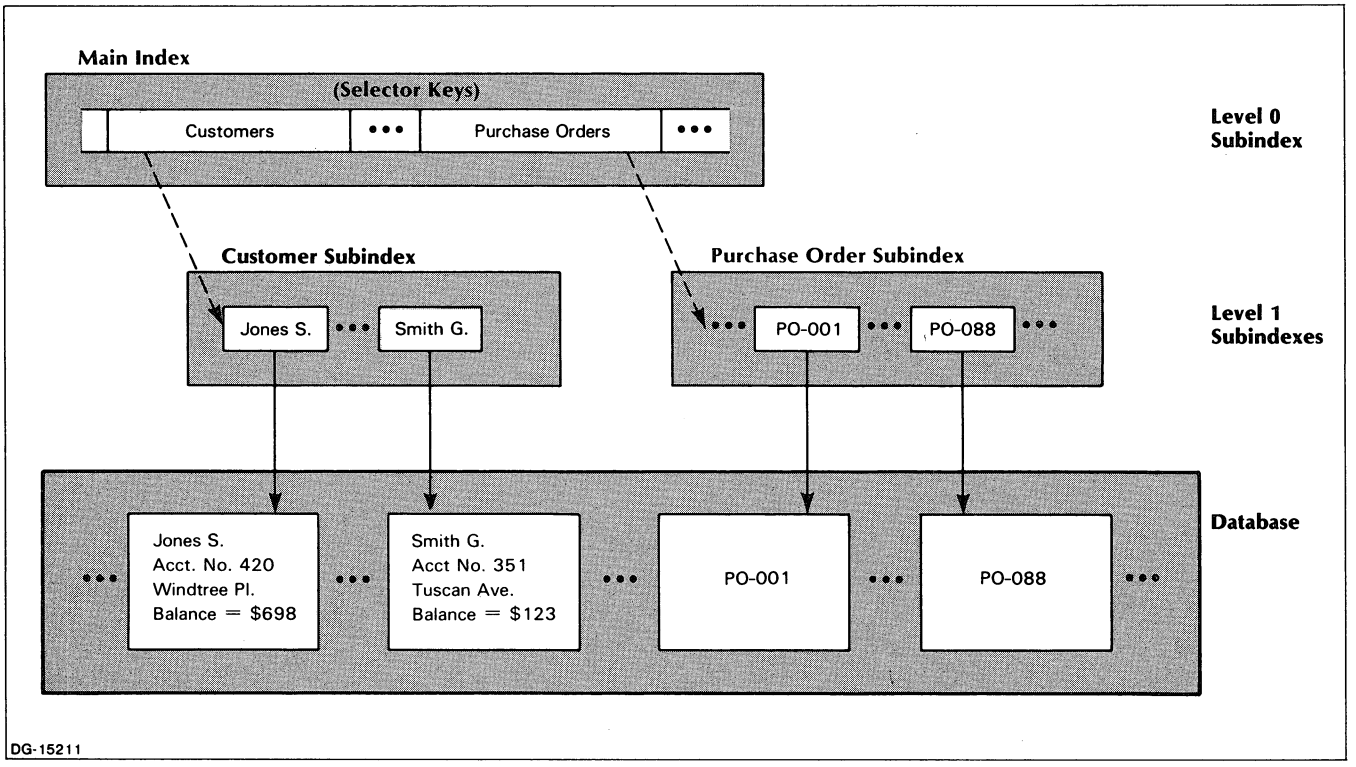


Figure 1-7. DBAM File with Subindexes

As you can see, the index file in Figure 1-7 contains three index levels. The one at the top is the main or level 0 selector subindex. The other two subindexes are level 1 subindexes: a customer subindex and a purchase order subindex.

Subindexes with Inversion

You can also use the inversion feature with subindexes in a DBAM file if your application requires it. When you use inversion, you don't need to keep multiple copies of a single record in order to retrieve the same information with two or more different keys. For example, an accountant might want to gain access to a record using a customer's name as the key, while a shop foreman might want to gain access to the same record using a part number as the key. Inversion makes the database records accessible to both of these people. Figure 1-8 illustrates this concept.

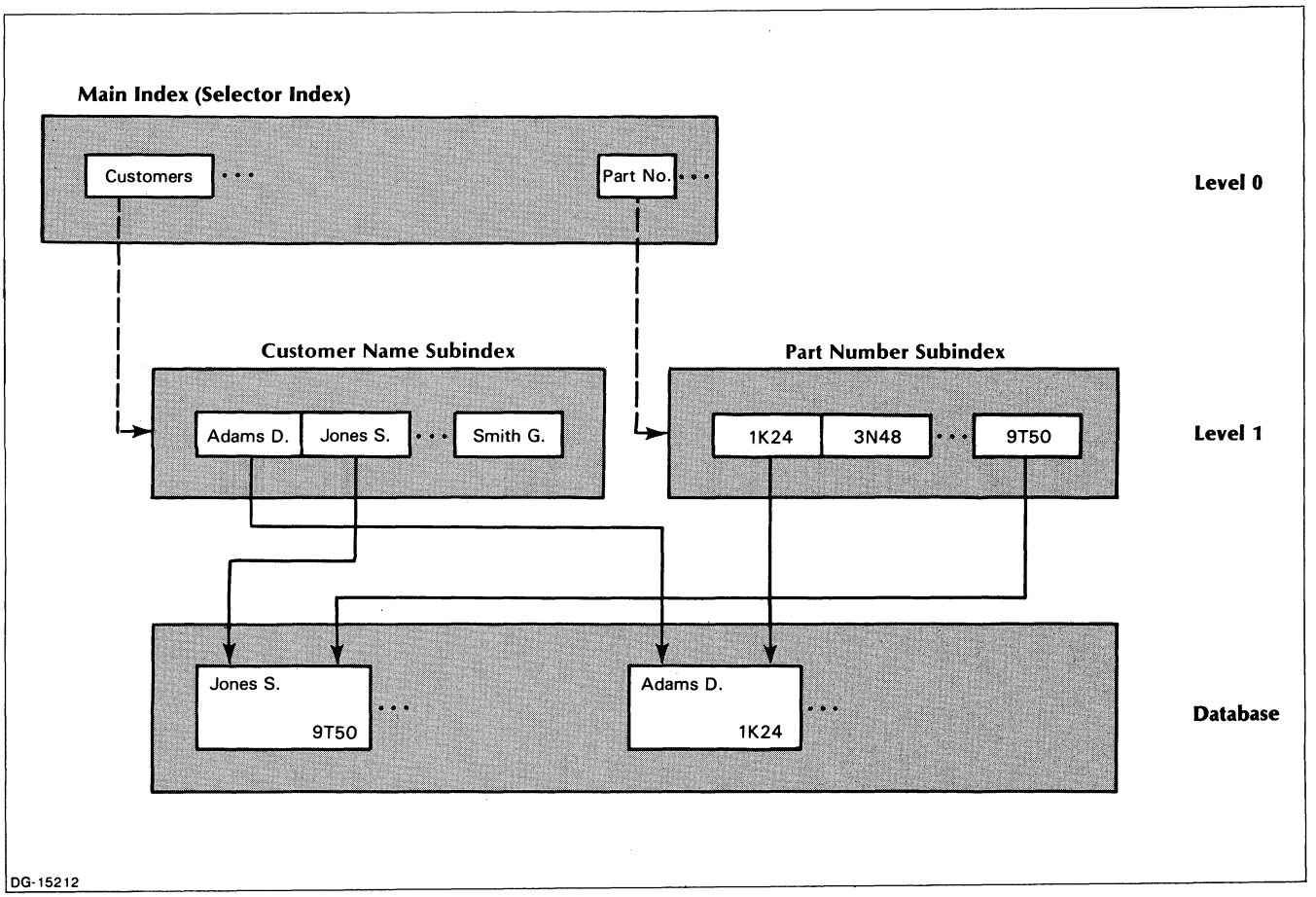


Figure 1-8. INFOS II File with Inversion in Subindex Level 1

With inversion, you also avoid the problem of making sure all identical records are updated each time you gain access to any one of them; there is only one copy of the record to modify.

Linking Subindexes

You can share subindexes among different keys in the same file through a command called LINK SUBINDEX. (See Chapter 3 for more information on this command.) Once you have created a subindex, you can link any key in any other subindex to it, which means you never have to duplicate a set of keys in a DBAM index.

For example, let's assume that for each customer, you want to have subindexes that contain invoice information. But you want to be able to gain access to them by either company name or company number. You will need a main selector index and two level 1 subindexes: one that contains company names and one with company numbers. The invoice number keys for each company reside in subindexes below the company name and number subindexes; in other words, in level 2. To avoid duplication of the invoice keys, we link both the company name subindex keys and the company number subindex keys to the subindexes that contain the invoice keys. Figure 1-9 illustrates this arrangement.

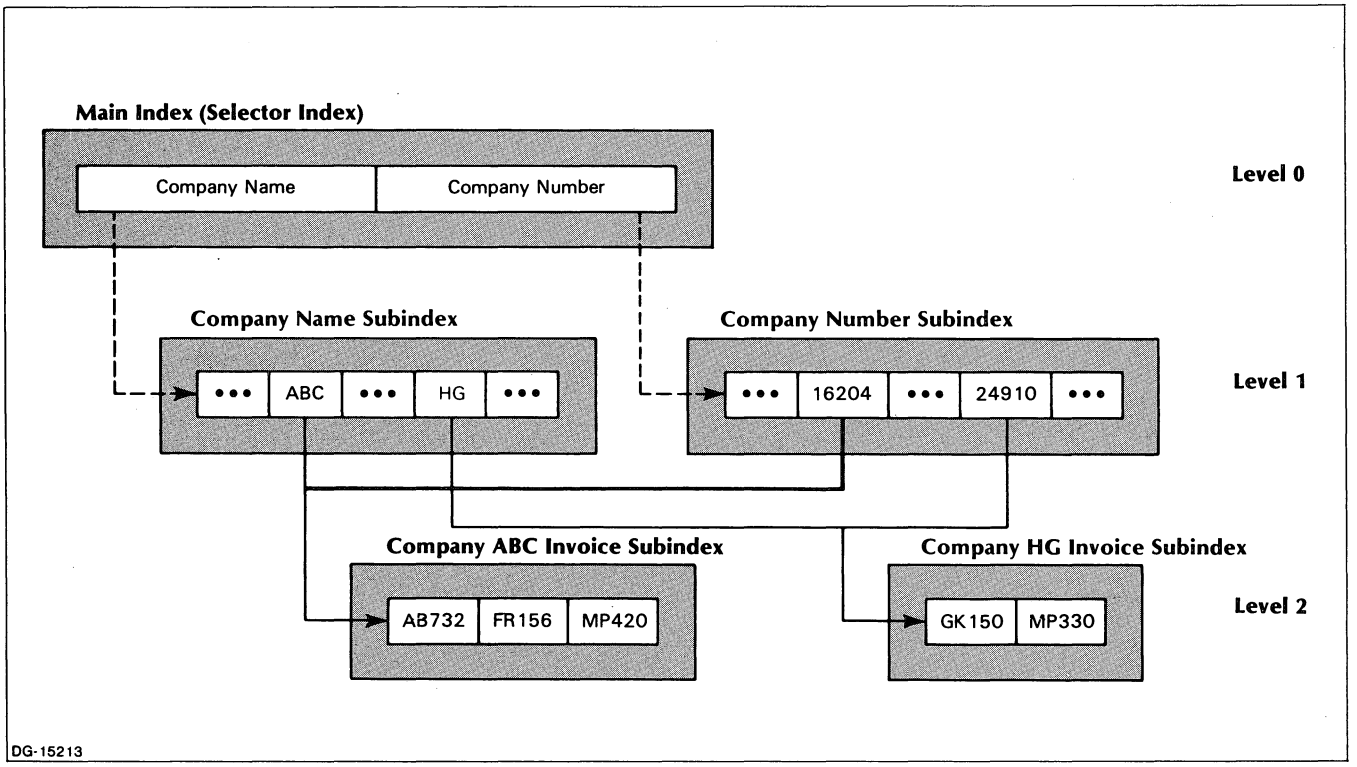


Figure 1-9. INFOS II Index File with Linked Subindexes

Assume that invoice number GK150 belongs to the HG company. You could gain access to the record for this invoice with two key combinations in this index:

```
COMPANY NAME,HG,GK150
```

Or:

```
COMPANY NUMBER,24910,GK150
```

Which combination you use would depend, of course, on whether you knew the company's name or just its number. If you knew both, you could use either.

Subindex Use Counts

The AOS INFOS II system maintains a record of how many keys are linked to a subindex through a subindex use count, much like the record use count we described earlier. Each time you link a key to a subindex, you increase the subindex's use count by one. When you unlink a key from a subindex (the DELETE SUBINDEX command), you reduce its count by one. The system will never physically delete a subindex with a use count greater than one. It will, however, let you physically delete a subindex that has a use count of one. (See Chapter 3 for a description of physical and logical deletion.)

File Processing

You gain access to DBAM files with the same techniques you use for ISAM files, but with some enhancements.

Relative Motion in a DBAM File

The relative accessing capabilities of ISAM are enhanced in a DBAM file. In addition to moving forward and backward within an index, you can move up and down between subindex levels. You can also combine keyed and relative access into one positioning request to get exactly where you want to go within your index. We discuss these access techniques in more detail in Chapter 3.

Keyed Access

As with ISAM files, you can gain access to records in a DBAM file by specifying random keys. You can also use generic and approximate search keys on both single-level and multilevel files.

DBAM Summary

With the AOS INFOS II Database Access Method, you have all the features of the Indexed Sequential Access Method, plus the ability to create several levels of subindexes. Thus you can build more complex INFOS II data structures. You can use DBAM subindexes with inversion, and you can link subindexes to avoid duplicating sets of keys. DBAM files are technically nothing more than multilevel ISAM files; and some, in fact, may only be single-level DBAM files. For this reason, we often refer to INFOS II files in the remainder of this manual as "single-level" or "multilevel," rather than ISAM or DBAM.

End of Chapter

Chapter 2
INFOS II File Creation Parameters

When you create an INFOS II file, you must provide information about the file's *parameters*. Parameters are data items that define the ways in which you can process the file. Some parameters are optional, and you must tell the system whether you want to use them. Others are required, and you must assign them a numeric value or name. In this chapter, we describe the INFOS II file creation parameters and the relationships among them.

Each parameter, except for the index filename, has a default or standard value that the system assigns it if you do not specify an alternative. You can override the system default with whatever alternative is best for your needs. Table 2-1 at the end of this chapter summarizes the default and alternative values for each parameter. You will probably find that the default values are adequate for most of your needs.

Using Parameters with Application Languages

You can create a file either with the INFOS II OPEN command from your application language or with the ICREATE utility. We describe the ICREATE utility in Chapter 5, and we explain how to create a file with the OPEN command later in this chapter.

How you code the OPEN command depends on your application language. Not all languages support all the possible OPEN/CREATE parameters. Check the programming manual for your application language to see which parameters are available to you. If you use the ICREATE utility, you have access to all INFOS II parameters.

With some programming languages, you can specify that you want a standard file, which has the default value for every parameter. If you specify a standard file, you need not make any further decisions about parameters; the system provides the defaults listed in Table 2-1.

INFOS II Filenames

When you create an INFOS II file, one parameter value that the system expects is a filename. Remember that an INFOS II file is composed of two files: an index file and a database file. Each of these files is an AOS control point directory (CPD). (See the *AOS Programmer's Manual* for a description of control point directories.) The index CPD and the database CPD hold their keys and data records in files called *volumes*. So when you name your INFOS II file, you actually select three types of filenames: an index filename, a database filename, and volume filenames.

There is no default index filename; you must supply a name. Based on that name, however, the system will default the filenames for the database and volume files unless you explicitly name them otherwise. The filenames you specify must comply with AOS structure and character set conventions. Each filename must be unique within its directory, or you will receive an error message and the system will not create the file.

Default Filenames

The system default for the database filename is the index filename with a .DB extension. The volume filename default is VOLnn, where nn is the volume number. For example, if you specify the name INVENTORY for your index file, the INFOS II system automatically names the database file INVENTORY.DB. It also creates two VOL01 files: one each under INVENTORY and INVENTORY.DB (see Figure 2-1).

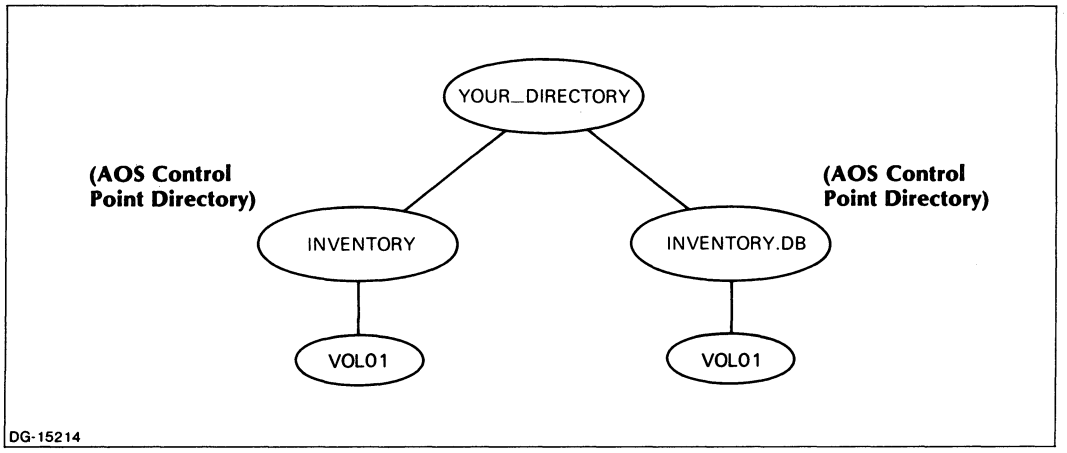


Figure 2-1. INFOS II File with Default Names in an AOS Directory

We recommend that you use these default filenames to simplify your application, unless there is a specific reason why an alternative name would be more helpful.

Alternative Filenames

For certain applications, you may want to have a database filename that differs from that of the index. If you have more than one index associated with a database, for example, you probably would not want the database to have a name that appears to associate it with one of the indexes and not the other.

Suppose you want to access the inventory records in Figure 2-1 through two different indexes: one with part number keys and one with vendor name keys. Let's assume that when you first created the file you named the part number index PART_NUMBER. Rather than using the system default for the database name (PART_NUMBER.DB), specify a database name of INVENTORY.DB. (Note that it is helpful if you include a .DB extension in your database filename.) When you create the second index, you can name it VENDORS. (See the section "Creating a New Index for an Existing Database" in Chapter 4.) Figure 2-2 shows an INFOS II file with these names.

You can gain access to the INVENTORY.DB database through either index: PART_NUMBER or VENDORS. You can also do multiple openings of this file by specifying one index name for each OPEN operation.

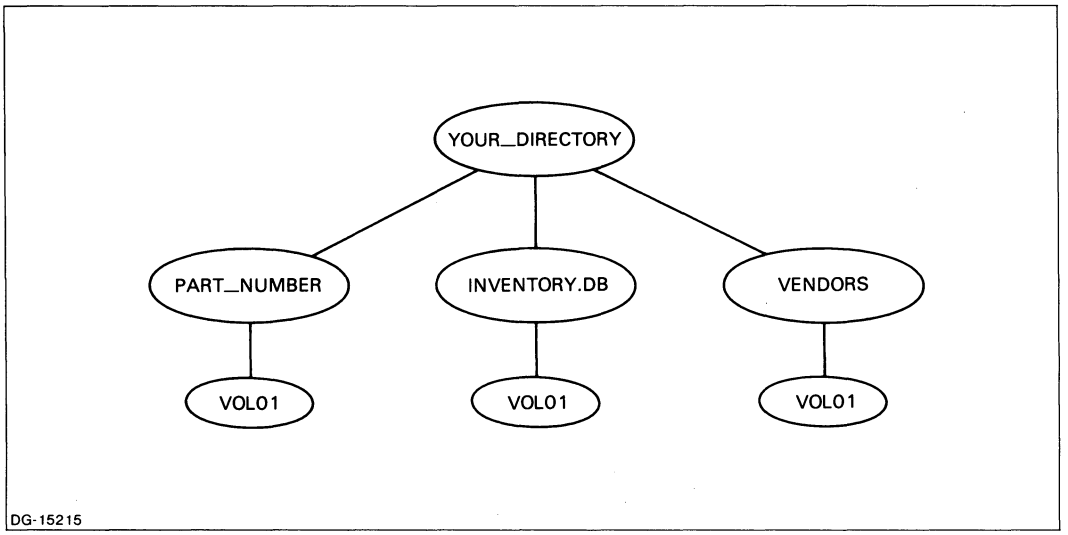


Figure 2-2. INFOS II File with Alternative Filenames

We suggest that you use the default volume names. If you choose to use alternative filenames for your volumes, do not name any of your standard volumes DVL01 through DVL16; these names are reserved for differential files. See Chapter 7 for more information on differential files. If you want to define various volumes under different directories, first read Chapters 9 and 10.

Filename Summary

In summary, any name you specify for an INFOS II file must be unique within the AOS directory in which it becomes an entry. The name you choose at creation time is always the name of the index; it is also the name you specify when opening the file for processing. Unless you indicate otherwise, the INFOS II system uses the index filename with the extension .DB for the database name. It automatically numbers volumes for the database and indexes sequentially from 01 to 32 unless you specify unique names.

Access Methods and Subindex Levels

You will choose an access method according to the number of index levels you want. ISAM gives you a single level index. With DBAM, however, you can select from 1 to 32 levels of indexing.

ISAM

With INFOS II ISAM, you can have only one index level for your keys. All the keys are stored in a main index, as shown in Figure 2-3. You will probably want to use ISAM if all of your records are of one logical type, such as accounts receivable, and if they have only one set of keys, such as customer account numbers, associated with them. With the ICREATE utility and with most application languages, the default access method is DBAM and you must specify ISAM if you want it.

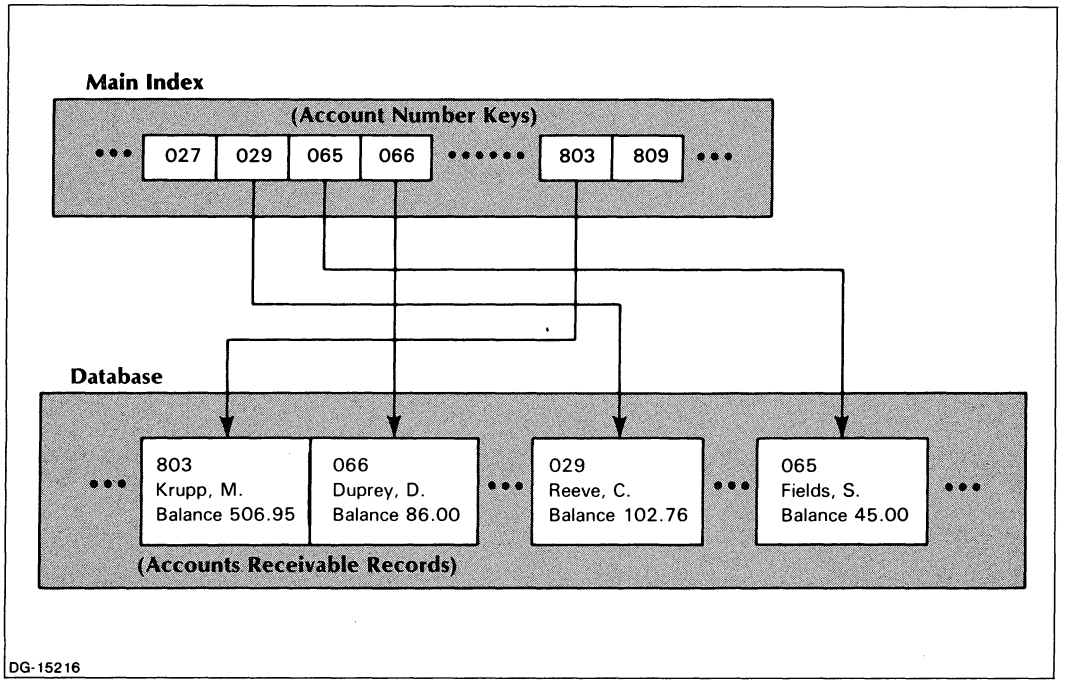


Figure 2-3. Typical INFOS II ISAM File

DBAM

If your database contains data records of two or more different logical types, you'll want to store the different sets of keys in different indexes. Hence you'll need a DBAM file, so you can use levels of subindexing. Of course, you can have a DBAM file with only one index level; the system will treat it as an ISAM file.

Levels of Subindexes

If you choose the DBAM access method, you must indicate how many levels of subindexing you want. You can specify from 1 to 32 levels, including the main index at level 0. The default is one level below the main index.

If you're not sure how many index levels you are going to need, you should specify a number that you don't expect to exceed. You might not use all of these levels, but if you don't initially choose enough levels, you can't get any more unless you create the file all over again. The system will not let you exceed the number of subindex levels you specify at file creation.

Remember that here we are talking about *levels* of subindexes, and not the total *number* of subindexes that you will create. As we said in Chapter 1, each subindex level can have as many unique subindexes as there are keys in the next higher level.

Examples of ISAM and DBAM Files

Let's assume, as we mentioned earlier in this section, that you have data records of all one logical type: accounts receivable. If you only have one set of keys associated with them, such as customer account numbers, then all you need is a single-level index. We picture this situation in Figure 2-3.

Suppose, however, that you have data records of two logical types; for example, part number records and purchase order records. You could store the part number keys in one subindex and the purchase order keys in another. In this situation, you would need a two-level DBAM file with a main, selector index and a unique subindex for each of the two key types. We have illustrated this arrangement in Figure 2-4.

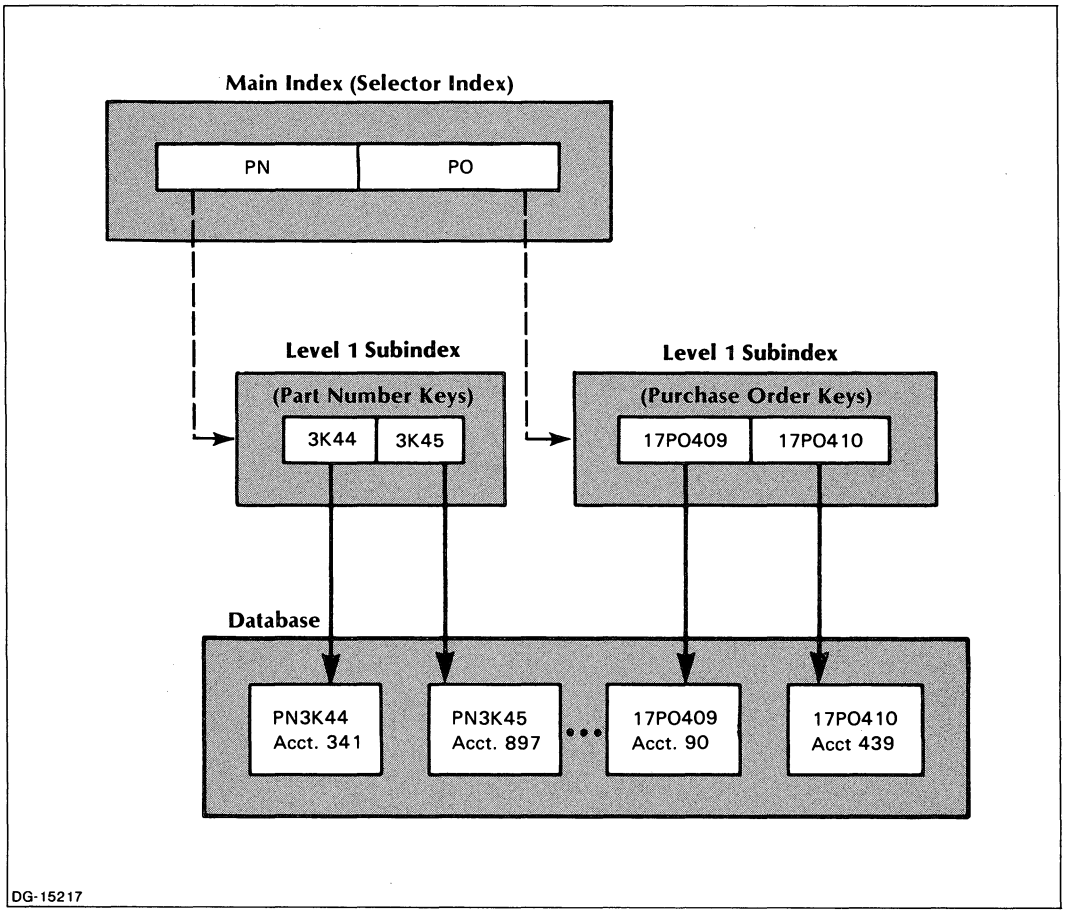


Figure 2-4. DBAM File with Two Types of Keys and Two Types of Data Records

In another arrangement, you might have two sets of keys associated with the same data records. For instance, customer account number keys and customer name keys could both be associated with a database of accounts receivable records. You could store the customer account number keys in one subindex and the customer name keys in another. Then you could gain access to any accounts receivable record with either an account number key or an account name key. Figure 2-5 diagrams this arrangement.

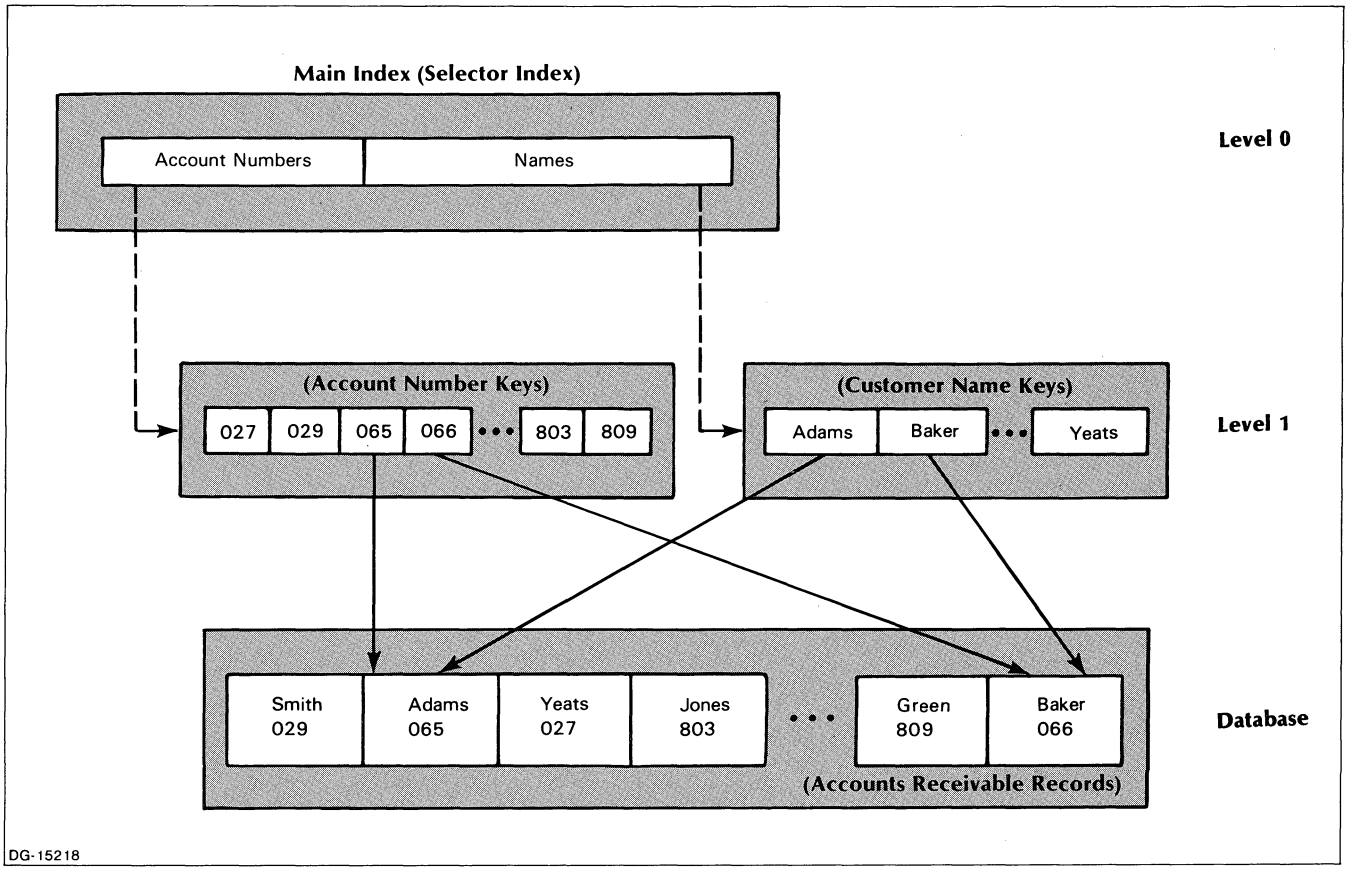


Figure 2-5. DBAM File with Two Sets of Keys Associated with One Type of Data Record

You will also use the Database Access Method (DBAM) when you have more than one type of data record and each record type has one or more set of keys associated with it. Figure 2-6 illustrates an INFOS II file with inventory records keyed by part number and vendor name, and purchase order records keyed by purchase order number and date.

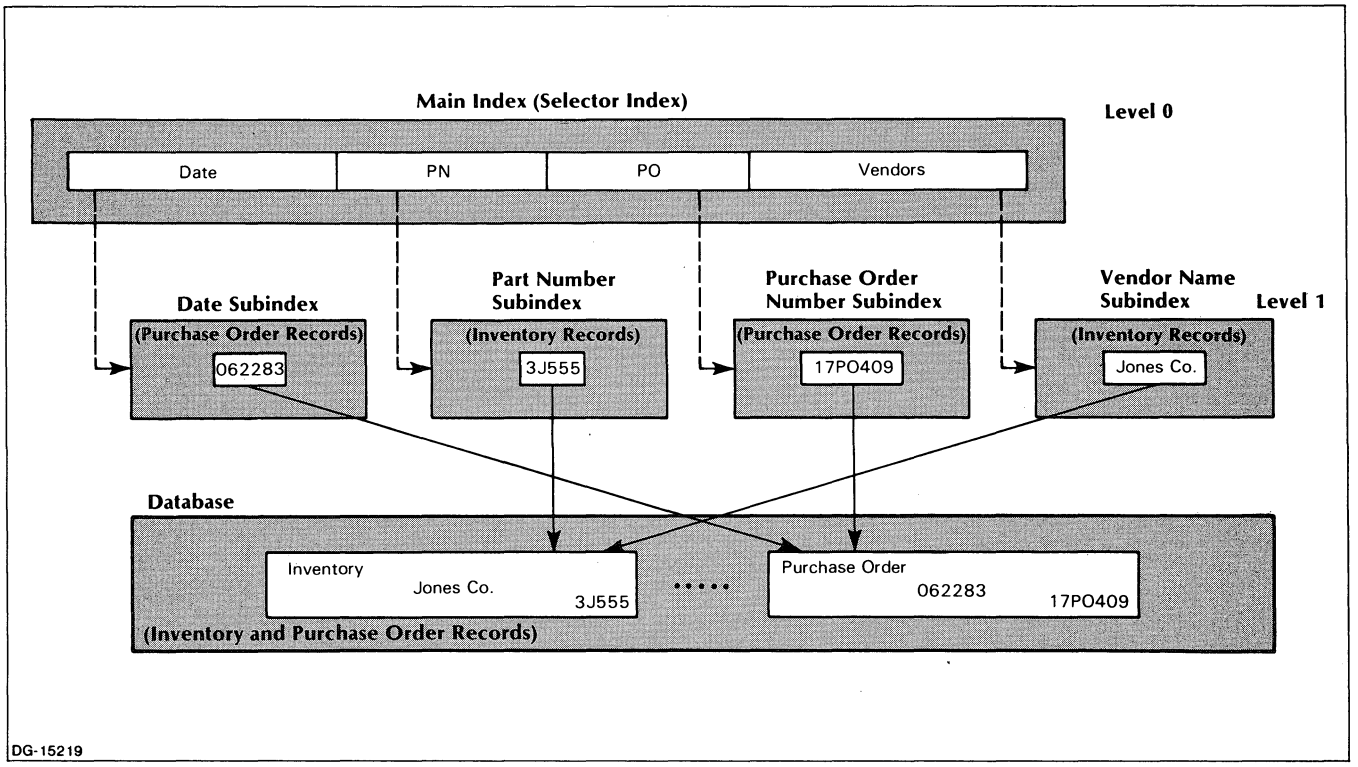


Figure 2-6. DBAM File with Multiple Sets of Keys Sharing Multiple Types of Data Records

Root Node Definition Parameters

The *root node* controls the information that defines every subindex or main index that you create. This information includes maximum key length, root node size, partial record length, and whether duplicate keys are allowed. When you create a file (ICREATE), the values you specify for these parameters apply only to the main index of the file. During file processing you'll use the DEFINE SUBINDEX request to specify a separate set of these parameter values for each new subindex you create. (We describe subindex definition in Chapters 3 and 4.)

Maximum Key Length

The default key length can vary, depending on your application language. The maximum key length is 255 bytes. This is also the default key length for the main index if you use the ICREATE utility. You can specify a different maximum key length for each subindex you create.

You can, however, limit the maximum key length in the main index or in any subindex. To do this, you specify the number of bytes you want as the upper limit. For example, if you assign a maximum key length of 25 bytes, the system checks the length of each key you attempt to enter. If the key length exceeds 25 bytes, the INFOS II system will refuse to accept it, and will return the following error to your program:

```
(IOKYL) ?KYLN ILLEGAL KEY BYTELENGTH -- USE 1 TO MAXIMUM ALLOWED IN SUBINDEX
```

Any key you enter can, of course, be shorter than the maximum key length. The INFOS II system lets you use *variable-length* keys. Consequently, no two keys in a main index or a subindex need to be the same length. Each key you specify can be exactly as long as you want, as long as it does not exceed the maximum key length. If, for example, you are using vendor name keys, one key might be ABLE CO. and another might be ABLE TOOL AND DIE COMPANY. Both of these keys would be acceptable with a maximum key length of 25 bytes, or of 255 bytes.

Root Node Size

A *node* is a unit of space that contains your index entries. The *root node* is the initial unit of space that the INFOS II system allocates to contain index entries in a main index or subindex. We deal with the subject of root nodes more thoroughly in Chapter 10.

Pages consist of nodes. Node size is normally 6 bytes less than your page size. Therefore, a 2048-byte page has a 2042-byte node, and a 4096-byte page has a 4090-byte node. The INFOS II system uses the 6 bytes that aren't included in your node.

Partial Record Length

In certain applications, you might want rapid access to small amounts of data, without having to gain access to a data record. You can store this type of information in the keys of any index in storage spaces that we call *partial records.*

Partial records are useful if there is a small bit of information associated with each key that you need often. This information might also be contained in the database records, but if you use partial records you don't have to search all through a record to retrieve that one piece of information. And more importantly, partial records save you processing time because the system must gain access to the disk only once to retrieve the key from the index file. It does not have to gain access to the database.

For example, suppose you have a personnel file like the one in Figure 2-7. The database records contain, among other things, employees' internal mailing addresses, which you frequently need for interoffice mail. You'd waste time if you had to go through an employee's entire record each time you needed just that one piece of information. So instead, you can include the mailing address as a partial record with each employee's key. Then you can retrieve the key and its partial record at the same time, without needing to retrieve the data record.

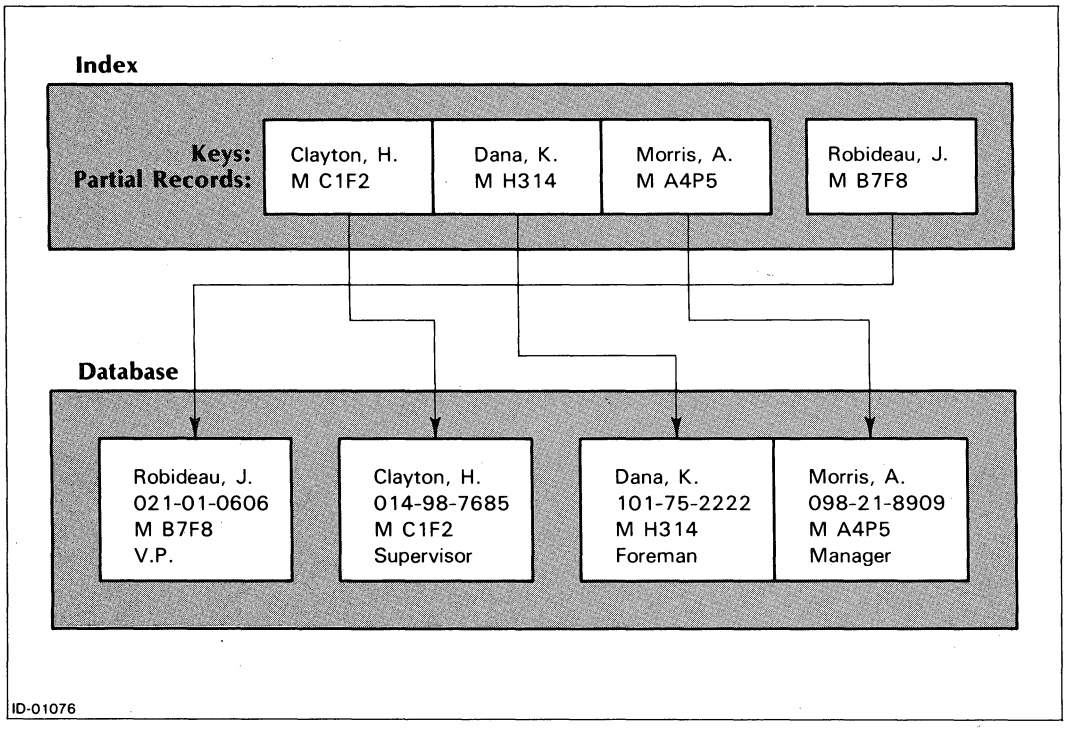


Figure 2-7. INFOS II File with Partial Records

The default partial record length is 0 bytes, but you can allocate up to 255 bytes for partial records in the main index or any subindex. If you take the default value, then you cannot store any information in partial records for that index. Whatever length you choose is a *fixed-length* specification; the system automatically allocates the specified number of bytes for each key that you write to the index.

We describe how to read and write partial records in Chapter 3. You don't have to write a partial record for each key, but the space is allocated whether you use it or not.

Duplicate Keys

With the allow duplicate key parameter, you specify whether you want to allow multiple keys to have the same name within the main index or a subindex. If you allow duplicate keys, then you can write a key to that subindex or main index that has the same name as an existing key, and then write a separate record for the duplicate key. The INFOS II system maintains the individuality of each key.

The system automatically assigns an *occurrence number* to each key you write to an index. The occurrence number acts as a unique extension to the key, so that the INFOS II system can identify it. With duplicate keys, you can gain access to the particular record you want by entering the key's name and its occurrence number. See Chapter 4 for information on how the system returns occurrence numbers to you when you use duplicate keys.

The default for this parameter is no duplicate keys, which means you must provide a unique name for each key in an index. If you intend to use duplicate keys in the main index or a subindex, you must explicitly specify this when you create the index.

The duplicate key parameter is useful when you want to process subsets of data records, using the same primary key for each record in the subset. For example, you might want to use duplicate keys in a file which contains data records of customer names and addresses. You could group all of your accounts in the Boston area under the key BOSTON. If you have specified duplicate keys, the first Boston account has the lowest occurrence number. The second Boston account has the key with the next higher occurrence number, and so on. Figure 2-8 shows such a file, where the key with the lowest Boston occurrence number is BOSTON 1 and the next is BOSTON 5.

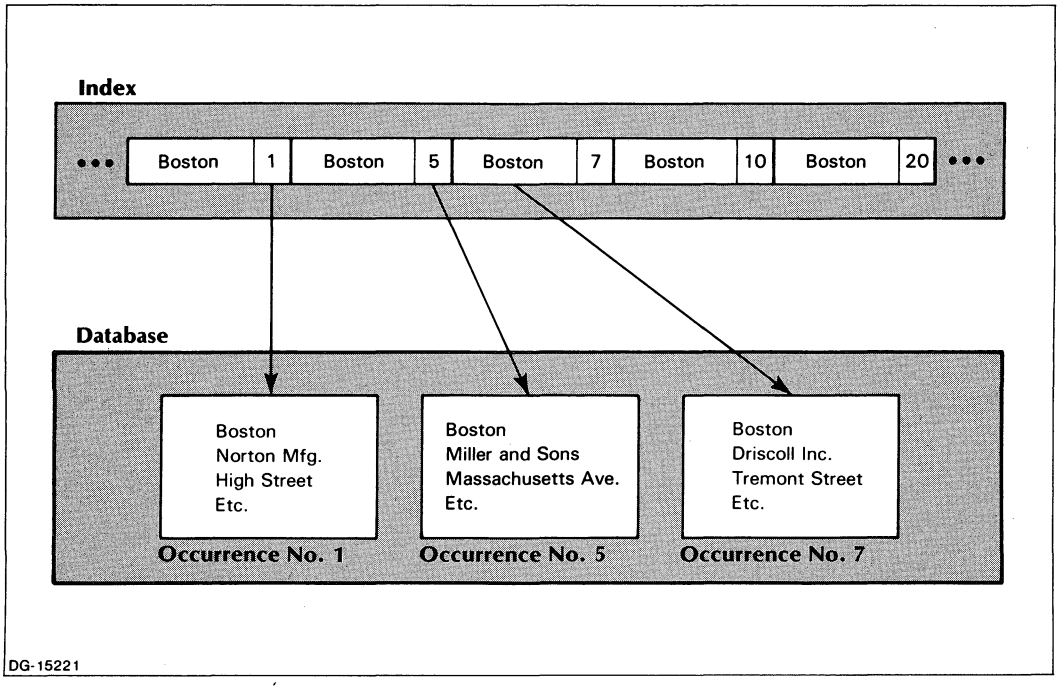


Figure 2-8. INFOS II File with Duplicate Keys

If you wanted to generate a mailing list for your customers in Boston, you would access the BOSTON key and read sequentially to the end of the BOSTON keys. The duplicate key parameter is also useful if you have keys that consist, for example, of last names. You might have several people in an index with the same last name.

Page Size, Element Size, and Volume Size

We describe these parameters together because what you choose for one usually affects what you can specify for the others. In most cases, the default values for these parameters are adequate for the index and database of any INFOS II file. You should be familiar with the contents of Chapter 10 before you choose alternative values for an index.

Page Size

Data is physically transferred between the disk file and the INFOS II system in units called *pages*. Table 2-1 shows that the default page size is 2048 bytes, and the alternative size is 4096 bytes. The length of your longest data record determines the page size you specify for a database. A 2040-byte record fits on a 2048-byte page and a 4088-byte record fits on a 4096-byte page. In both cases, the system uses the other 8 bytes. Your page size also limits the maximum size your volumes can become. (See "Volume Size," below.)

Chapter 10 contains more information on database pages.

Element Size

An *element* is a group of 512-byte blocks that the operating system allocates when you need disk space. The element is located wherever the system finds the appropriate number of contiguous disk blocks to equal your element size.

The default INFOS II element size is 32 blocks. For most applications, the default size is adequate because it uses disk space efficiently and reduces access time.

Your alternative options for element size depend on the page size you specified for your file. For 2048-byte pages you can choose any element size that is a multiple of four, and for 4096-byte pages, you can choose any element size that is a multiple of eight. If you don't select a multiple of four or eight, the INFOS II system will round down your specification to the next lowest multiple.

Regardless of page size, your element size cannot exceed the size of a Logical Disk (LD), established through the AOS Disk Formatter Utility. The *AOS Programmer's Manual* and *Learning to Use Your Advanced Operating System* contain more information on LDs.

Volume Size

A *volume* is the space allocated on one or more disks for the storage of an INFOS II file. The maximum size a volume can become depends on your page size. Maximum volume size for the smaller page is 1,048,576 512-byte disk blocks; for the larger page size, it is 2,097,152 blocks. The maximum sizes are the default sizes. The smaller size volume gives you more than 536 million bytes of disk storage. The larger standard size volume gives you twice as many bytes of disk storage.

Because two or more INFOS II files can share the same disk and because the system allocates space as needed, these figures indicate the maximum size that volumes have the potential to become, per file.

To make your element size and volume size equal, specify the same number of blocks for each in the OPEN command when you create the file. With your first WRITE command, the INFOS II system allocates a number of contiguous blocks equal to the element size. This ensures that you'll get all the space you asked for, if it's available. But it also means that there is no potential for growth on that volume. If you want more space, you can specify another volume. The INFOS II system automatically switches to the next volume when the current volume is full.

Number of Volumes

The default number of volumes is one for the index and one for the database. You can specify up to 32 volumes for your index and database, unless you use differential files. With differential files, you can only use up to 16 volumes. See Chapter 7 for more information on differential files.

Saving Disk Space

Space management, key compression, and data record compression are all ways of saving disk space. Depending on your application, you might want to use one or all of these features. Each one requires some disk space itself, but will almost always save more space than it uses.

Space Management

You can specify space management for the index, the database, or both. The default for this feature is no space management. Space management recovers and reuses disk space freed for any of the following reasons:

- The physical deletion of keys or records.
- The relocation of records due to rewriting.
- The deletion of a subindex.

If your application requires that you frequently insert and delete keys and records, you might want to use the space management feature. See Chapter 10 for more information on space management.

Key Compression

Key compression can save significant room in your index if you have a large number of keys with identical leading characters. You select key compression for an entire index structure; it cannot be chosen on a per-subindex basis. When you use this feature, the system stores only the unique part of any key that has the same leading characters as another key. From your point of view, however, each key remains separate and complete, just as you entered it.

For example, if you are using customer names as keys, and you have the keys "Jones A." and "Jones B.", the INFOS II system stores the common part of these keys just once in the index. It stores the unique part of every key. Figure 2-9 represents how the INFOS II system internally stores compressed keys.

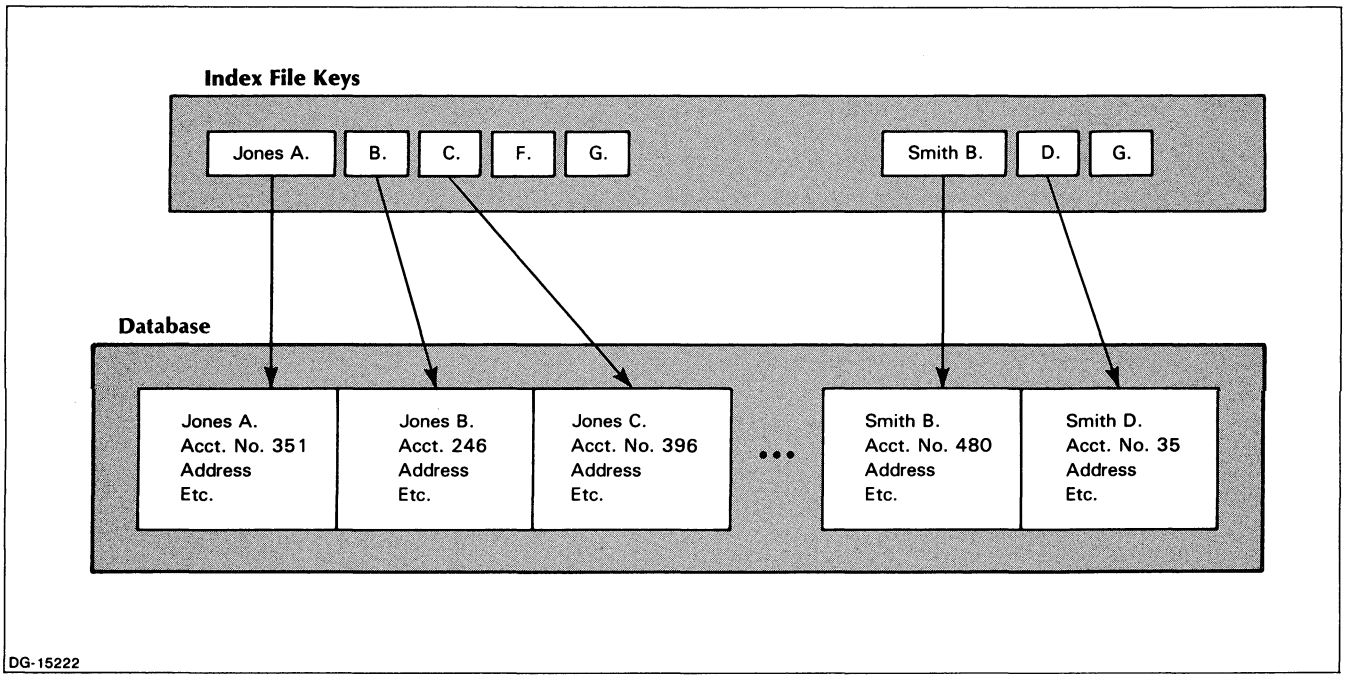


Figure 2-9. Internal Representation of an INFOS II File with Key Compression

If the leading characters are not the same, the INFOS II system does not compress subsequent identical characters. For example, the system does not compress the keys "Warren, Barbara" and "Jackson, Barbara".

Once you select key compression, you need never consider its existence. The INFOS II system performs all the compression without your help. You gain access to information in the same way you would without key compression: by entering the entire key. The system will not recognize a key if you specify only its unique portion.

Key compression saves space in large files that contain many redundant keys. You should note, however, that key compression can slow your processing speed slightly, because the INFOS II system must expand each key out to its entire length in order to process it.

The default for this feature is no key compression.

Data Record Compression

In many applications, it is convenient to specify one record length for an entire INFOS II file. However, this length has to be general enough to suit all possible record entries. As a result, you may have to allocate a large, fixed number of bytes to every record in your database, and pad the unused sections of your records with blank characters. This method can waste enormous amounts of disk space.

To help solve this problem, the INFOS II system provides an option called data record compression. When you use this feature, the INFOS II system automatically compresses any character that appears at least three consecutive times in a data record and makes note of the number of repetitions. Thus, a series of blanks or zeros, for example, will be compressed and will not take up as much space.

Figure 2-10 illustrates how the INFOS II system stores the information in a compressed record. Each block in the figure symbolizes 1 byte. The first byte indicates the total number of *descriptors* in the record. Each descriptor is a 2-byte segment that contains the offset of the character and the number of times the character is repeated consecutively in the record. There are descriptors only for characters that appear at least three times in a row in a record; not for those that will not be compressed.

The descriptors are all listed after the first byte in the compressed record. Following the descriptors are the record's actual characters. Those that don't appear at least three times in a row in the original record are contained in their original form in the compressed record. Therefore, you might have some characters appear twice in a row in the compressed record.

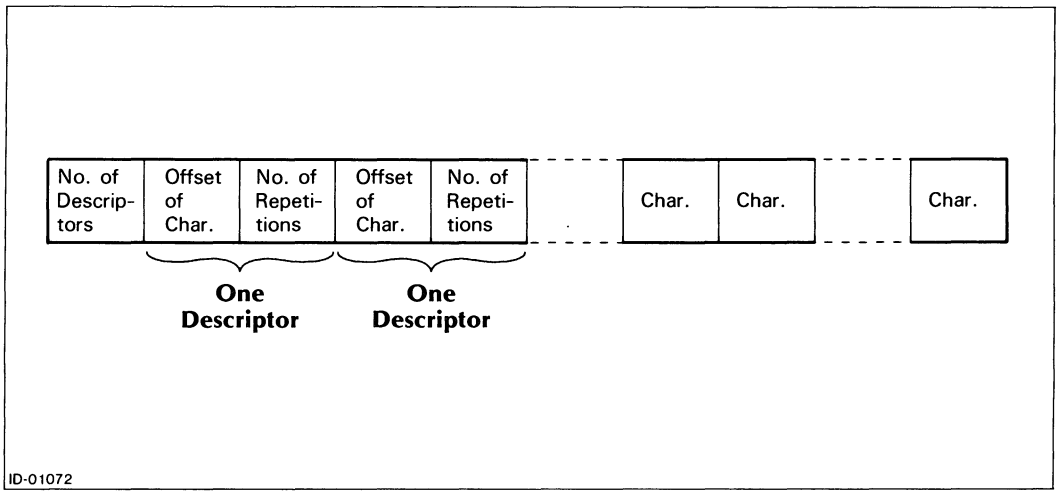


Figure 2-10. Representation of a Compressed INFOS II Data Record

If you use data record compression, you will notice a decrease in the size of your database file as compared to the same file without compression. The system will never compress a record if its compressed length is greater than its uncompressed length. This could be the case if your record contains characters of only three repetitions, because the descriptor and the character itself take up 3 bytes, and you have the additional first byte in the record that indicates how many descriptors there are.

Figure 2-11 shows two representations of data records in original form and in compressed form.

Limitations of Data Record Compression

Although data record compression is often useful, it is not always to your advantage to use it. Data compression information is stored in bytes, which places the following restrictions on the compression feature.

- Because the position indicating the number of descriptors is only 1 byte long, you can't have more than 255 descriptors per data record.
- Because the position that indicates the offset of the compressed character is only 1 byte long, no more compression is possible after the length of the compressed record reaches 256. Thus, if you have a lot of identical characters, blanks for example, towards the end of a lengthy record, they might not get compressed. We illustrate this situation in Figure 2-12.
- Because the position of the descriptor that indicates the number of repetitions of the character is also only 1 byte long, each descriptor can only contain up to 255 repetitions. If the number of repetitions is greater than 255, additional compression descriptors must be used.
- A file using data record compression might take longer to process than one not using it.
- Rewriting compressed records could force indirect rewrites, which slows down processing even further.

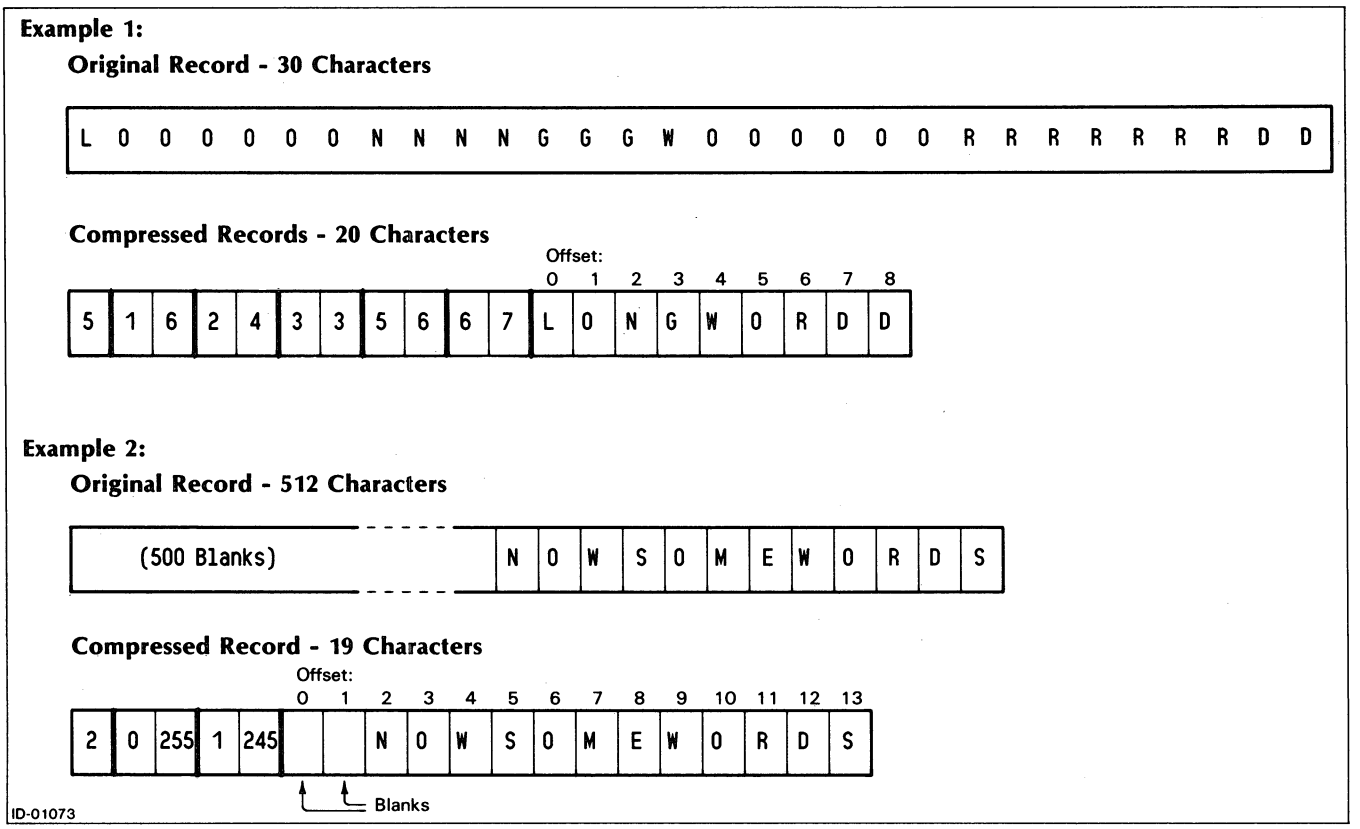


Figure 2-11. Examples of Data Record Compression

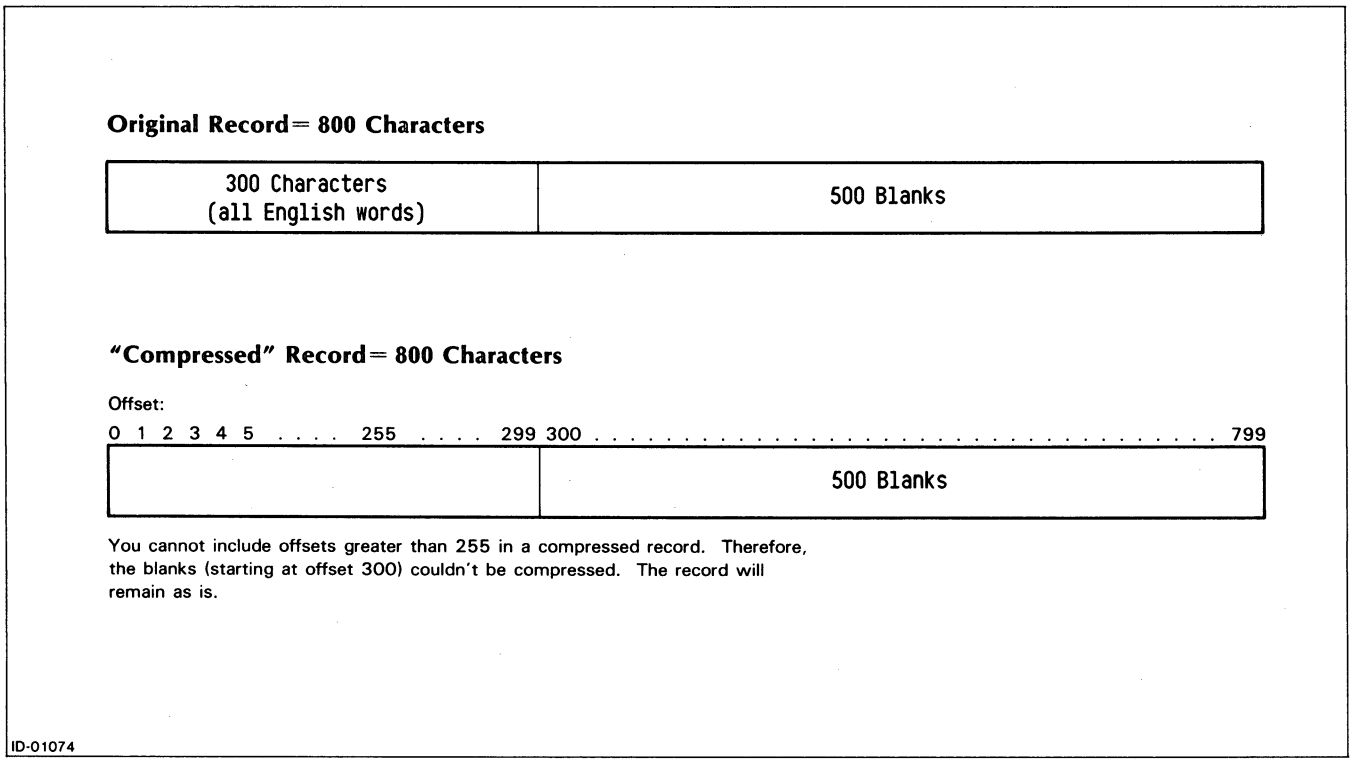


Figure 2-12. Example of Data Record That Will Not Be Compressed

When to Use Data Record Compression

Data record compression is most helpful when all of the following conditions are true:

- Your data records contain a lot of identical characters in a row (usually blanks or zeros).
- You don't have much disk available.
- Your consecutive identical characters occur before the 256th character in the data record.

As with key compression, you never need to concern yourself with data record compression once you turn it on. Treat your database in the normal manner. When you look at a compressed record, the system will expand it to its original contents before you see it. Since the system does have to expand the compressed record, this slows system performance slightly.

Optimized Record Distribution

With optimized record distribution, you can store the index and database volumes that you use most frequently on your fastest access devices. This is a useful feature if you have a large installation with different types of storage devices. The default for this parameter is no optimized record distribution. We list the default values for all parameters in Table 2-1. See Chapter 10 for more information.

Error Codes

We list the INFOS II error codes in Appendix A. If your program encounters an error, you can issue the MESSAGES command from the CLI to determine the cause of the error. Alternatively, you can issue the AOS call ?ERMSG, as described in your *AOS Programmer's Manual,* to determine how to handle the error.

Table 2-1. INFOS II Parameters

Parameter		Default Value	Alternative Values
Main Index	Filename	N/A	Any valid AOS filename
	Access Method	DBAM	ISAM
	Subindex Levels	ISAM = 0 DBAM = 1	None DBAM = 0 to 31
	Space Management	No	Yes
	Maximum Key Length	255 bytes	1 to 254 bytes
	Root Node Size	2042 bytes. (page size minus 6)	Up to 4090 bytes. If > 2042, you must use the 4096-byte page size. Minimum to allow 3 keys.
	Partial Record Length	0 bytes	1 to 255 bytes
	Allow Duplicate Keys	No	Yes
	Key Compression	No	Yes
	Optimized Record Distribution	No	Yes
	Page Size	2048 bytes	4096 bytes
	Number of Volumes	1	2 to 32 (2 to 16 for differential files)
	Volume Name	VOLnn, where nn is the volume number, beginning at 01.	Any allowable name except DVLnn, where nn is the volume number.
	Volume Size	Maximum vol. size: 1 million blocks for 2048-byte page; 2 million blocks for 4096-byte page.	Less than maximum volume size.
	Element Size	32 blocks	For 2048-byte page, any multiple of 4. For 4096-byte page, any multiple of 8. Cannot exceed LD size.

(continues)

Table 2-1. INFOS II Parameters

Parameter		Default Value	Alternative Values
Database	Define Database	No	Yes
	Database Name	Filename.DB	Any valid AOS filename
	Space Management	No	Yes
	Optimized Record Distribution	No	Yes
	Data Record	No	Yes
	Page Size	2048 bytes	4096 bytes
	Number of Volumes	1	2 to 32 (Maximum is 16 for differential files.)
	Volume Name	VOLnn, where nn is the volume number, beginning at 01.	Any allowable name except DVLnn
	Volume Size	Maximum size: 1 million blocks for 2048-byte page; 2 million blocks for 4096-byte page.	Any number of blocks.
	Element Size	32 blocks	For 2048 page size, any multiple of 4. For 4096 page size, any multiple of 8. Cannot exceed LD size.

(concluded)

The OPEN/CREATE Command

With the OPEN command, you can create a new AOS INFOS II file or open an existing INFOS II file. Here, we describe how to use the OPEN command to create an AOS INFOS II file, referred to as the OPEN/CREATE request. In Chapter 3 we describe how to use the OPEN command to open an existing INFOS II file.

Once you have completed an OPEN/CREATE operation, you will have a fully defined, empty INFOS II disk file, which you can immediately begin to process.

Your application language will determine how you code the OPEN command. See the manual for your application language for the syntax of the INFOS II OPEN command. We have included the generic syntax for the OPEN command here. Appendix D contains a sample COBOL program, which opens an AOS INFOS II file.

Remember, you can also define and create INFOS II files with the ICREATE utility program described in Chapter 5. With this utility, you can create both standard and nonstandard INFOS II files through the CLI.

Required Argument

Filename: The name of the index file you want to create. This name can be any valid AOS filename. The INFOS II system will use this name with the extension .DB as the database filename unless you specify an alternative filename.

For the index file, you can specify either the filename or the pathname of an index file. However, for the database file, only a filename is permitted.

Parameters

Specify values for one or more parameters, as shown in Table 2-1. The INFOS II system automatically uses default values for those you do not specify.

Your application language might have the facility in which you can create a standard file with all default parameters. If so, you can create the file in this way.

Optional Arguments

Locks: Specify the number of outstanding locks you expect to invoke while you process the file on the current channel. This number must be between 0 and 32. If you don't specify a number greater than 0, you cannot use the lock option while you process the file on this channel. We describe the lock feature in the "Lock/Unlock" section of Chapter 3.

Exclusive Use: You can ask for exclusive use of the index or database or both. This allows you to exclude all other users from your INFOS II file. We describe this feature in Chapter 3.

System Returned Information

Channel Number: The INFOS II system returns a channel number that associates the program with the file. Depending on your application language, you must specify either this channel number or the name of the index for all subsequent commands you issue during the opening of the file.

Examples

1. Assume that the default access method for your application method is ISAM. To create a standard file with default values for all the parameters, code the equivalent of:

```
OPEN: CREATE filename
```

   (If you use the ICREATE utility, the default access method is DBAM. You must specify ISAM if you want it. See Chapter 5 for information on the ICREATE utility.)

2. To create an ISAM file whose maximum key length is 50 bytes, code the equivalent of:

```
OPEN:  CREATE filename
       MAXIMUM KEY LENGTH = 50
```

3. To create an ISAM file in which you intend to use duplicate keys, code the equivalent of:

```
OPEN:  CREATE filename
       ALLOW DUPLICATE KEYS = YES
```

4. To create a DBAM file that will have three levels of subindexes below the main index (four index levels altogether), code the equivalent of:

```
OPEN:  CREATE DBAM filename
       SUBINDEX LEVELS = 4
```

5. To create a DBAM file with a contiguous database of 100,000 blocks for the larger page size, code the equivalent of:

```
OPEN:  CREATE DBAM filename
       DEFINE DATABASE = YES
       PAGE SIZE = 4096
       ELEMENT SIZE = 100000
       NUMBER OF VOLUMES = 1
       VOLUME 01 SIZE = 100000
```

End of Chapter

Chapter 3
INFOS II Commands

In this chapter we describe all of the INFOS II commands, including OPEN and CLOSE. We include the processing options and the access methods you can choose when you issue these commands. How you actually code a command and specify the options you want will depend upon the conventions and syntax of your application language. We describe the INFOS II scientific language interface in Chapter 8. In addition, you can find more information on the syntax of INFOS II commands in the reference manual for your language. Table 3-1 is an alphabetical list of the INFOS II commands.

Table 3-1. INFOS II Commands

Command	Purpose
CLOSE	Close an INFOS II file.
DEFINE SUBINDEX	Create a new subindex and specify its parameters.
DELETE (LOGICAL)	Mark a partial record, a data record, or both, as logically deleted.
DELETE (PHYSICAL)	Physically remove a key, its partial record, and its data record.
DELETE SUBINDEX	Physically delete a subindex or unlink a subindex from a key.
LINK SUBINDEX	Associate a key with an already defined subindex.
OPEN	Open an INFOS II file for creation or update processing.
READ	Retrieve a data record or partial record from an INFOS II file and move it into your program's address space.
REINSTATE	Remove the logically deleted mark from a partial record or data record.
RELEASE LOCKS/POSITION	Release data record and partial record locks and/or reset current position to above the index.
RETRIEVE HIGH KEY	Locate the highest key in the current index and move it into your program's address space.
RETRIEVE KEY	Locate a key in the current index and move it into your program's address space.
RETRIEVE STATUS	Examine the current status of a key and its associated partial record and data record.
RETRIEVE SUBINDEX DEFINITION	Examine the parameter values of the current index.
REWRITE	Update information in an existing data record or partial record.
WRITE	Enter a new key into the main index or a subindex. Write a data record and/or a partial record, if desired.

Most INFOS II commands have several common processing features. These features are optional and are included in the descriptions of each command, later in this chapter. In the following sections, we describe these processing features and the information that the INFOS II system normally returns to your program at the successful completion of an operation. We call the processing options "Auxiliary Processing Features," and the INFOS II system returned information, "System Returned Status."

Auxiliary Processing Features

There are four general-purpose auxiliary processing features that you control:

- Set Current Position.
- Suppress Database Access.
- Suppress Partial Record Processing.
- Lock/Unlock.

Inversion is a special-purpose processing feature that you use with certain of the INFOS II commands. With the WRITE and REWRITE commands you can perform database inversion, and with the OPEN command you can do file inversion. We describe these procedures under "Linking Keys to Existing Data Records" and "Creating a New Index for an Existing Database" in Chapter 4.

Set Current Position

Current Position in an INFOS II file is simply a place holder. The INFOS II system uses current position as a reference point when it uses relative access to move within an index. When you are processing a file, you can change the current position with each command you issue, or you can retain the initial current position.

The INFOS II system automatically sets current position above the main index when you open a file; you control it after that. You can set the current position on the key you access for any command you issue. The system will set the current position after successfully completing the requested operation.

Of course if you specify Set Current Position with a DELETE command, or if you were previously positioned on the key that you delete, the system cannot set the current position on the accessed key because it no longer exists. Instead, it sets current position on the key preceding the one that you deleted. If the key you deleted was the first in a subindex, your current position will be in front of the subindex. Similarly, if you issue a DELETE SUBINDEX command with Set Current Position, and your current position is in that subindex, then at the completion of the command, current position will be on the key under which the subindex previously resided.

In general, you must explicitly specify Set Current Position if you want to set it on an accessed key. Otherwise, you will remain positioned on the last established current position. For example, let's say you are positioned above a customer-name index, and you access the key for customer Jones. You can set the current position on that key, and the place holder will remain there until you reset it on another request, or close the file. If you do not set the current position on Jones, the current position will remain above the index until you set it somewhere else. You will find more information on current position in the "Relative Access" section of this chapter.

Suppress Database Access

When issuing a command that normally reads or writes a data record, you can use the Suppress Database Access feature. This prevents the record from being returned to you on a Read request, and allows you to enter a key without a data record on a Write request. But you must specify Suppress Database Access if you don't want the system to gain access to the database as a normal part of executing such a command.

For example, keys in a selector subindex normally do not have a record associated with them. If you want to enter a new selector key, you can specify Suppress Database Access with the Write request. You will then have a new selector key with no associated data record.

Or if, for example, you want to set current position on a key and read its partial record, but don't want to read its data record, you can issue a Read request with Set Current Position and Suppress Database Access. The system will set current position on that key and return its partial record to your program, but not its data record. You can save time doing this because the system does not have to obtain access to the disk to retrieve the data record. It must only gain access to the key. This will also prevent you from getting a lock error if another user has locked the data record associated with the accessed key. (See the "Lock/Unlock" section, below.)

Suppress Partial Record Processing

You can suppress partial record processing when issuing a request that normally reads or writes partial records. In these requests, you must explicitly specify Suppress Partial Record Processing if you do not want to read or write a partial record.

For example, if you want to update a data record and do not want to change the partial record, specify Suppress Partial Record Processing when you issue the REWRITE command. The system will not read the partial record already in the file. In addition, you will not get a lock error if the partial record is locked and you suppress partial record processing. (See the "Lock/Unlock" section, below.)

Lock/Unlock

You might want to prevent other users from gaining access to a partial record or a data record while you process it. With the lock option, you can lock a record, process it, and then unlock it and return it to general use when you are finished. Locks are allowed on partial records even if they are zero bytes long.

Note, however, that if you have suppressed the partial record, you cannot lock it; likewise, if you suppress the data record, you will not be able to lock it.

All locks, whether on partial records or data records, are binary: each lock is either on or off. You can, for example, lock a data record when using one command and subsequently issue another command that also specifies to lock the same data record. The record is not locked twice; a single unlock will return it to general use.

NOTE: Only the user who issued a lock on a particular record can unlock that record.

For example, suppose you want to read, modify, and rewrite a record to which other users have access. To prevent other users from gaining access to the record while you are updating it, lock the record when you read it. As long as it is locked, no one else can use it. After you rewrite the modified record, unlock it. It is then available for general use. This will ensure that other users will retrieve the record with its most up-to-date information.

The lock option is useful when you want to update a partial or data record. But using it at other times could prevent other users from obtaining information they need.

If you lock a record on a particular channel, you can gain access to it any number of times while on that channel. A user on any other channel cannot gain access to the record and will receive an error message if he or she attempts to do so.

If you receive a lock error from another user's partial record or data record lock, you can try the same command again, specifying Suppress Partial Record Processing or Suppress Database Access. If only the partial record is locked, you can gain access to the data record; and if only the data record is locked, you can obtain the partial record.

Outstanding Locks

You can specify the maximum number of locks you expect to have outstanding at any one time on the channel you will open. This refers to the channel you open with the OPEN command or with the INQUIRE utility (see Chapter 6 for information about INQUIRE). You can specify 0 to 32 locks. The system default is 0 (no locks). With no locks, you cannot lock any partial or data records.

We recommend that you specify as few outstanding locks as possible, for the greatest efficiency. The INFOS II system's record-locking algorithm optimizes the performance of a file with no locks set. It does this regardless of how you open the file. Note that if you attempt to exceed the maximum number of locks allowed, you will receive the following error:

(IOTML) LOCK REQUEST EXCEEDS MAXIMUM NUMBER OF LOCKS REQUESTED AT OPEN

Let's look at an example using several outstanding locks. Suppose a single business transaction affects the contents of four different types of records: accounts receivable, customer invoice, goods-in-stock, and parts reorder. You want to update an account, and need to modify one record of each type to do so. If you specified at least four outstanding locks when you opened the file, you can lock all four records before you update them. Read each affected record into its own data area, specifying a lock each time. Then, examine each record, modify it as necessary, and rewrite it. After rewriting the last modified record, issue a RELEASE LOCKS command and the INFOS II system will unlock all of your currently locked records. Alternatively, you could specify an unlock on each REWRITE, returning each record to general use as soon as it's updated.

NOTE: If you lock a data record, and then physically delete any key that points to the locked record, the data record lock will be cleared. A logical delete, however, does not affect the lock. Another user will not be able to delete a key that points to any record that you have locked. (See the command descriptions for physical and logical delete, later in this chapter.)

The INFOS II system automatically unlocks all partial and data records before executing the CLOSE command. If you have locks outstanding and your program terminates abnormally (without a CLOSE command) the INFOS II system removes all locks before disassociating your program from its assigned channel number.

Locks have an effect on the following INFOS II commands: OPEN, CLOSE, DELETE, REINSTATE, DELETE SUBINDEX, READ, WRITE, REWRITE, and RETRIEVE STATUS. We detail the specific effects of locks on these commands later in this chapter under the command descriptions.

Local and Global Locks

Some languages refer to a partial record lock as a *local lock* and a data record lock as a *global lock*. Check your language manual to see how your application language refers to partial record locks and data record locks.

Exclusive Use

Some application languages allow you to request exclusive use of the entire index file, database file, or both, when you use the OPEN command. Check your application language manual to find out if this is possible. When you close an exclusively opened file, the system automatically returns it to general use. If you have exclusive use of a file, then no other user can gain access to any part of that file until you close it.

System Returned Status Information

The INFOS II system automatically returns certain types of information to your program about the keys and records you have gained access to or modified. The information you receive depends on which command you issued. System returned status information includes the following:

- The length of the accessed key, its subindex level, and if it is linked to a subindex.
- The partial record length.
- The length of the data record associated with the accessed key and if it is longer than you specified for data record length.
- The locked status of the accessed partial record or data record. (The INFOS II system will give you this information only if you have not suppressed access to the record.)
- A logically deleted status flag for the partial record or data record. (See the DELETE (LOGICAL) command description for details on logical deletion.)
- A duplicate key status flag.
- The occurrence number of the accessed key, if it is a duplicate. (We explain occurrence numbers in Chapter 4.)
- Data record feedback (0 if no data record). This is the data record's logical address for the accessed key. The system returns the feedback as a 32-bit, internal-format, binary integer. (See "Linking Keys to Existing Data Records" and "Creating a New Index for an Existing Database" in Chapter 4 for more detail.)
- Data Record Merit Factor. This only applies if you use optimized record distribution. (See Chapter 10 for information.)

Table 3-3 at the end of this chapter summarizes which commands receive what types of information. In addition, each command description notes what type of status information the system returns to you upon completion of that command.

Access Methods

You must specify an access method for each INFOS II command except OPEN, CLOSE, and RELEASE LOCKS/POSITION. Three types of access are available:

- Keyed access.
- Relative access.
- A combination of keyed and relative access (for multilevel DBAM files only).

We will explain each access method in relation to Figure 3-1, which is a simplified illustration of a multilevel index.

Relative Access

A major benefit of an index file is that the system maintains the information necessary to link a key with its associated data record. To find a record in the database, the INFOS II system must first gain access to its key. But this doesn't mean you must specify a record's key for every command you issue. You can use the relative access technique, which doesn't require a key.

The term *relative access* indicates that the key to which the INFOS II system will gain access is in an index location relative to some pre-established position. We call this pre-established position *current position,* and the INFOS II system uses it as a home base.

After executing a command, the system returns to the previously established current position, unless you have set a new current position (see our discussion of Set Current Position, earlier in this chapter). If you have reset the current position, then the system is positioned on the key you specified for that command; i.e., the new current position. If you get an error for a request in which you also specified Set Current Position, your current position will remain where it was before you issued the command.

Movement, then, is relative to the current position. When you issue a command with relative access, you can choose one of seven directions of movement or static motion. The following sections describe these options. Figure 3-1 shows the keys used in the examples.

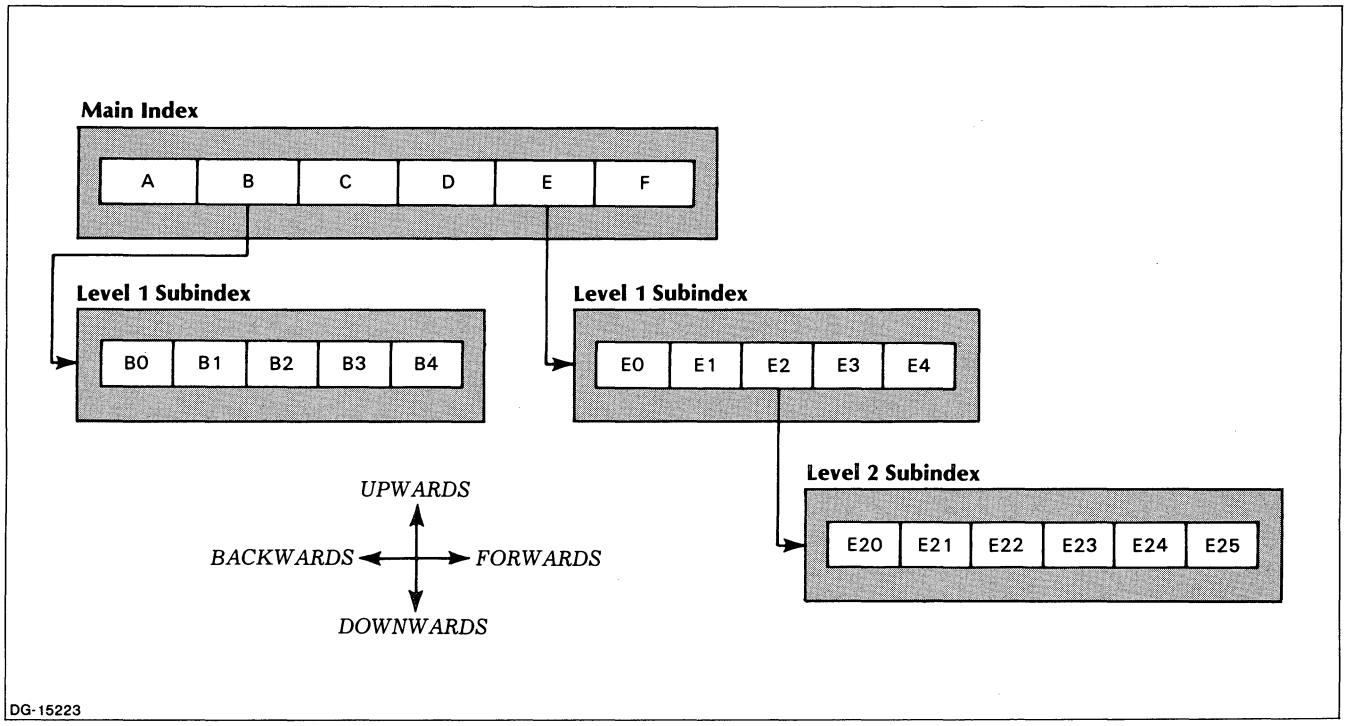


Figure 3-1. Simplified Multilevel Index

Forward Motion

Forward motion means movement toward the next higher key in the subindex, relative to current position. Since the INFOS II system stores keys sequentially by binary value, a "higher" key is one that has a greater binary value than the key at current position. Let's assume that current position is on the key C. To gain access to D, you would issue a command specifying forward motion. If you specify Set Current Position, the current position will be on D at the successful completion of the command. If you don't specify Set Current Position, current position will remain on C when the command completes.

Now let's assume that current position is on the key F. If you specify forward motion, you'll cause an End of Subindex error since there is no key beyond F in that subindex. The current position would remain on F even if you specified Set Current Position.

Backward Motion

Backward motion means movement toward the next lower key in the subindex, relative to the current position. Let's assume that the current position is on key D. To gain access to C you would issue a command specifying backward motion. If you specify Set Current Position, the current position will be on C at the successful completion of the command. Otherwise, current position will remain on D when the command is completed.

If the current position is on the key A, and you issue a command with backward motion, you will cause an End of Subindex error, since there is no key before A. Current position will remain on A whether you specify Set Current Position or not.

Downward Motion

Downward motion means movement down from the current position. There are only two situations in which you can use downward motion:

- When the current position is above the entire index.
- When the current position is on a key linked to a lower level subindex.

If you specify downward motion when current position is not at one of these two places, you will cause an error.

A command using downward motion does not gain access to a key. If you specify downward motion and set the current position from above the entire index, you are positioned in front of the main index. If you specify downward motion and Set Current Position from a key linked to a lower level subindex, you will be positioned in front of that subindex.

For example, if your current position is on the key E2 and you specify downward motion and set current position, your current position will be in front of the key E20 in the level 2 subindex.

Upward Motion

Upward motion means movement up from the current position. If the current position is on key F when you issue a command with upward motion and Set Current Position, you will be positioned above the entire index at the successful completion of the command. (You will also receive a Top Level Warning because you can't move any higher in the index.) If the current position is on a key in a lower level subindex, and you issue a command with upward motion, then you will gain access to the key linked to that lower level subindex. For example, assume the current position is on E25 in the level 2 subindex and you issue a command with upward motion and Set Current Position. The INFOS II system would position you on E2 of the level 1 subindex.

You'll also get a Top Level Warning if you use upward motion when current position is set above the entire index. The system will retain the current position above the main index.

Down and Forward Motion

Down and Forward motion means movement downward from the current position and forward to the first key. There are only two situations in which you can use down and forward motion:

- When the current position is above the entire index.
- When the current position is on a key linked to a lower level subindex.

Down and forward motion from above the index brings you to the first key in the main index. When current position is on a key linked to a lower level subindex, down and forward motion gains access to the first key in that subindex. If you attempt down and forward motion when the current position is not in one of these two places, you will cause an error condition.

For example, if the current position is above the main index and you issue a command with down and forward motion, you will gain access to the key A. If you also set the current position in that command, you'll be positioned on A at the successful completion of the command. Otherwise, you'll be positioned above the main index when the command is completed.

If the current position is on B and you issue a command with down and forward motion and Set Current Position, you will access B0 and set the current position there. If the current position is on C and you issue a command with down and forward motion, you will cause an error because C has no lower level subindex under it. If you attempt to move down and forward into a subindex that exists but has no entries, you will receive an End of Subindex error.

Up and Forward Motion

Up and Forward motion means movement upward from the current position to the key linked to the subindex, and then forward from that key to the next higher key in the subindex. Current position must be on a key in a lower level subindex when you issue a command with up and forward motion or you'll get a Top Level Warning.

If current position is on key E4 and you issue a command with up and forward motion and Set Current Position, you will gain access to key F and will be positioned there at the completion of the command. If you then issue a command from F with up and forward motion and Set Current Position, you will receive a Top Level Warning, but the current position will be above the main index.

Up and Backward Motion

Up and Backward motion means movement upward from the current position to the key linked to the subindex, and then backward from that key to the next lower key. The current position must be on a lower level subindex when you issue a command with up and backward motion or you'll receive a Top Level Warning.

For example, if the current position is on B4 and you issue a command with up and backward motion and set the current position, you'll be positioned on key A. If the current position is on A and you issue a command with up and backward motion, you'll receive a Top Level Warning.

Static Motion

Static motion means no movement relative to the current position. You may not want movement in several situations. For instance, in a dynamic update operation, you could read a record, modify its contents, and then issue a REWRITE command with static motion. Or, if you set the current position in a lower level subindex, then you could gain access to any of its keys with commands that specify keyed access and static motion. (See the "Combined Relative and Keyed Access" section, below.)

You must be positioned on a key when you use static motion or you will receive an error. For instance, if you are positioned in front of a subindex you cannot issue a command with static motion without causing an error condition.

Keyed Access

Keyed access, unlike relative access, doesn't depend on current position. To perform keyed access, issue a command specifying the desired key. In a single-level index, you'll specify only one key, but in a multilevel index, you'll specify one key for each level you want the system to descend. This sequence of keys is called a *key path.*

Refer back to Figure 3-1. To read the data record for the key C, you would issue a keyed READ command that specifies the key C. To read the data record for the key E22, issue a keyed READ command and specify the key path E, E2, E22.

The INFOS II system always searches for the specified key from above the main index, regardless of the current position. For example, assume the current position is on the key B3. To gain access to the data record associated with the key E4, specify a keyed READ and the key path E, E4. If you don't set the current position on that command, you will still be positioned on key B3 at the completion of the READ command.

You can set the current position with keyed access just as you can with relative access. For example, you could do a keyed READ and specify the key C and Set Current Position. You might then want to do three READ requests in which you specify forward motion and Set Current Position to gain access to the data records for the keys D, E, and F.

Combined Relative and Keyed Access

In a multilevel index, you might want to combine relative and keyed access in a single command. This is more efficient than simple keyed access, and reduces the number of keys you must specify in the key path.

Let's assume that you just gained access to key E22 and set the current position. The next data record you want is associated with the key E25. (Remember, you need the three keys E, E2, and E25 to gain access to the data record associated with key E25 in a simple keyed access.) Rather than specifying three keys, you can combine relative and keyed access. When the current position is on a key in a subindex and you want to gain access to any other key in that subindex, specify static motion and the key you want.

On a combined relative and keyed access, the INFOS II system does the following:

1. Performs the relative motion.
2. Gains access to the specified key in the key path of the destination subindex (from the relative motion).

Therefore, if current position is on the key E22 and you specify static motion and the key E25, the system gains access to E25 in the subindex containing E22, since that is where you last set the current position.

Now let's assume current position is on key E25 and you want to obtain the data record associated with the key E4. Specify upward motion and the key E4. The upward relative motion positions you on the key E2 in the level 1 subindex. The INFOS II system then gains access to the first key in the key path in that subindex (E4).

Suppose the current position is on the key E and you want to obtain the data record associated with the key E20. Specify downward motion and the key path E2, E20. The downward motion places you in front of the level 1 subindex. The INFOS II system then locates the first key in your key path (E2), then goes down to the second key in the key path (E20).

Summary

When you combine relative and keyed access in a single command, specify the following:

- One of three directions of motion: static, upward, or downward.
- A complete key path to the key you want, starting from the subindex you reached by relative motion.

The INFOS II system performs the relative motion first, then gains access to the first key in the key path of the destination subindex.

Data Record and Partial Record Areas

There are two more INFOS II terms that you should understand before reading the command descriptions: *Data Record Area* and *Partial Record Area.* These areas are buffers in your address space where you actually process data. Perhaps you know these areas as Work Areas or as Internal Buffers. They are the areas to and from which INFOS II moves data on a READ, WRITE, or REWRITE command.

Often a database contains more than one type of record; for example, invoice, accounts receivable, and accounts payable. Usually records of the same type have the same length, and those of different types are apt to have different lengths. You can allocate as many data record areas as you need, each of any length, to accommodate the different types of records you want to process. Also, the main index and each unique subindex in a multilevel file can have fixed-length partial records. However, no two subindexes in the same file need to have partial records of the same length. You can allocate as many partial record areas as you have subindexes, and make each an appropriate size. Just remember that the partial record length remains fixed for every key in any one subindex.

When you issue a READ command, for example, specify by name or address (depending on your application language) the area to which the INFOS II system should return the data record. To write a partial record, for example, specify by name or address the area from which the system should take the partial record.

Error Codes

We list the INFOS II error codes in Appendix A. If your program encounters an error, you can use the CLI MESSAGE command with the code number of the error to determine what type of error you received. Alternatively, you can issue the AOS call ?ERMSG to determine how to handle the error. (?ERMSG is described in your *AOS Programmer's Manual.*) Check the programming manual for your application language for further information on INFOS II error messages. Consult Appendix F, "Operating the INFOS II Process," for information on how to install INFOS II error files.

Your request packets are in an undefined state after an error condition. The actual contents of your packets depends upon how much of the request INFOS II processed before it found the error. This is not the case, however, when you receive warning messages; your packets are intact if you receive a warning. See Chapter 9 for more information about packets.

Use of Examples

The examples we include with each INFOS II command in this chapter do not reflect the syntax of a particular language. As we said in the beginning of the chapter, you should check your application language manual to obtain the specific syntax you must use. Our generic examples give you the parameters and quantities you need to specify in your application language to solve these particular problems.

We use the same sample INFOS file in all the examples. You will see this file in Figure 3-2 at the end of the chapter.

CLOSE

Close an INFOS II file.

Issue the CLOSE command at the end of a processing run to disassociate your program from the file. The INFOS II system automatically writes to the file all modified index and database information currently in the system buffers, releases the current position, and deactivates the channel number it assigned when the file was opened. The system also unlocks any partial or data records that were locked on that channel.

The INFOS II system automatically closes (without a CLOSE command) all files opened by a process if the process terminates abnormally.

Required Argument

Filename or Channel Number:	Which you must use depends on your application language.

Optional Arguments

There are no optional arguments for the CLOSE command.

System Returned Status

The INFOS II system does not return any status information to you when it completes a CLOSE request.

Example

Your current position doesn't matter. Simply code the equivalent of:

```
CLOSE: FILENAME = EXAMPLE
```

DEFINE SUBINDEX

Create a new subindex.

Use this command to specify the parameters for the subindex you want to create. Position yourself on the key under which you want to create the new subindex. This key must reside in a subindex that can be linked to lower level subindexes. In addition, the key can not be already linked to a subindex when you issue this command. The new subindex cannot exceed the maximum number of levels allowed for this index file or you will receive an error when you issue this command. See the "Defining Subindexes" section of Chapter 4 for more information.

Required Arguments

Access Method:	Keyed, relative, or keyed and relative.
• Keyed Access:	Specify a complete key path.
• Relative Access:	For a single-level index, specify forward, backward, down and forward, or static motion. For a multilevel index, specify any of these directions or one of the following: up, up and forward, or up and backward.
• Keyed and Relative Access:	For a single-level index, specify static or downward motion and a key path. For a multilevel index, specify up, down, or static motion and a key path.
Filename or Channel Number:	Which you must use depends on your application language.

Optional Arguments

Set Current Position:	You can ask the system to set the current position on the key that you gained access to for this request.
Maximum Key Length:	You can specify a maximum key length of 1 to 255 bytes. If you don't specify one, the INFOS II system automatically assigns a maximum key length of 255 bytes.
Partial Record Length:	You can specify a partial record length of 0 to 255 bytes. Specifying 0 length means that you will not use partial records with the keys you'll write to the subindex. The INFOS II system automatically assigns the 0 length if you do not specify a partial record length.
Root Node Size:	The default root node size is 2042 if you created the file with a 2048-byte page size, or 4090 if you created the file with a page size of 4096 bytes. These are also the maximum root node sizes. You can specify a different root node size as long as it is large enough to contain three maximum length index entries, but it cannot exceed the default size (6 bytes less than page size).
Root Node Merit Factor:	If you use optimized record distribution, you can choose a root node merit factor between 1 and 32. See Chapter 10 for more information.
No Subindexing:	The INFOS II system will allow subindexes under the subindex you are currently defining unless you set this option.
Duplicate Keys:	If you do not specify duplicate keys, the system will not let you write multiple keys with the same name to the subindex.

System Returned Status

The INFOS II system sends the following information at the successful completion of a DEFINE SUBINDEX command.

Duplicate Key Status: If the key is a duplicate, the INFOS II system notifies you of this and returns the key's occurrence number.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is on C. To define A's subindex, code the equivalent of:

```
DEFINE SUBINDEX: FILENAME = EXAMPLE
KEY = A
```

   A *standard* subindex, with the following parameter values, results:

   - Maximum key length is 255 bytes.
   - Partial record length is 0.
   - Subindexes are allowed under the keys in this subindex.
   - Duplicate keys are not allowed.

2. Assume the current position is on D. To define E's subindex, which will have a maximum key length of 4 bytes and a partial record length of 12 bytes, code the equivalent of:

```
DEFINE SUBINDEX:
        FILENAME = EXAMPLE
        FORWARD
        MAXIMUM KEY LENGTH = 4
        PARTIAL RECORD LENGTH = 12
        SET CURRENT POSITION
```

   You can define subindexes under the keys in the subindex, but you cannot write duplicate keys into it.

DELETE (LOGICAL)

Mark a partial record, a data record, or both as logically deleted.

A logical deletion marks the partial record or its associated data record or both as logically deleted. A logical deletion does not remove or modify any of a file's contents, except to set the logically deleted status bit. It also does not prevent subsequent access to partial records or data records. The INFOS II system informs users that a partial or data record has been marked as logically deleted.

Logical deletion is useful in many situations, including the following:

- You want to remove certain records temporarily, but be able to reinstate them at a later time.
- You want the opportunity to double-check before you physically delete a record.
- You want to save the deleted records for a special report program or for after-hours processing.
- You want to use a special flag system on certain records to signal users that the record is logically deleted, although it can still be processed like the others.

You cannot perform a logical deletion on a partial record or data record if it has been locked on another channel. If you attempt to do this, you will receive an error. If you attempt to logically delete a partial record or data record that is already logically deleted, the record will remain logically deleted.

Required Arguments

Access Method:	Keyed, relative, or keyed and relative.
• Keyed Access:	Specify a complete key path.
• Relative Access:	For a single-level index, specify forward, backward, down and forward, or static motion. For a multilevel index, specify any of these directions or one of the following: up, up and forward, or up and backward.
• Keyed and Relative Access:	For a single-level index, specify static or downward motion and a key path. For a multilevel index, specify up, down, or static motion and a key path.
Filename or Channel Number:	Which you must use depends on your application language.

Optional Arguments

Set Current Position:	You can have the system Set Current Position on the key you gained access to for this request.
Partial Record Logical Delete:	You can logically delete just the partial record and keep the data record as it is.
Data Record Logical Delete:	You can logically delete just the data record and keep the partial record as it is.
Lock/Unlock:	You can choose to lock or unlock the partial record, the data record, or both, on a logical delete request. If you lock a partial record or data record, and then logically delete it without unlocking it, it remains locked.

System Returned Status

The INFOS II system returns the following information when it completes a logical deletion.

Duplicate Key Status: If the key is a duplicate, the INFOS II system informs you of this and returns its occurrence number.

Locked Status: The system tells you if the partial record, the data record, or both, is locked (whichever you have marked as logically deleted).

Index Level: The system tells you which index level the logically deleted key is in.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is above the index. To mark the data record and partial record associated with the key C as logically deleted, code the equivalent of:

```
DELETE:     FILENAME = EXAMPLE
            KEY = C
            LOGICAL = PARTIAL RECORD,DATA RECORD
```

2. Assume the current position is on A. To mark the A1 data record as logically deleted but not the partial record associated with the A key, code the equivalent of:

```
DELETE:     FILENAME = EXAMPLE
            DOWN AND FORWARD
            LOGICAL = DATA RECORD
```

3. Assume the current position is on A1. To mark the partial record associated with the B key as logically deleted, but not the B data record, code the equivalent of:

```
DELETE:     FILENAME = EXAMPLE
            UP AND FORWARD
            LOGICAL = PARTIAL RECORD
```

DELETE (PHYSICAL)

Physically remove a key, its partial record, and its data record.

A physical deletion removes a key and its partial record, or a key, its partial record, and its associated data record from the file.

If, through inversion, you have associated the data record with more than one key, you can delete one of the keys without deleting the record. In this case, the record's use count decreases by one. When the use count drops to 0, the system deletes the data record (see Chapter 1 for more information on use counts).

Physically deleting a key removes any locks on the associated partial record and data record if the delete request comes from the same channel that initially placed the lock there. You will receive an error if you try to physically delete a partial record or data record that a user on another channel has locked.

You cannot physically delete a key if any of these conditions exist:

- Another user is positioned on the key.
- A subindex has been defined under that key.
- The partial record or data record associated with that key has been locked by another user.

The INFOS II system never removes a data record without deleting a key. It will also delete any key that is no longer linked to a subindex.

Required Arguments

Access Method:	Keyed or keyed and relative.
• Keyed Access:	Specify a complete key path. The INFOS II system deletes the matching key in the main index or subindex. In the case of duplicate keys, specify the occurrence number of the duplicate you want to delete.
• Keyed and Relative Access:	For a single-level index, specify static or downward motion and a key path. For a multilevel index, specify up, down, or static motion and a key path from the subindex you reach with the relative motion. The INFOS II system deletes the key in the appropriate subindex.
Filename or Channel Number:	Which you must use depends on your application language.

Optional Argument

Set Current Position:	If you specify Set Current Position on the deleted key, the system sets it on the previous key in that index. If you delete the first key in the index, the system will set the current position in front of that index or in front of the main index.

System Returned Status

The INFOS II system returns the following information when it completes a physical deletion.

Duplicate Key Status: The INFOS II system informs you if the key is a duplicate, and returns its occurrence number.

Locked Status: If another user has the partial record or data record locked, you cannot delete the key. The INFOS II system informs you if this is the case.

Index Level: The system tells you which index level the key is in.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is on A2. To physically delete the key and record E2, code the equivalent of:

```
DELETE:     FILENAME = EXAMPLE
            KEY = E, KEY = E2
```

2. Assume the current position is on A2. To physically delete the key and record A1, code the equivalent of:

```
DELETE:     FILENAME = EXAMPLE
            STATIC KEY = A1
```

DELETE SUBINDEX

Unlink a subindex from a key or physically delete a subindex.

For this command, gain access to a key that is linked to a subindex for either of these reasons:

- The subindex was defined under that key through a DEFINE SUBINDEX command.
- The key has been associated with the subindex through a LINK SUBINDEX command

You then issue the DELETE SUBINDEX command. The system will not delete the subindex if either of the following are true:

- One or more of the keys in the subindex are linked to other subindexes.
- Another user has a current position in the subindex.

If either of these conditions exists, the system will return an error.

The system also will not delete the subindex if any other key is currently associated with it. Instead, it will remove the link between the accessed key and the subindex.

When the INFOS II system physically deletes a subindex, it first deletes those data records that are associated exclusively with the keys in the subindex. The system then deletes the subindex and its contents. If the current position is set on a key in that subindex, and you then issue the DELETE SUBINDEX command, then after the operation is complete, the current position will be set on the key under which the subindex was defined.

Required Arguments

Access Method:	Keyed, relative, or keyed and relative.
• Keyed Access:	Specify a complete key path.
• Relative Access:	For a single-level index, specify forward, backward, down and forward, or static motion. For a multilevel index, specify any of the above directions or one of the following: up, up and forward, or up and backward.
• Keyed and Relative Access:	For a single-level index, specify static or downward motion and a complete key path. For a multilevel index, specify up, down, or static motion and a key path.
Filename or Channel Number:	Which you must use depends on your application language.

Optional Argument

Set Current Position:	You can ask the system to set the current position on the key you gained access to for the DELETE SUBINDEX request.

System Returned Status

The INFOS II system returns the following information at the completion of a DELETE SUBINDEX command.

Duplicate Key Status:	If the key is a duplicate, the INFOS II system informs you of this and returns its occurrence number.
Partial Record Length:	If the key has a partial record, the system returns its length.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is above the index. To delete E's subindex, code the equivalent of:

```
DELETE SUBINDEX:        FILENAME = EXAMPLE
                        KEY = E
```

2. Assume current position is on A. To delete A's subindex, code the equivalent of:

```
DELETE SUBINDEX:        FILENAME = EXAMPLE
                        STATIC
```

LINK SUBINDEX

Associate a key with an already defined subindex.

Use this command to associate a key with an already defined subindex. You must provide the system with positioning information for two keys: a *source key* and a *destination key.*

The source key is the key associated with the subindex to which you intend to link another key (the destination key). The source key may be the key under which the subindex was originally defined, or a key that was linked to the subindex with a previous LINK SUBINDEX command. The destination key is the key that will be linked to the subindex after the LINK SUBINDEX command is completed.

Required Arguments

Filename or Channel Number:	Which you must use depends on your application language.
Access Method:	You must specify one of the following access methods for the source key and one for the destination key.
• Keyed Access:	Specify a complete key path.
• Relative Access:	For a single-level index, specify forward, backward, down and forward, or static motion. For a multilevel index, specify any of the above directions or one of the following: up, up and forward, or up and backward.
• Keyed and Relative Access:	For a single-level index, specify static or downward motion and a key path. For a multilevel index, specify up, down, or static and a key path.

NOTE: If you specify relative access, motion is relative to the key on which the current position is set. This is true for both source and destination keys. If you set the current position on the source key, and use relative access for the destination key, motion is relative to the source. But if you don't set the current position on the source key, motion is relative to the key on which the current position was set before you issued the LINK SUBINDEX command.

Optional Argument

Set Current Position: You can Set Current Position on the source key, the destination key, or both. If you set the current position on both keys, the system will set current position on the source key, and then on the destination key, so that at the completion of the LINK SUBINDEX request, the current position will be on the destination key. (See Table 3-2, below.)

Table 3-2. Set Current Position Option with the LINK SUBINDEX Command

Set Current Position Specified?		Current Position at Completion of Command
Source Key	Destination Key	
No	No	C
No	Yes	B
Yes	No	A
Yes	Yes	B

Keys:

A = Source Key.

B = Destination Key.

C = Key on which the current position is set prior to the LINK SUBINDEX command.

System Returned Status

The INFOS II system sends the following information at the completion of a LINK SUBINDEX request.

Duplicate Key Status: If either the source key or destination key is a duplicate, the INFOS II system informs you of this and returns the key's occurrence number.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is on D. To link B to A's subindex, code the equivalent of:

```
LINK SUBINDEX:      FILENAME = EXAMPLE
                    SOURCE KEY = A
                    DESTINATION KEY = B
```

2. Assume the current position is on E. To link D to E's subindex, code the equivalent of:

```
LINK SUBINDEX:      FILENAME = EXAMPLE
                    SOURCE = STATIC
                    DESTINATION = BACKWARD
                    SET CURRENT POSITION (ON DESTINATION KEY)
```

Current position is on D at the completion of the operation.

OPEN

Open an INFOS II file.

You can use the OPEN command to create an INFOS II file or open one that already exists. We will discuss how to open an existing file here. For information on how to use the OPEN command to create an INFOS II file, see Chapter 2.

When you open an existing file, you do not need to provide the file characteristics you specified at creation time (key compression, page size, etc.). The INFOS II system returns them to you upon completion of the OPEN request.

Required Argument

Filename: This is the same name you used to create the file. You don't need to supply the database filename. The INFOS II system maintains the link between an index and its database.

Optional Arguments

Outstanding Locks: Specify the maximum number of outstanding locks you expect to invoke at the same time on the channel returned by this OPEN request. The number must be between 0 and 32. If you specify 0, you cannot use the lock option while you have the file open on this channel.

Exclusive Use: You can request exclusive use of the index, database, or both.

Read Only Access: If you specify read only access to the file, you don't need to run IVERIFY because you cannot modify the file (see Chapters 5 and 7 for more information on IVERIFY).

System Returned Status

The INFOS II system returns the following information at the completion of an OPEN request.

Channel Number:	This number associates your program with the file. You will specify either this channel number or the file's name with each command you subsequently issue during this processing run.
Index Page Size:	INFOS tells you if the page size is 2048 or 4096 bytes.
Index Volumes:	The system returns the number of index volumes.
Subindex Levels:	The maximum number of subindex levels allowed in this index file.
File Type:	INFOS tells you if this file is an ISAM or DBAM file.
Key Compression:	The system notifies you if key compression is enabled.
Optimized Record Distribution:	The system notifies you if optimized record distribution is enabled.

Additionally, the INFOS II system returns information about the database if you provide a packet for it in your program. It returns the following:

Database Page Size:	INFOS tells you if the page size is 2048 or 4096 bytes.
Database Volumes:	The system returns the number of database volumes.
Data Record Compression:	The system informs you if data record compression is enabled.
Optimized Record Distribution:	The system informs you if optimized record distribution is enabled.

The INFOS II system positions you above the main index at the completion of an OPEN request.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. To open the file shown in Figure 3-2 without the ability to lock records, code the equivalent of:

```
OPEN:    EXAMPLE
```

2. To open the file with the ability to have three outstanding locks, code the equivalent of:

```
OPEN:    EXAMPLE
         LOCKS = 3
```

READ

Retrieve a data record or partial record from an INFOS II file and move it into your program's address space.

Normally, a READ request moves a data record, a partial record, or both into your program's address space. You can use the READ command with the Suppress Database and Suppress Partial Record features to help you avoid lock errors and warning messages such as *(IONDR) WARNING - DATA BASE RECORD NOT PRESENT.*

Required Arguments

Access Method: Keyed, relative, or keyed and relative.

- Keyed Access: Specify a complete key path.
- Relative Access: For a single-level index, specify forward, backward, down, down and forward, or static motion. For a multilevel index, specify any of the above directions or up, up and forward, or up and backward.
- Keyed and Relative Access: For a single-level index, specify static or downward motion and a key path. For a multilevel index, specify up, down, or static motion and a key path.

Filename or Channel Number: Which you must use depends on your application language.

Name or Address: Depending on your application language, you must supply the name or address of the data record buffer, the partial record buffer, or both (unless you suppress access to them). The specified buffer is where you'll find the record after the request is finished.

Buffer Length: You must specify, in bytes, the length of the data record buffer, the partial record buffer, or both. Because partial record lengths are fixed, you don't need to supply the length of the partial record buffer for some application languages. Check your application language manual for more information on reading partial records.

Optional Arguments

Set Current Position: You can have the system set the current position on the key you gained access to for the READ request.

Suppress Database Access: If you suppress access to the database, the INFOS II system will not move the data record associated with the accessed key into your data record area. Therefore, you need not supply a name, address, or length for the data record buffer.

Suppress Partial Record Processing: If you suppress partial record processing, the INFOS II system does not gain access to the partial record. Therefore, you need not provide a name, address, or length for the partial record buffer.

Number of Bytes: You can specify the number of bytes to be moved. Normally, the system moves the entire data record into your data record area. However, if you want only the leading part of the record, specify the number of bytes you want moved. The system will move the specified number of bytes and inform you if the actual record is longer than the length you have specified.

Lock/Unlock: You can lock or unlock the partial record, the data record, or both, if you have not suppressed access to them.

NOTE: You cannot read or lock a partial record or data record if another user has it locked on a different channel. Suppress the partial record or data record, depending on which is locked, if you want to read or lock the record that is not locked.

System Returned Status

The INFOS II system sends the following information at the completion of a READ request.

Duplicate Key Status:	If the key is a duplicate, the system informs you of this and returns its occurrence number.
Key Length:	The system returns the length of the key, in bytes.
Index Level:	The system tells you the index level of the key to which you gained access for the READ request.

If you have not suppressed the data record or partial record, the system returns this information:

Logically Deleted Status:	The INFOS II system informs you if the data record or partial record is logically deleted.
Data Area Overflow:	The system informs you if the length of the data area you specified for the record is less than the record's length.
Data Record Length:	The system returns the length of the data record, in bytes.
Partial Record Length:	The system returns the length of the partial record, in bytes.
Data Record Merit Factor:	If you use optimized record distribution, the system returns the data record's merit factor.
Feedback Information:	The system returns the data record feedback.
Locked Status:	You will receive an error message if you try to read a partial record or data record that another user has locked.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is above the index. To read the C record, code the equivalent of:

```
READ:    FILENAME = EXAMPLE
         KEY = C
         DATA AREA = AREA1
         SUPPRESS PARTIAL RECORD ACCESS
```

2. Assume the current position is above the index. To read the A2 record, code the equivalent of:

```
READ:    FILENAME = EXAMPLE
         KEY = A, KEY = A2
         DATA AREA = AREA1
         SUPPRESS PARTIAL RECORD ACCESS
```

3. Assume the current position is on A2. To read the A999 record and its partial record, code the equivalent of:

```
READ:     FILENAME = EXAMPLE
          STATIC KEY = A999
          DATA AREA = AREA1
          PARTIAL RECORD AREA = AREA2
```

4. Assume the current position is on B. To read the C record, code the equivalent of:

```
READ:     FILENAME = EXAMPLE
          FORWARD
          DATA AREA = AREA1
          SUPPRESS PARTIAL RECORD ACCESS
```

5. Assume the current position is on E. To read the first 50 bytes of the E1 record, code the equivalent of:

```
READ:     FILENAME = EXAMPLE
          DOWN AND FORWARD
          DATA AREA = AREA1
          RECORD LENGTH = 50
```

   Note that if the E1 record is longer than 50 bytes, the INFOS II system will set the data record overflow status bit.

6. Assume the current position is above the index. To read E2's partial record but not its data record, code the equivalent of:

```
READ:     FILENAME = EXAMPLE
          KEY = E, KEY = E2
          SUPPRESS DATABASE ACCESS
          PARTIAL RECORD AREA = AREA2
```

7. Assume the current position is on A. You can establish a new current position from which you can sequentially process the records associated with the keys in A's subindex. Code the equivalent of:

```
READ:     FILENAME = EXAMPLE
          DOWN
          SET CURRENT POSITION
```

   Then loop on:

```
READ:     FILENAME = EXAMPLE
          FORWARD
          DATA AREA = AREA1
          PARTIAL RECORD AREA = AREA2
          SET CURRENT POSITION
```

REINSTATE

Remove the logically deleted mark from a partial record, a data record, or both.

You cannot reinstate a partial record or data record if it was locked through a different channel. If you attempt to do so, you will receive an error. Attempting to reinstate a partial record or data record that is not logically deleted has no effect.

Required Arguments

Access Method:	Keyed, relative, or keyed and relative.
• Keyed Access:	Specify a complete key path.
• Relative Access:	For a single-level index, specify forward, backward, down and forward, or static motion. For a multilevel index, specify any of the above directions or one of the following: up, up and forward or up and backward.
• Keyed and Relative Access:	For a single level index, specify static or downward motion and a key path. For a multilevel index, specify up, down or static motion and a key path.
Filename or Channel Number:	Which you must use depends on your application language.

Optional Arguments

Set Current Position:	You can Set Current Position on the key you gained access to for the REINSTATE request.
Partial Record Reinstate:	You can reinstate just the partial record and leave the data record logically deleted.
Data Record Reinstate:	You can reinstate just the data record and leave the partial record logically deleted.

System Returned Status

The INFOS II system returns the following information at the successful completion of a REINSTATE request.

Duplicate Key Status:	If the key is a duplicate, the INFOS II system informs you of this and returns its occurrence number.
Locked Status:	The system informs you if the partial record or data record that you reinstated is locked.
Partial Record Length:	The system returns the length of the partial record, if one exists.
Index Level:	The system tells you what index level the key is in.

REINSTATE (continued)

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is above the index. To reinstate the key C and the partial record and data record associated with it, code the equivalent of:

```
REINSTATE:      FILENAME = EXAMPLE
                KEY = C
                PARTIAL RECORD,DATA RECORD
```

2. Assume the current position is on A. To reinstate the A1 data record, code the equivalent of:

```
REINSTATE:      FILENAME = EXAMPLE
                DOWN AND FORWARD
                DATA RECORD
```

3. Assume the current position is on A1. To reinstate the partial record associated with the key B, code the equivalent of:

```
REINSTATE:      FILENAME = EXAMPLE
                UP AND FORWARD
                PARTIAL RECORD
```

RELEASE LOCKS or RELEASE POSITION

Release data record and partial record locks. Reset the current position above the index.

Use the RELEASE LOCKS command to release any outstanding locks you have on an INFOS II file. The RELEASE POSITION command resets your current position above the main index.

Required Argument

Filename or Channel Number:	Which you must use depends on your application language.

Optional Arguments

Release Locks:	You can choose to release the locks, but not reset the current position.
Reset Current Position:	You can choose to reset the current position, but not release the locks.
Both:	You can both release locks and reset the current position.

System Returned Status

The INFOS II system does not return status for a RELEASE LOCKS or a RELEASE CURRENT POSITION request.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. To release all currently locked data and partial records, code the equivalent of:

```
RELEASE:    FILENAME = EXAMPLE
            RELEASE LOCKS
```

2. To release the current position and reset it to above the index, code the equivalent of:

```
RELEASE:    FILENAME = EXAMPLE
            RELEASE CURRENT POSITION
```

3. To release all currently locked records and reset current position to above the index, code the equivalent of:

```
RELEASE:    FILENAME = EXAMPLE
            RELEASE LOCKS
            RELEASE CURRENT POSITION
```

RETRIEVE HIGH KEY

Examine the highest key in the main index or a subindex.

The INFOS II system retrieves the highest key in the main index or a subindex in which the accessed key resides. The highest key is the last key in the main index or subindex. You can specify Set Current Position with this command, but you cannot set it on the high key that is retrieved. For example, you can use keyed access to specify a key in a certain subindex, and retrieve the high key in that subindex. If you specify to set current position, it will be set on the key you specified for keyed access, and not on the high key.

Required Arguments

Access Method:	Keyed, relative, or keyed and relative.
• Keyed Access:	Specify a complete key path.
• Relative Access:	For a single-level index, specify forward, backward, down, down and forward, or static motion. For a multilevel index, specify any of the above directions or one of the following: up, up and forward or up and backward.
• Keyed and Relative Access:	For a single-level index, specify static or downward motion and a key path. For a multilevel index, specify up, down, or static motion and a key path.
Filename or Channel Number:	Which you must use depends on your application language.
Name or Address:	Depending on your application language, you must specify the name or address of your data record buffer.

Optional Argument

Set Current Position:	You can set the current position on the key you specify if you use keyed access. You cannot specify to set it on the high key.

System Returned Status

The INFOS II system returns the highest (last) key in the main index or a subindex to the data area you specify. It returns the following information:

Duplicate Key Status:	If the retrieved key is a duplicate, the INFOS II system informs you of this and returns its occurrence number.
Logically Deleted Status:	The system informs you if the partial record is logically deleted.
Subindex Present:	The INFOS II system tells you if the retrieved key has a subindex defined below it.
Partial Record Length:	If the key has a partial record, the system tells you its length, in bytes.
Key Length:	The system tells you the length of the key it retrieved, in bytes.
Subindex Level:	The system tells you the subindex level of the retrieved key.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is on key E. To retrieve the highest key in the subindex to which E is linked, code the equivalent of:

```
RETRIEVE HIGH KEY:          FILENAME = EXAMPLE
                            DOWN
                            DATA AREA = AREA1
```

2. Assume the current position is on B, and you want to retrieve the highest key in the subindex under A. You know one of the keys in that subindex is A2, so you use keyed access and specify key A2, set current position on it, and retrieve the high key. Code the equivalent of:

```
RETRIEVE HIGH KEY:          FILENAME = EXAMPLE
                            KEY = A, KEY = A2
                            DATA AREA = AREA1
                            SET CURRENT POSITION
```

RETRIEVE KEY

Examine a key in the main index or a subindex.

Use the RETRIEVE KEY command to gain access to any key in the INFOS II index. With this request, you position to a key and the system returns it to your data area.

Required Arguments

Access Method:	Keyed, relative, or keyed and relative.
• Keyed Access:	Specify a complete key path.
• Relative Access:	For a single-level index, specify forward, backward, down, down and forward, or static motion. For a multilevel index, specify any of the above directions or one of the following: up, up and forward or up and backward.
• Keyed and Relative Access:	For a single-level index, specify static or downward motion and a key path. For a multilevel index, specify up, down, or static motion and a key path.
Filename or Channel Number:	Which you must use depends on your application language.
Name or Address:	Depending on your application language, you must specify the name or address of the data area to which you want the key returned.

Optional Argument

Set Current Position:	You can Set Current Position on the key you gain access to for this request.

System Returned Status

The INFOS II system returns the key to the data area you specify, and sends the following information at the successful completion of a RETRIEVE KEY request.

Duplicate Key Status:	If the key is a duplicate, the INFOS II system informs you of this and returns its occurrence number.
Logically Deleted Status:	The system informs you if the data record or partial record is logically deleted.
Subindex Present:	The system tells you if the retrieved key has a subindex defined below it.
Data Record Length:	The system reports the length of the data record, in bytes.
Key Length:	The system returns the key length, in bytes.
Partial Record Length:	The system returns the partial record length, in bytes.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is above the index. To retrieve the key C and set current position, code the equivalent of:

```
RETRIEVE KEY:        FILENAME = EXAMPLE
                     KEY = C
                     DATA AREA = AREA1
                     SET CURRENT POSITION
```

2. Assume the current position is on C. To retrieve the key D, code the equivalent of:

```
RETRIEVE KEY:        FILENAME = EXAMPLE
                     FORWARD
                     DATA AREA = AREA1
```

RETRIEVE STATUS

Examine the current status of a key and its associated partial record and data record.

The RETRIEVE STATUS command returns the current status of a key, its partial record, and data record.

Required Arguments

Access Method:	Keyed, relative, or keyed and relative.
• Keyed Access:	Specify a complete key path.
• Relative Access:	For a single-level index, specify forward, backward, down, down and forward, or static motion. For a multilevel index, specify any of the above directions or one of the following: up, up and forward, or up and backward.
• Keyed and Relative Access:	For a single-level index, specify static or downward motion and a key path. For a multilevel index, specify up, down, or static motion and a key path.
Filename or Channel Number:	Which you must use depends on your application language.

Optional Arguments

Set Current Position:	You can ask the system to set the current position on the key for which it retrieved status.
Suppress Database Access:	You can suppress the database to retrieve status information only about the key and partial record.
Suppress Partial Record Processing:	You can suppress the partial record to retrieve status information only about the key and data record.

System Returned Status

The INFOS II system returns the following information at the successful completion of a RETRIEVE STATUS request.

Duplicate Key Status:	If the key is a duplicate, the system informs you of this and returns its occurrence number.
Subindex Present:	The system informs you if the key has a subindex defined below it.
Index Level:	The INFOS II system tells you what index level the retrieved key is in.
Key Length:	The system returns the length of the key, in bytes.

If you have not suppressed access to the data record or partial record, the system also returns this information:

Logically Deleted Status:	The system informs you if the data record or partial record is logically deleted.
Data Record Length:	The system returns the length of the data record, in bytes.
Partial Record Length:	The system returns the length of the partial record, in bytes.
Feedback Information:	The system returns the data record feedback information.
Merit Factor:	The system returns the merit factor of the data record, if you use optimized record distribution.
Locked Status:	The system informs you if the partial record or data record is locked.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is above the index. To retrieve the status of the key A, its partial record and data record, code the equivalent of:

```
RETRIEVE STATUS:        FILENAME = EXAMPLE
                        DOWN AND FORWARD
```

2. Assume the current position is on B. To retrieve the status of the key and data record B, code the equivalent of:

```
RETRIEVE STATUS:        FILENAME = EXAMPLE
                        STATIC
                        SUPPRESS PARTIAL RECORD ACCESS
```

3. Assume the current position is above the index. To retrieve the status of the key and data record A999 code the equivalent of:

```
RETRIEVE STATUS:        FILENAME = EXAMPLE
                        KEY = A, KEY = A999
                        SUPPRESS PARTIAL RECORD ACCESS
```

RETRIEVE SUBINDEX DEFINITION

Examine the parameter values of the main index or a subindex.

Use this command to retrieve the information you used to define the subindex of the accessed key. Your current position must be in or in front of the main index or the subindex you want to examine.

Sometime before you issue the first RETRIEVE SUBINDEX DEFINITION command, you might have to allocate a Subindex Definition Packet. This is a special 16-byte area to which the INFOS II system will return the requested information. (See Chapter 9 for more information on packets.) Check the manual of your application language for more information on this command.

Required Arguments

Access Method: Keyed, relative, or keyed and relative.

- Keyed Access: Specify a complete key path.
- Relative Access: For a single-level index, specify forward, backward, down, down and forward, or static motion. For a multilevel index, specify any of the above directions or one of the following: up, up and forward, or up and backward.
- Keyed and Relative Access: For a single-level index, specify downward or static motion and a key path. For a multilevel index, specify up, down, or static motion and a key path.

Filename or Channel Number: Which you must use depends on your application language.

Name or Address: Depending on your application language, you must specify the name or address of the area that will receive the subindex definition information.

Optional Argument

Set Current Position: You can Set Current Position on the key you gain access to for this request.

System Returned Status

The INFOS II system returns the following information about the main index or subindex at the successful completion of a RETRIEVE SUBINDEX DEFINITION request.

Root Node Size: The system tells you the root node size of the index.

Maximum Key Length: The system reports the maximum allowable length for keys in this index.

Partial Record Length: If keys in the index are allowed to have partial records, the system reports what length they can be.

Duplicate Key Status: The system informs you whether or not duplicate keys are allowed in the index.

Subindexing Status: The system tells you whether or not keys in the index can have subindexes defined below them.

Merit Factor: The system tells you the merit factor of the root node.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is on B. To retrieve the main index definition, code the equivalent of:

```
RETRIEVE SUBINDEX DEFINITION:     FILENAME = EXAMPLE
                                  STATIC
                                  SUBINDEX AREA = AREA3
```

2. Assume the current position is on C. To retrieve the subindex definition of the key A, code the equivalent of:

```
RETRIEVE SUBINDEX DEFINITION:     FILENAME = EXAMPLE
                                  KEY = A
                                  SUBINDEX AREA = AREA3
```

REWRITE

Update information in an existing partial record or data record.

You can use the REWRITE command to do any of the following:

- Update the contents of an existing data record. The record you supply for a REWRITE command need not be the same length as the corresponding record in the database.
- Update the contents of an existing partial record. Note that the partial record you supply for a REWRITE command can be any length up to the maximum length specified in the index. If it is less than the maximum, INFOS II will pad it with null characters.
- Write a new record into the database for a key that does not currently have a data record associated with it.
- Write a new partial record for a key that does not currently have a partial record associated with it.

In the section "Linking Keys to Existing Data Records" in Chapter 4, we show you how to use a REWRITE request to associate two different keys with the same data record.

Required Arguments

Access Method:	Keyed, relative, or keyed and relative.
• Keyed Access:	Specify a complete key path.
• Relative Access:	For a single-level index, specify forward, backward, down and forward, or static motion. For a multilevel index, specify any of the above directions or one of the following: up, up and forward, or up and backward.
• Keyed and Relative Access:	For a single-level index, specify downward or static motion and a key path. For a multilevel index, specify up, down, or static motion and a key path.
Filename or Channel Number:	Which you must use depends on your application language.
Data Record Length:	You must supply the length, in bytes, of the data record, unless you suppress access to it. The system supplies the partial record length because it is fixed.

Optional Arguments

Set Current Position:	You can set the current position on the key you enter for the REWRITE command.
Suppress Database Access:	If you use this option, the INFOS II system system will not modify the data record associated with the key you gained access to for the REWRITE request.
Suppress Partial Record Processing:	If you suppress the partial record, the system will not modify it; it will only modify the data record.
Inversion:	Use inversion to associate more than one key with the same database record. The system will associate an existing record with the key you specify on a REWRITE request with inversion. However, if the key already has a data record associated with it, you cannot use an inverted REWRITE to switch its data record.

Data Record Merit Factor: If you are using optimized record distribution, you can change the merit factor of the data record. See Chapter 10 for more information.

Lock/Unlock: You can lock or unlock the partial record, data record, or both.

NOTE: You cannot rewrite or lock a partial record or data record if it has been locked by another user on a different channel. You can, however, suppress the partial record or data record, depending on which is locked, and then lock and rewrite the record that is not locked.

System Returned Status

The INFOS II system sends the following information at the completion of a REWRITE request.

Duplicate Key Status: If the key is a duplicate, the system informs you and returns its occurrence number.

Index Level: The system tells you what index level the key is in.

If you have not suppressed access to the partial record or data record, the system also returns the following information:

Feedback Information: The system returns the data record feedback information.

Locked Status: If the partial record or data record is locked by another user, the system returns the appropriate error message.

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is above the index. To rewrite the E1 data record, code the equivalent of:

```
REWRITE:    FILENAME = EXAMPLE
            KEY = E, KEY = E1
            DATA AREA = AREA1
            RECORD LENGTH = 60
            SUPPRESS PARTIAL RECORD
            SET CURRENT POSITION
```

2. Assume the current position is on E1. To rewrite the E2 data and partial records, code the equivalent of:

```
REWRITE:    FILENAME = EXAMPLE
            STATIC KEY = E2
            DATA AREA = AREA1
            RECORD LENGTH = 60
            PARTIAL RECORD AREA = AREA2
            SET CURRENT POSITION
```

3. Assume the current position is on E2. To rewrite the partial record for D, but not its data record, code the equivalent of:

```
REWRITE:    FILENAME = EXAMPLE
            UP AND BACKWARD
            SUPPRESS DATABASE ACCESS
            PARTIAL RECORD AREA = AREA2
            SET CURRENT POSITION
```

WRITE

Enter a new key into the main index or a subindex, and write its data record and partial record.

When you enter a new key, you can supply a partial record if you have allocated a partial record length greater than 0 bytes. The WRITE command usually enters a new record into the database also. In the "Linking Keys to Existing Data Records" section of Chapter 4, we show you how to use a WRITE command with inversion to associate two different keys with the same data record.

Required Arguments

Key Name:	You must supply the name of the key that you want to enter into the subindex.
Access Method:	Keyed or keyed and relative. You cannot use relative access alone on a WRITE request.
• Keyed Access:	Specify a complete key path. The INFOS II system writes the last key specified in the path to the subindex at the appropriate level. It determines the correct index level from the number of keys that precede the new key in the key path.
• Relative and Keyed Access:	For a single-level index, specify static or downward motion and a key path. For a multilevel index, specify up, down, or static motion and a complete key path from the subindex that you reach by the relative motion. The INFOS II system writes the key to the appropriate subindex.
Filename or Channel Number:	Which you must use depends on your application language.
Data Record Length:	You must supply the length of the data record, in bytes, if you have not suppressed it. The INFOS II system supplies the partial record length because it is fixed.
Name or Address:	Depending on your application language, you must specify the name or address of the data record area and the partial record area from which you want the system to take the record.

Optional Arguments

Set Current Position: You can set the current position on the key you gain access to for the WRITE request.

Suppress Database Access: If you suppress the database, the system will not write a data record to the database or associate one with the key you entered for the WRITE request. If you suppress the database to write a key, and subsequently perform a READ request on that key, you will receive the following message:

```
(IONDR) WARNING - DATA RECORD NOT PRESENT
```

Suppress Partial Record Processing: If you suppress the partial record, the system will not write one. Otherwise, the INFOS II system automatically writes the contents of the partial record area as the key's partial record.

Lock: You can lock the partial record, the data record, or both.

Duplicate Key Flag: You can set the duplicate key status flag to write a potentially duplicate key. If you don't use this option, you cannot write a key into the main index or a subindex if it has the same name as a key that already exists there. Note that you cannot write duplicate keys in an index unless you used the parameter to allow duplicate keys when you originally defined the index.

Inversion: Use inversion with a WRITE request to associate more than one key with the same data record. If you specify inversion, you must supply the data record feedback. If you specify a data record feedback number of 0, the system acts as if you suppressed the database. If you perform an inverted WRITE request and do not suppress the database, the system will rewrite the data record to whatever you specify in the data area.

Data Record Merit Factor: If you use optimized record distribution, you can specify a merit factor for the data record. (See Chapter 10 for more information.)

System Returned Status

The INFOS II system sends the following information at the completion of a WRITE request.

Duplicate Key Status: If the key is a duplicate, the system informs you of this and returns its occurrence number.

If you have not suppressed access to the data record, you will also receive the following:

Feedback Information: The system returns the data record feedback information for the record you wrote.

WRITE (continued)

Examples

(Refer to Figure 3-2 at the end of this chapter.)

1. Assume the current position is above the index. To write the C key and its associated data record, code the equivalent of:

```
WRITE:      FILENAME = EXAMPLE
            KEY = C
            DATA AREA = AREA1
            RECORD LENGTH = 80
```

2. Assume the current position is on E. To write the E2 key, its associated data record and partial record, code the equivalent of:

```
WRITE:      FILENAME = EXAMPLE
            DOWN KEY = E2
            DATA AREA = AREA1
            RECORD LENGTH = 80
            PARTIAL RECORD AREA = AREA2
```

3. Assume the current position is above the index. To write the key E999 without an associated data record, code the equivalent of:

```
WRITE:      FILENAME = EXAMPLE
            KEY = E, KEY = E999
            SUPPRESS DATABASE ACCESS
```

4. Assume the current position is on B. To write C and a partial record, but not a data record, code the equivalent of:

```
WRITE:      FILENAME = EXAMPLE
            STATIC KEY = C
            SUPPRESS DATABASE ACCESS
            PARTIAL RECORD AREA = AREA2
            SET CURRENT POSITION
```

5. Assume the current position is on C. To write D and a data record without a partial record, code the equivalent of:

```
WRITE:      FILENAME = EXAMPLE
            STATIC KEY = D
            DATA AREA = AREA1
            RECORD LENGTH = 80
            SUPPRESS PARTIAL RECORD PROCESSING
```

Table 3-3 summarizes the INFOS II system's returned status for all commands.

Table 3-3. INFOS II System Returned Status

Commands \ Status Information	Duplicate Key Status	Locked Status	Logical Delete Status	Occurrence No.	Key Length	Partial Record Length	Data Record Length	Data Record Feedback	Data Area Overflow	Data Record Merit Factor	Root Node Merit Factor	Root Node Size	Index Level	Subindexes Allowed	Subindex Present	Open Information*
CLOSE																
DEFINE SUBINDEX	X															
DELETE (Logical)	X	X											X			
DELETE (Physical)	X	X											X			
DELETE SUBINDEX	X					X										
LINK SUBINDEX	X															
OPEN																X
READ	X	X	X		X	X	X	X	X	X			X			
REINSTATE	X	X				X							X			
RELEASE POSITION/LOCKS																
RETRIEVE HIGH KEY	X		X	X	X								X		X	
RETRIEVE KEY	X		X	X	X	X									X	
RETRIEVE STATUS	X	X	X	X	X	X	X	X		X			X		X	
RETRIEVE SUBINDEX DEFINITION	X				X	X					X	X		X		
REWRITE	X	X						X					X			
WRITE	X							X								

*The INFOS II system returns certain information on an OPEN command that it does not return on any other commands. See the OPEN command description for details.

DG-26347

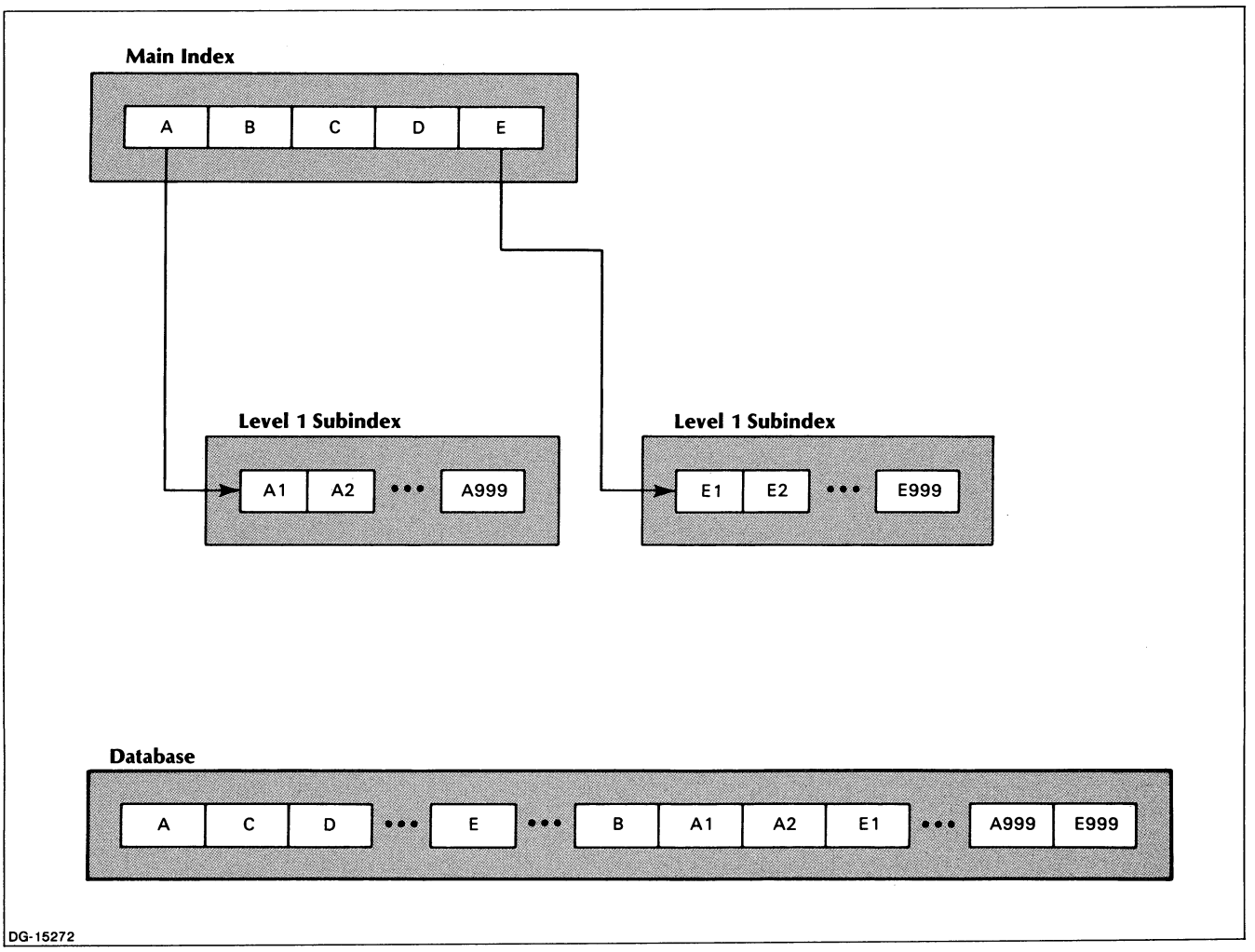


Figure 3-2. INFOS II File EXAMPLE

End of Chapter

Chapter 4
INFOS II File Processing Techniques

In this chapter, we describe some of the ways you can apply INFOS II processing techniques to common application problems. Although these problems might appear to be limited in scope, they pertain to many applications. In this chapter, we pay particular attention to certain processing techniques that we did not explain in detail earlier in this manual: duplicate key processing, generic and approximate search keys, linking subindexes, database inversion, and file inversion.

While we present universal processing problems, our coding examples show just one of many possible solutions. You will undoubtedly have specific coding ideas to suit your own application needs.

Sequential Processing

Let's assume you've created a standard single-level file for your accounts receivable records. The file's name is BILLING and its records are keyed by customer surname, as shown in Figure 4-1. Once a month you process the file sequentially, producing itemized bills for your customers.

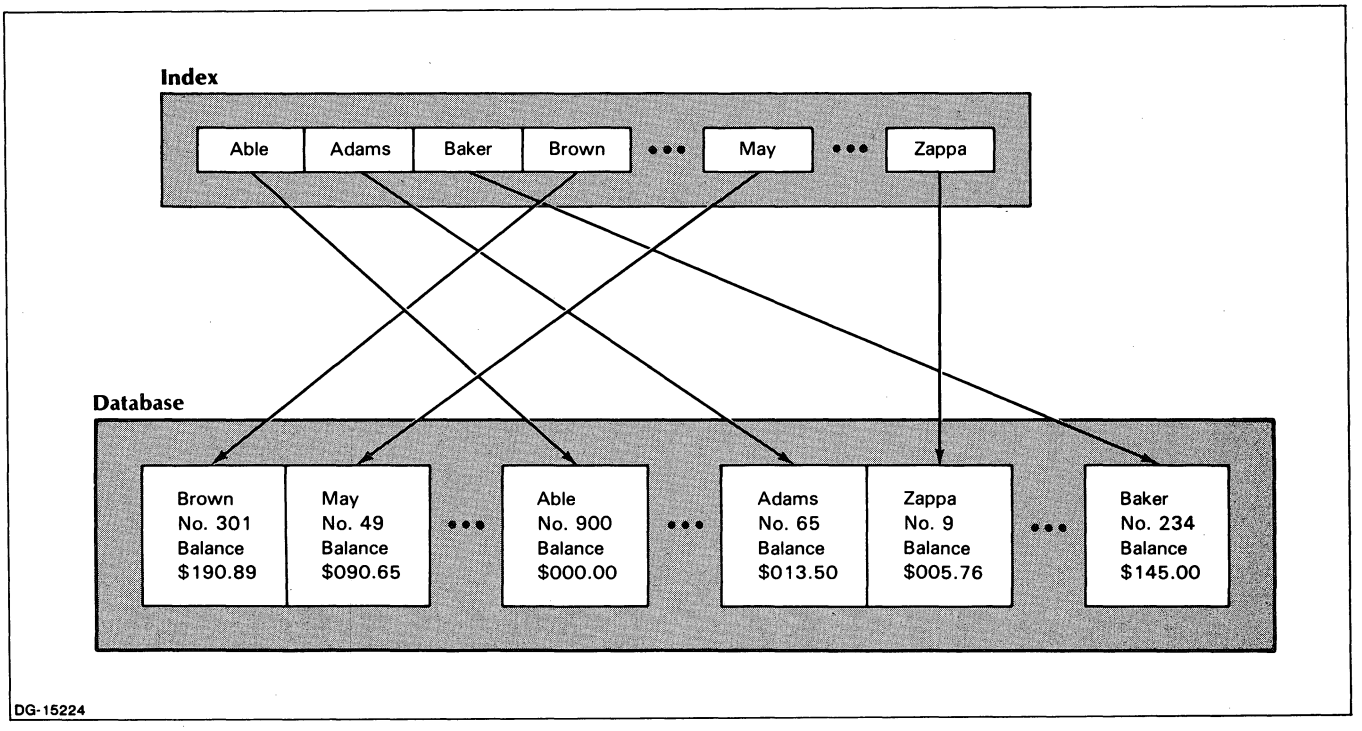


Figure 4-1. INFOS File BILLING

Note that the records in Figure 4-1 are dispersed randomly in the database; but the keys are in sequential order. As we mentioned in Chapter 1, the INFOS II system automatically sorts the keys in sequential order, and maintains the information required to link a key to its associated data record. It does this regardless of the sequence in which you write the records.

To process these records sequentially, open the file and code the equivalent of:

```
READ: FILENAME = BILLING
      DOWN
      SUPPRESS DATABASE ACCESS
      SET CURRENT POSITION
```

NOTE: Some application languages, such as COBOL, automatically perform a READ command with downward motion as part of the OPEN command. If this is the case, you can omit this first step. Check your application language manual for more information on the OPEN command.

Then include a loop with the equivalent of:

```
READ: FILENAME = BILLING
      FORWARD
      DATA AREA = AREA1
      SET CURRENT POSITION
```

Dynamic Updating

Let's assume that you use magnetic tape to record each customer transaction during business hours. Then after business hours you want to update the BILLING file in Figure 4-1. You do not need to sort the transaction tape because you can access each record directly by its key.

You decide to update the BROWN record, so you open the BILLING file. Your current position is now above the index. To dynamically update BROWN's record, code the equivalent of:

```
READ: FILENAME = BILLING
      KEY = BROWN
      DATA AREA = AREA1
      LOCK = DATA RECORD
      SET CURRENT POSITION

REWRITE: FILENAME = BILLING
         STATIC
         DATA AREA = AREA1
         DATA RECORD LENGTH = 80
         UNLOCK = DATA RECORD
```

Locking the data record on the READ request ensures that another user will not gain access to the record until you have finished updating it. If you are in a single-user environment, you need not use the lock. But, we recommend using it in an environment where multiple users can update the file.

We can summarize a dynamic update such as this in the following way: a keyed READ request with Set Current Position, followed by a static REWRITE request.

Processing Duplicate Keys

In the preceding examples we used a standard ISAM file whose records were keyed by unique customer surnames. As the file grows, however, a problem may arise when you get two customers who have the same surname. To solve this type of problem, you can ask for duplicate keys when you create the file to differentiate between those customers. You must specify the duplicate key parameter at file creation or you will not be able to have duplicate keys in the main index. You can have duplicate keys in a subindex if you specify this parameter when you issue the DEFINE SUBINDEX command. (See Chapter 2 for more information on file creation parameters.)

To insert duplicate keys, you must specify that a key is a duplicate of an existing key when you write it into the file. The INFOS II system differentiates a duplicate key from an existing key by its unique *occurrence number*. There is no limit to how many duplicate keys you can have in an index.

The INFOS II system automatically assigns an occurrence number to each key when you create it. It gives occurrence number 1 to the first key you create and assigns subsequent numbers sequentially. The occurrence numbers of the keys in Figure 4-1 would look like this if you entered them in the order shown:

Key	Occurrence Number
BROWN	1
MAY	2
ABLE	3
ADAMS	4
ZAPPA	5
BAKER	6

After assigning occurrence numbers, the INFOS II system sorts the keys by their binary values. In our example, the INFOS II system sorts the index as follows:

Key	Occurrence Number
ABLE	3
ADAMS	4
BAKER	6
BROWN	1
MAY	2
ZAPPA	5

The next occurrence number that the system will assign is number 7. If the key that receives that number has a name that duplicates an existing key, the INFOS II system will show you the duplicate key's occurrence number. The system returns the occurrence number in the system returned status information, as described in Chapter 3.

For example, suppose you want to create another ADAMS key in the sample ISAM file we've been using. Assume that the file is open and the current position is on BROWN. To write another ADAMS key, you will code the equivalent of:

```
WRITE:  FILENAME = BILLING
        DUPLICATE KEY = ADAMS
        DATA AREA = AREA 1
        DATA RECORD LENGTH = 80
```

Note that you must specify that ADAMS is a duplicate key. Even if you have allowed duplicate keys when defining an index or subindex, the INFOS II system won't let you write a key that duplicates an existing key unless you specify that it is a duplicate. Of course as you add keys and records to a file, you might not know if a key is a duplicate or not. In subindexes for which you allowed duplicate keys, always specify DUPLICATE KEY when you write a new key, so the system will be prepared to handle it if it is a duplicate.

The system returns the occurrence number of the duplicate key after you enter it. In this example, the second ADAMS key will have an occurrence number of 7. The list of keys now looks like this:

Key	Occurrence Number	Visible to User?
ABLE	3	NO
ADAMS	4	YES
ADAMS	7	YES
BAKER	6	NO
BROWN	1	NO
MAY	2	NO
ZAPPA	5	NO

Notice that the system enters the duplicate key in its proper sequential location. It distinguishes the duplicate from the original ADAMS key by its unique occurrence number.

Suppose you next decide to delete MAY from the file and add the new key CHASEN. The key sequence and occurrence numbers would look like the following:

Key	Occurrence Number	Visible to User?
ABLE	3	NO
ADAMS	4	YES
ADAMS	7	YES
BAKER	6	NO
BROWN	1	NO
CHASEN	8	NO
ZAPPA	5	NO

Note that the system does not reuse occurrence numbers of deleted keys. This is always true, even if the key with the highest occurrence number is deleted. The next key written receives the next sequential occurrence number.

NOTE: Do not attempt to assign occurrence numbers to duplicate or unique keys. Doing so is impossible because the system assigns the numbers.

Gaining Access to Duplicate Keys

To gain access to a duplicate key, you must include its occurrence number. For example, let's say you want to obtain the second ADAMS record to update the balance. Assume that the file is open and your current position is above the index. Code the equivalent of:

```
REWRITE: FILENAME = BILLING
         DUPLICATE KEY = ADAMS
         OCCURRENCE NUMBER = 7
         DATA AREA = AREA1
         DATA RECORD LENGTH = 80
```

With any processing request including READ, WRITE, REWRITE, DELETE, REINSTATE, RETRIEVE STATUS, RETRIEVE KEY, or RETRIEVE HIGH KEY, the INFOS II system automatically returns the key's occurrence number with the System Returned Status Information, if the key is a duplicate.

The INFOS II process does not return occurrence numbers of keys that do not have duplicates. If you physically delete one of two duplicate keys, the system will not return the remaining key's occurrence number when you gain access to it later. It will return this information only if at least two duplicate keys remain.

If a key is a duplicate, you can gain access to the original (primary) key by specifying an occurrence number of 0. The INFOS II system will find the first occurrence of that key and return it with the System Returned Status Information.

Note that having duplicate keys does not affect the procedure for sequentially processing the file. You can use the same procedure we described earlier, since the system stores all keys in sequential order, distinguishing duplicates by their unique occurrence numbers.

Defining Subindexes

Suppose you have an application that requires that many keys point to one record, such as an accounts payable file. You want to have access to purchase order records by vendor name (Vendor), purchase order number (PO), and purchase order date (Date). For this application, you have created a multilevel (DBAM) file and named it PAYABLES.

The main index of PAYABLES has three selector keys, as shown in Figure 4-2. Since the primary purpose of these keys is to select a lower level subindex, they are not linked to data records; they were written with the Suppress Database Access option.

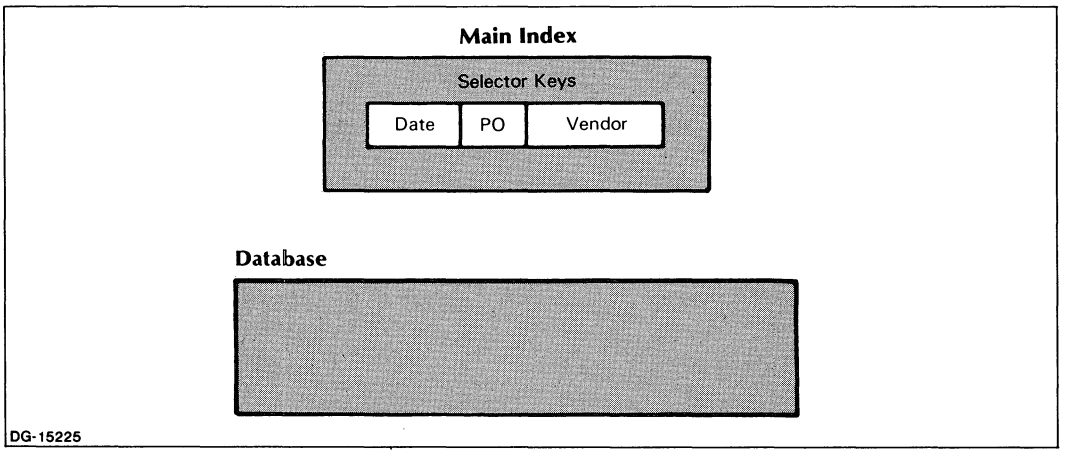


Figure 4-2. DBAM File PAYABLES After the Creation of the Main Index

In this section and the next section, we will define subindexes under the main index of this file and use the database inversion feature to link keys to an existing data record. For this example, we won't use partial records.

To define a subindex under a selector key, you use the DEFINE SUBINDEX command. Gain access to the selector key you want, and specify one or more subindex parameter values. Table 4-1 shows the subindex parameters, their standard values, and alternative values.

Table 4-1. Subindex Parameters

Parameter	Default Value	Alternative Values
Root Node Size	Page size minus 6 bytes	Large enough to contain three maximum-length index entries, but not greater than the default (page size minus 6 bytes).
Maximum Key Length	255 bytes	1 to 254 bytes
Partial Record Length	0 bytes	1 to 255 bytes
Allow Subindexes	Yes	No
Allow Duplicate Keys	No	Yes

As you know from Chapter 2, a default DBAM file allows one level of subindexes below the main index. You can, however, specify as many as 31 levels of subindexes below the main index if you don't choose the default. You can define lower level subindexes under any subindex you create in a nonstandard DBAM file, up to the number of levels you specified at file creation. You can also prohibit the creation of lower level subindexes under any subindex, by making the Allow Subindexes parameter value equal to No.

Our example file, PAYABLES, is a DBAM file with two index levels, the default. Therefore, you can create only one level of subindexes below the main index. This rule applies regardless of the Allow Subindexes parameter value you specify for any of the level 1 subindexes.

Let's define the Date subindex first. We'll assume that the keys are 6-digit numbers in the form yymmdd. We'll also allow duplicate keys since it's likely that you'll write several purchase orders (POs) on any given workday. Assuming that PAYABLES is open and that your current position is above the index, code the equivalent of:

```
DEFINE SUBINDEX: FILENAME = PAYABLES
                 KEY = Date
                 MAXIMUM KEY LENGTH = 6
                 ALLOW DUPLICATE KEYS = YES
```

At the successful completion of this DEFINE SUBINDEX operation, the PAYABLES file will look like Figure 4-3.

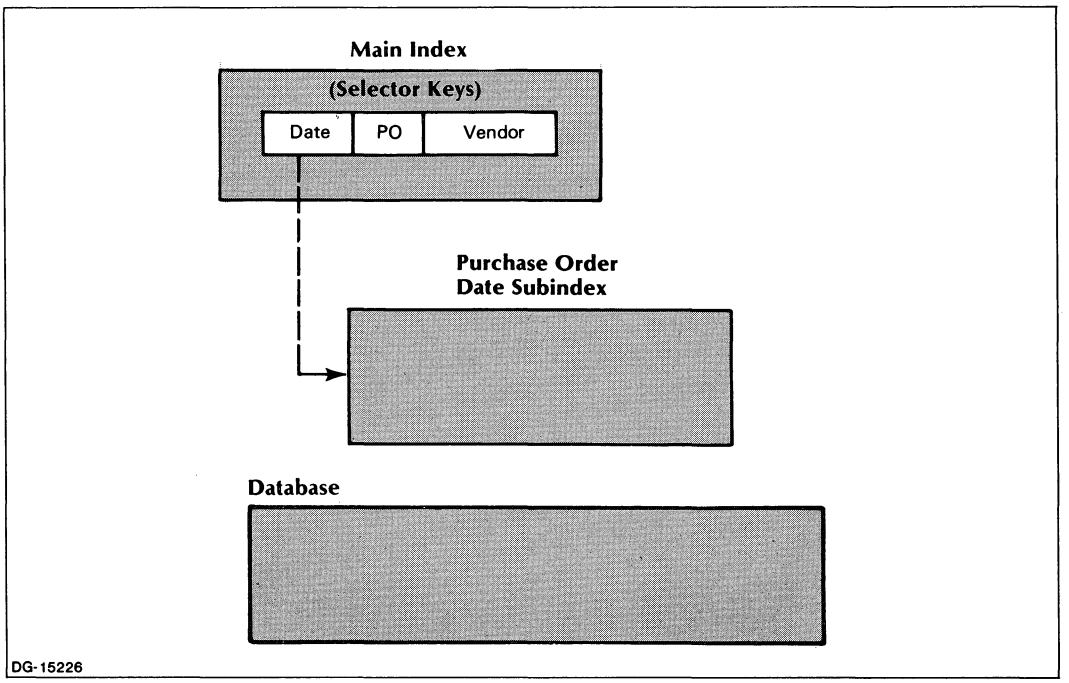


Figure 4-3. DBAM File PAYABLES with Date Subindex

For convenience in illustrating certain concepts described later, let's assume that you've written one Date key for each business day in 1983, using the Suppress Database Access option. You have also defined the PO and Vendor subindexes and have written the first purchase order record to the database through the PO subindex. The PAYABLES file now looks like Figure 4-4.

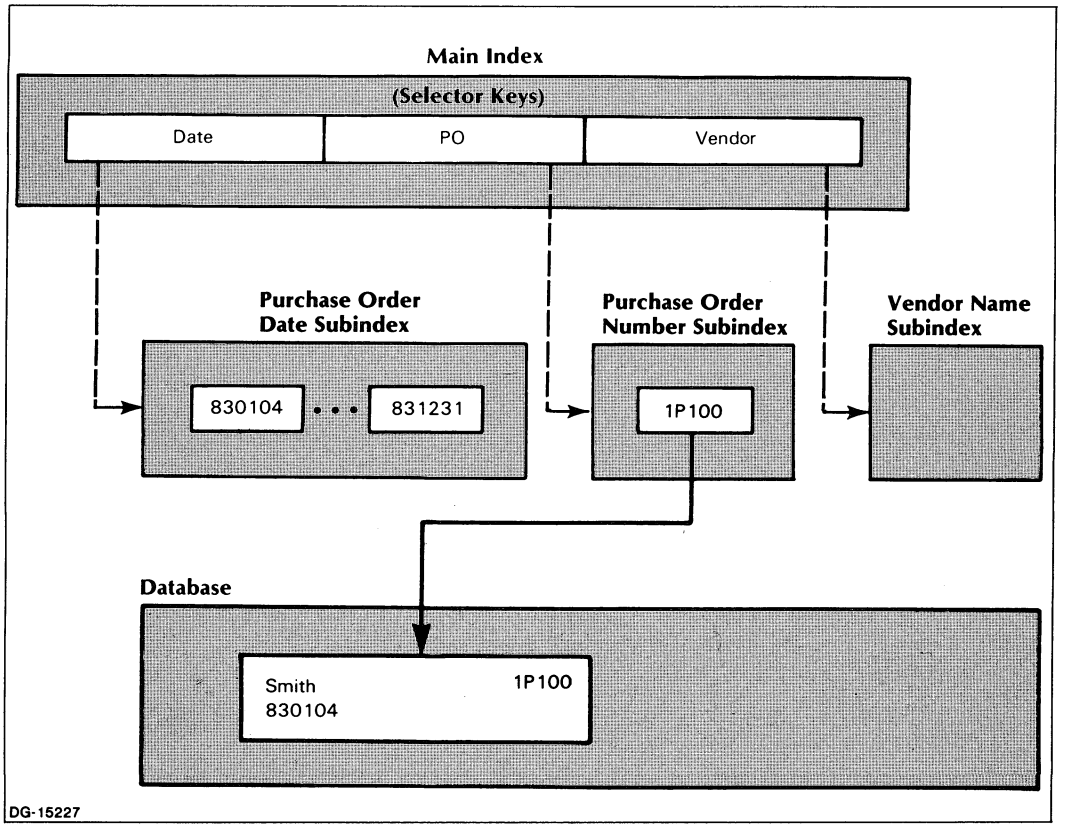


Figure 4-4. DBAM File PAYABLES with Date, Purchase Order Number, and Vendor Name Subindexes

Linking Keys to Existing Data Records (Database Inversion)

Linking a key to an existing data record is a two-step operation, similar to a dynamic update. The first step is a READ; the second is either a REWRITE or a WRITE. Use a REWRITE to link an existing key to an existing data record, and a WRITE to link a new key to an existing data record. You will use inversion with WRITE and REWRITE commands to perform the linking operation.

In the preceding section, we assumed that you used the WRITE command with suppress database access to insert keys into the Date subindex without data records, as shown in Figure 4-4. Now we can link the existing unattached keys to existing data records. You can also link a new key to an existing data record, which is why we created a Vendor subindex and didn't write anything in it. In this section we'll link the first key in the Date subindex (830104) to the existing data record for Smith. We'll also write a key to the Vendor subindex, linking it to the same data record.

Data Record Feedback

Every time you retrieve a record with the READ command, the INFOS II system returns information to your program called *data record feedback*. This information contains the logical address of the record in a 32-bit internal format, binary integer. You must supply this data record feedback for the REWRITE or WRITE inverted operation. With inversion you will perform the linking operation.

Remember that the feedback information can change if you rebuild your INFOS II file. Check to be sure that you have the most up-to-date address information before performing your inversion procedure.

The Inversion Procedure

Let's assume you've set up a special 4-byte area in your program, called AREA2, to which you'll move the feedback information returned on the READ. First we will link the key 830104 in the Date subindex to the data record shown in Figure 4-4. Assuming the file is open and your current position is on the key PO in the main index, code the equivalent of:

```
READ: FILENAME = PAYABLES
      DOWN AND FORWARD
      FEEDBACK = AREA2
      SET CURRENT POSITION
      SUPPRESS DATABASE ACCESS
```

You are now ready to do the REWRITE with inversion. Code the equivalent of:

```
REWRITE INVERTED: FILENAME = PAYABLES
                  KEY = Date, KEY = 830104
                  FEEDBACK = AREA2
                  SUPPRESS DATABASE ACCESS
```

On a REWRITE, the INFOS II system expects you to alter the contents of the data record. When inverting, however, your primary concern is with linking and not with updating. You should always specify Suppress Database Access when you do not intend to update the data record during a linking operation. If you do not specify Suppress Database Access, you must specify a data area and the length of the updated record; otherwise, you'll get an error.

Now you are ready to issue an inverted WRITE command. The data record feedback was placed in AREA2 by the previous READ command. Code the equivalent of:

```
WRITE INVERTED: FILENAME = PAYABLES
                KEY = Vendor, KEY = Smith
                FEEDBACK = AREA2
                SUPPRESS DATABASE ACCESS
```

On a WRITE operation, you normally write a new record to the database and a key to the index. However, when using database inversion, you are primarily concerned with writing a new key to the index and linking it to an existing data record. If you do not want to modify the contents of the data record, specify Suppress Database Access. If you don't suppress access, you must specify a data area and the length of the updated record; otherwise, you'll get an error.

Note that the other positioning options described in Chapter 3 are available to you on the READ, WRITE, and REWRITE commands you use to link keys to existing data records. The PAYABLES file will look like Figure 4-5 after you link the keys 830104 and Smith to the data record.

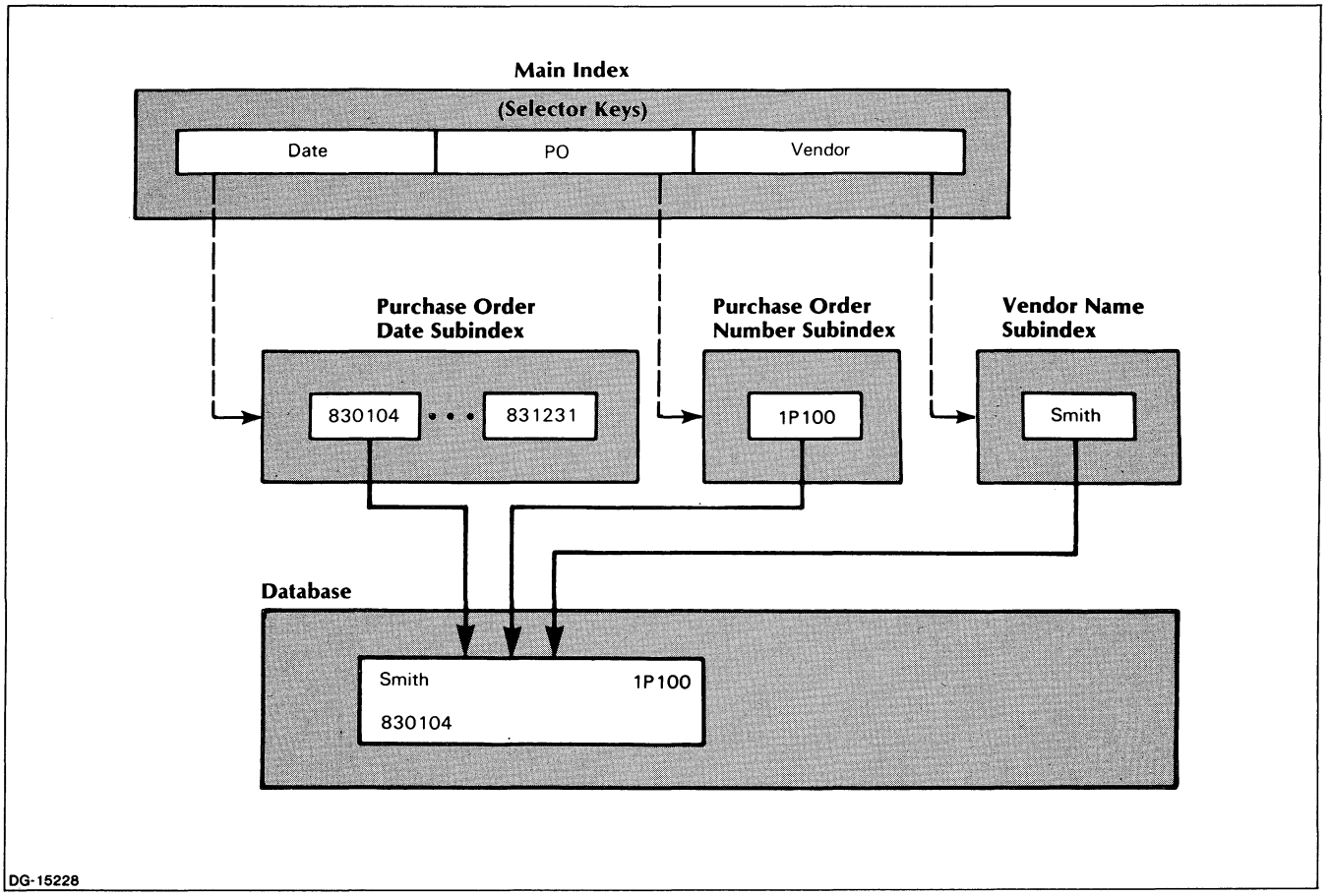


Figure 4-5. DBAM File PAYABLES with Inversion

Processing with Generic Search Keys

As we mentioned in Chapter 1, you can use generic keys to locate records when you don't know the precise names of the keys associated with them. When you use a generic key, the INFOS II system finds the first key in the main index (or current subindex) that exactly matches the generic key, up to the length of the generic key.

Using Figure 4-5, let's suppose you want to read the first purchase order written in May, 1983. Assuming the file is open and your current position is above the index, code the equivalent of:

```
READ: FILENAME = PAYABLES
      KEY = Date, GENERIC KEY = 8305
      DATA AREA = AREA1
```

The system then returns the first record associated with a key that begins with 8305. For instance, if the subindex contains the keys 830504 and 830501, the system will return the record associated with the key 830501. Note, however, that if there are no purchase orders written in May, and the first thereafter is dated June 1 (830601), the INFOS II system will return an error. In this case, no key in the index has four initial characters that match the generic key (8305).

Processing with Approximate Search Keys

Approximate keys are similar to generic keys. With approximate keys, however, the INFOS II system locates the first key in the main index (or the current subindex) that is equal to or greater than the approximate key, up to the length of the approximate key.

Using Figure 4-5, suppose you want to read the first purchase order you wrote in the period beginning August, 1983. Assuming that the file is open and your current position is above the index, code the equivalent of:

```
READ: FILENAME = PAYABLES
      KEY = Date, APPROXIMATE KEY = 8308
      DATA AREA = AREA1
```

If the search key 8308 doesn't match any key in the Date subindex, the system will return the record associated with the first key greater than 8308. For example, if no POs were written in August or September, the returned record could be for the first of October (831001). If, on the other hand, you had written purchase orders on every day in August, the system would return you the record associated with the key 830801. Finally, suppose you had accidentally typed in 8408 for the approximate key instead of 8308. In this case, no keys would match and you would receive the error *(IOKPE) KEYED POSITIONING ERROR.*

Linking Subindexes

As we said in Chapter 1, linking subindexes allows you to share subindexes among different keys in the same index file. You do this by linking a subindex already defined under a key (the *source key*) to a key in another subindex (the *destination key*). See the description of the LINK SUBINDEX command in Chapter 3 for more information about this process.

A common problem that you can solve by linking subindexes is when two or more groups of keys in a multilevel index need access to the information in one particular subindex. For example, let's use a file with these two related groups: Payroll and Personnel. The Payroll Department uses employee numbers as keys for the employee records and the Personnel Department uses employee names as keys for these records. In the Payroll Department, each employee number has a subindex that contains keys that point to that employee's deduction records. The Personnel Department, however, also needs access to this information.

Figure 4-6 illustrates such a file, named EMPLOYEES. But in this arrangement, Personnel cannot access the deduction records. We could solve this problem by defining another subindex under each name in the Employee Name subindex to hold deduction records. But this would duplicate information and waste a great deal of space and time. Instead, let's link the keys in the Employee Name subindex to the Deduction subindexes so that both Payroll and Personnel can share the deduction records.

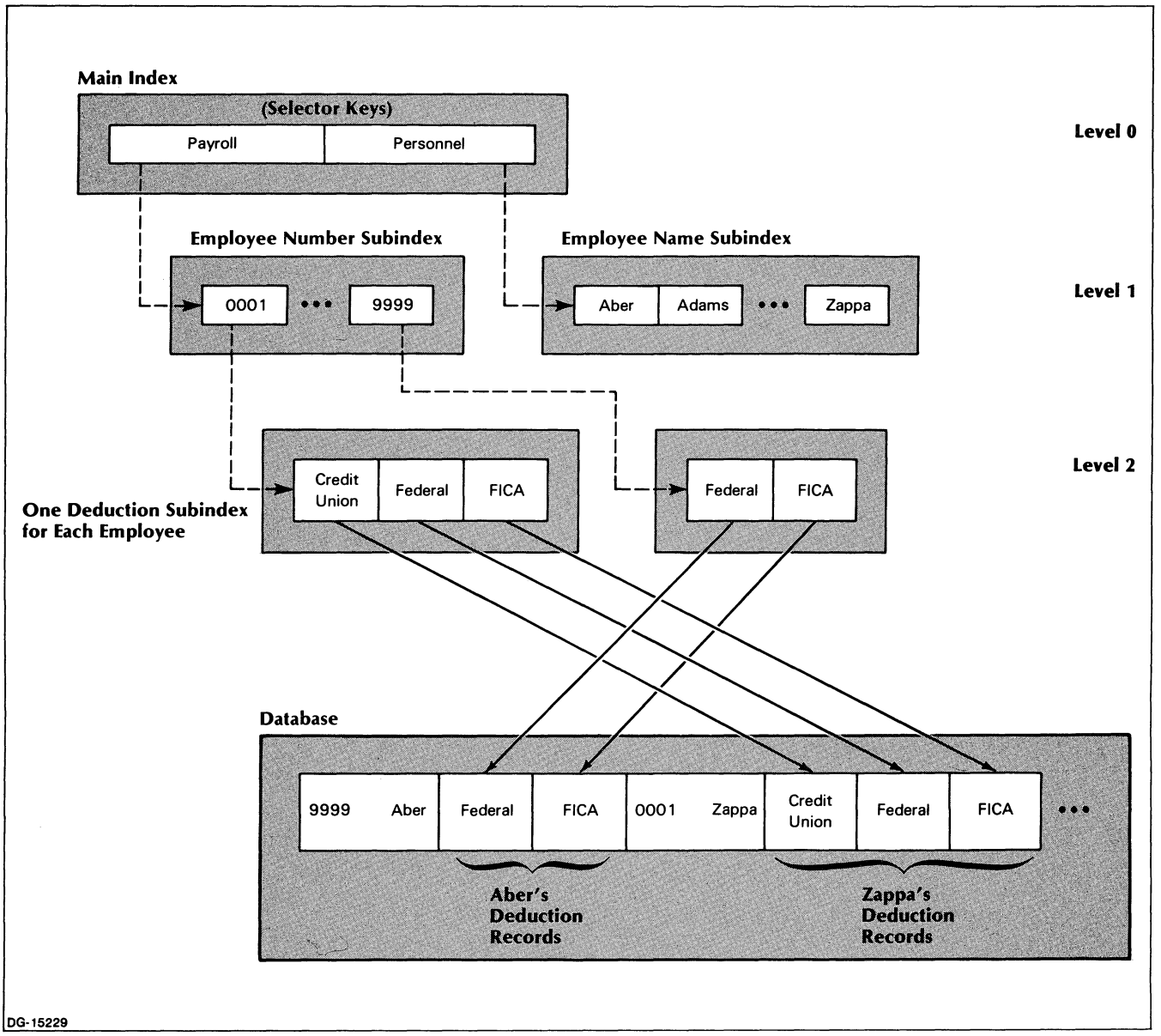


Figure 4-6. DBAM File EMPLOYEES

Bear in mind that when you issue the LINK SUBINDEX command to link a key to an existing subindex, the key cannot already be linked to a subindex.

The Linking Procedure

The following five-step procedure shows you how to enter a new employee record, Sanders (employee number 4123), into the database. Both Payroll and Personnel will be able to access deduction records for Sanders when you are finished.

1. Write Sanders' employee record using the following code:

```
WRITE:  FILENAME = EMPLOYEES
        KEY = Payroll, KEY = 4123
        FEEDBACK = AREA2
        DATA AREA = AREA1
        DATA RECORD LENGTH = 80
        SET CURRENT POSITION
```

   The resulting file will look like Figure 4-7.

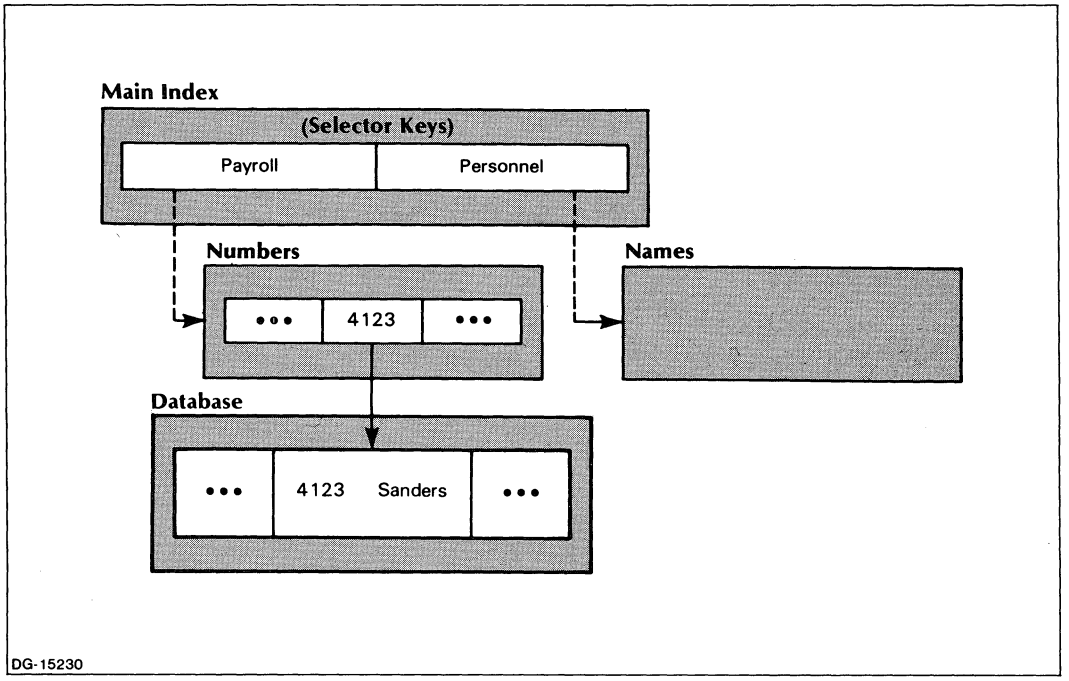


Figure 4-7. EMPLOYEES File – I

2. Define a subindex under Sanders' employee number to hold deduction keys. Follow this code:

```
DEFINE SUBINDEX: FILENAME = EMPLOYEES
                 STATIC
                 MAXIMUM KEY LENGTH = 12
                 NO SUBINDEXING
```

The file will then look like Figure 4-8.

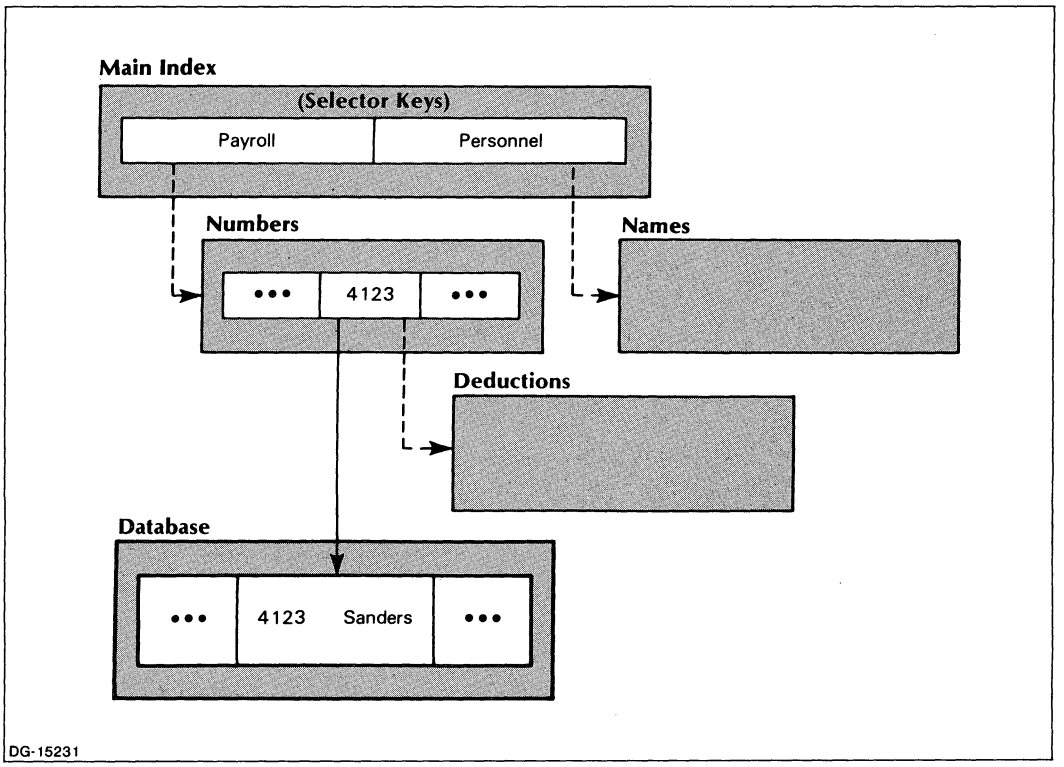


Figure 4-8. EMPLOYEES File – II

3. Use the WRITE command with inversion to enter the Sanders key into the Employee Name subindex and link it to the data record that is already linked to the key 4123. The previous WRITE placed the data record feedback into AREA2. If you code the equivalent of the following, your file will then look like Figure 4-9.

```
WRITE INVERTED: FILENAME = EMPLOYEES
                KEY = Personnel, KEY = Sanders
                FEEDBACK = AREA2
                SUPPRESS DATABASE ACCESS
```

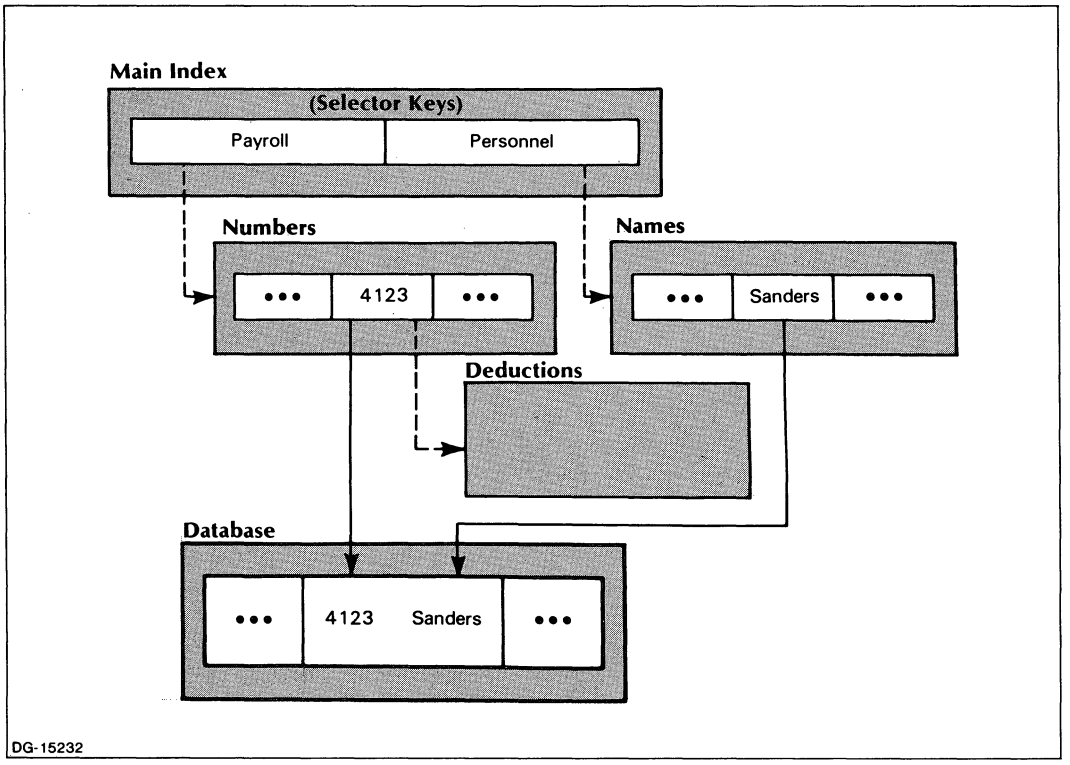


Figure 4-9. EMPLOYEES File – III

4. Now you can link the Sanders key with the subindex defined under the key 4123. To make this link, use the key 4123 as the source key and Sanders as the destination key. (See the LINK SUBINDEX command in Chapter 3 for a description of source and destination keys.) The file will look like Figure 4-10 after you code the following:

```
LINK SUBINDEX:FILENAME = EMPLOYEES
              SOURCE KEY = Payroll, KEY = 4123
              DESTINATION KEY = Personnel, KEY = Sanders
```

For this LINK SUBINDEX command, you can use keyed, relative, or a combination of keyed and relative access.

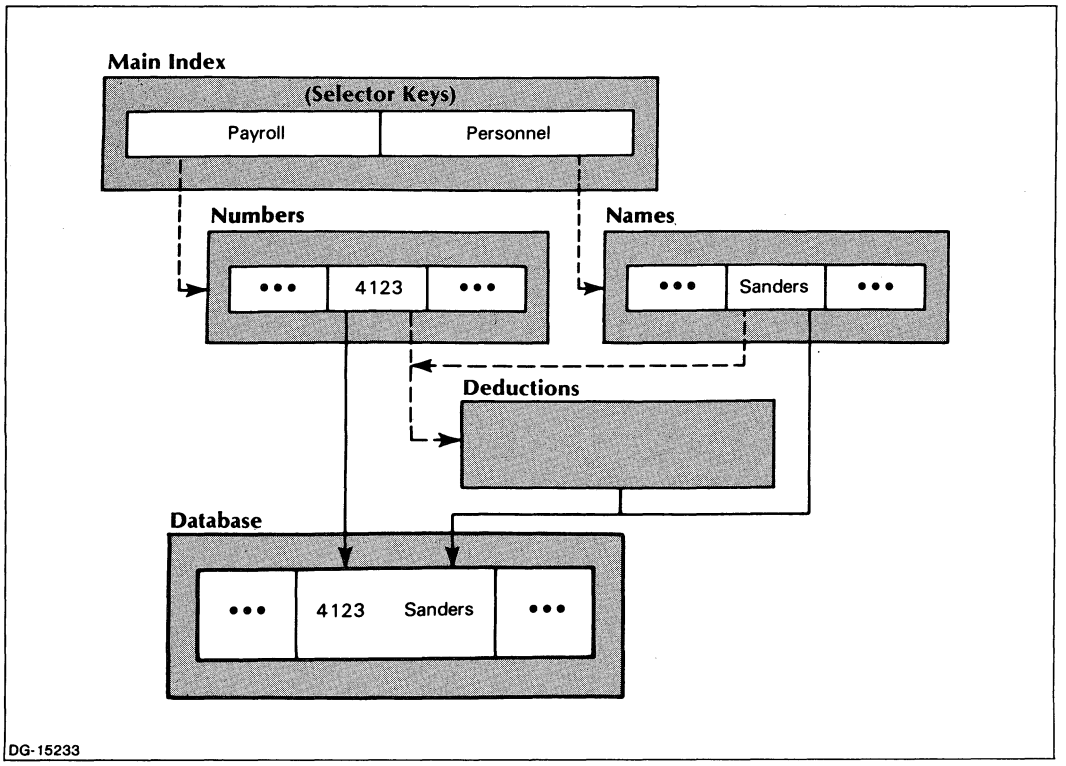


Figure 4-10. EMPLOYEES File – IV

5. Finally, you will enter the deduction records themselves. Code the equivalent of the following:

```
WRITE: FILENAME = EMPLOYEES
       KEY = Payroll, KEY=4123, KEY = FICA
       DATA AREA = AREA1
       DATA RECORD LENGTH = 80
```

You could also use the keypath Personnel,Sanders,FICA to write this same record. You will then proceed to write the remaining deduction records, substituting Federal and Credit Union for FICA.

Figure 4-11 shows the EMPLOYEES file after the deduction records have been written. Payroll and Personnel both share the Deduction Records Subindex.

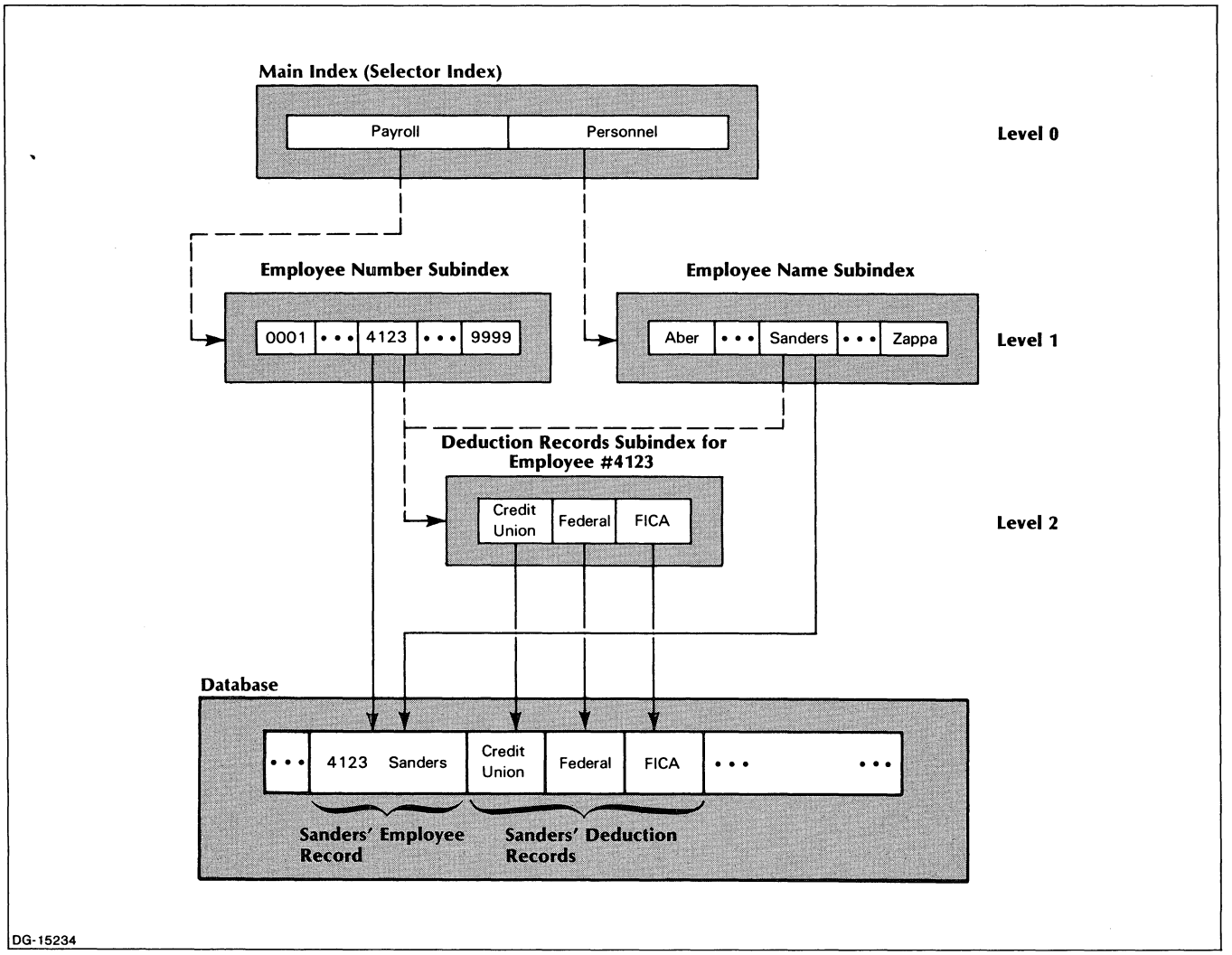


Figure 4-11. EMPLOYEES File with Deduction Records Subindex

Creating a New Index for an Existing Database (File Inversion)

In the preceding section, we solved one type of database access problem for the Personnel and Payroll departments — they needed to gain access to the same employee records by using different types of information as keys. We linked the Employee Name subindex to the Employee Deduction Records subindex so that each department could access deduction records in the Employees file through the same subindex. We can solve other accessing problems with file inversion. We outline this procedure in the following sections.

Let's assume that you want to give each department manager access to the employee records of his or her department. For example, let's assume that the Personnel Department Manager wants access to employee records primarily by surname and alternatively by employee number. The manager also wants to write job description records, keyed by job code number, for the department positions. You can allow this, without duplicating the existing employee records, by creating a new index linked to the same database through file inversion.

In this example, we will use the EMPLOYEES file in Figure 4-11. To define a separate DBAM index for the Personnel Department Manager, code the equivalent of:

```
OPEN: CREATE DBAM PERSONNEL
      INVERT
      DATABASE = EMPLOYEES.DB
```

You must specify that you are inverting and give an existing database name. All the options for an index file that we described in Chapter 2 are available to you when you create an index with inversion. Moreover, each new index can be a standard or a nonstandard ISAM or DBAM index. You can define one subindex level below the main index of PERSONNEL, the index we just defined. But PERSONNEL's main index cannot have duplicate keys or partial records.

Assuming you've written the three selector keys — Names, Employee Nos., and Codes — to the main index of PERSONNEL, and defined a subindex under each, the EMPLOYEES database should now look like Figure 4-12.

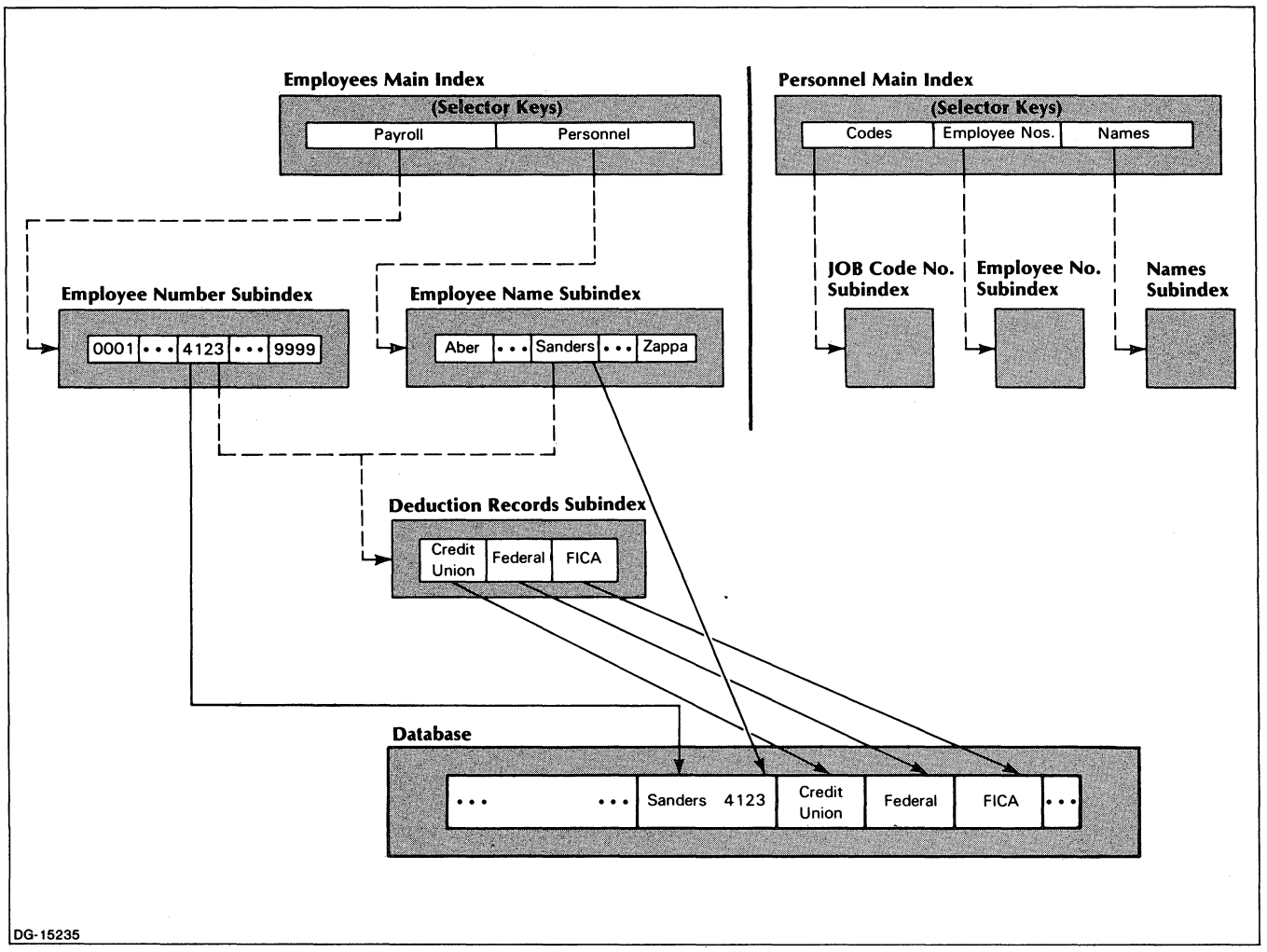


Figure 4-12. EMPLOYEES File with Two Indexes After Inversion

Writing to the New Index

Now you want to write the department's employee names to the Names subindex. Essentially, you'll do an inverted Write request, as we described in an earlier section. The one major difference is that you'll do a Read request in one file (EMPLOYEES) and a Write request in another (PERSONNEL).

To do this type of WRITE request, you must be sure both the original file, EMPLOYEES, and the newly created file, PERSONNEL, are open. For example, assume that employee Sanders works in the Personnel Department. To create a Sanders key, you will code the equivalent of:

```
READ: FILENAME = EMPLOYEES
      KEY = Personnel, KEY = Sanders
      DATA AREA = AREA1
      FEEDBACK = AREA2

WRITE INVERTED: FILENAME = PERSONNEL
                KEY = Names, KEY = Sanders
                FEEDBACK = AREA2
                SUPPRESS DATABASE ACCESS
```

Then, after writing all the department's employee names to the Names subindex of PERSONNEL, you can close the EMPLOYEES file and issue an inverted Write command to PERSONNEL's Employee Nos. subindex. For example, to enter Sanders' employee number, code the equivalent of:

```
READ: FILENAME = PERSONNEL
      KEY = Names, KEY = Sanders
      DATA AREA = AREA1
      FEEDBACK = AREA2

WRITE INVERTED: FILENAME = PERSONNEL
                KEY = Employee Nos., KEY = 4123
                FEEDBACK = AREA2
                SUPPRESS DATABASE ACCESS
```

Now the EMPLOYEES database looks like Figure 4-13.

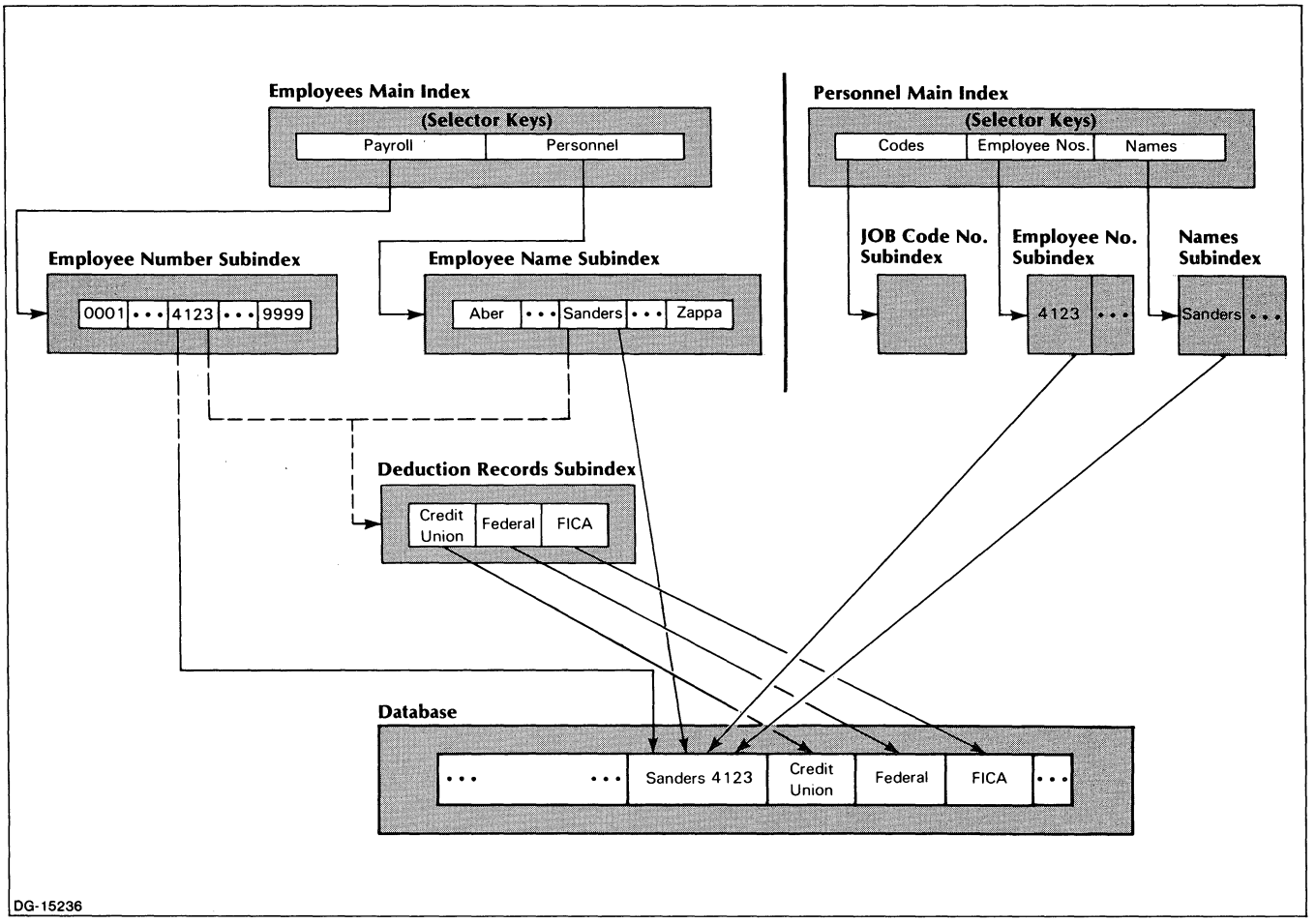


Figure 4-13. EMPLOYEES File With Subindex Keys in the Second Index

Of course, your processing problems will differ somewhat from those we have presented in this chapter. These examples should, however, serve as a useful guide as you code solutions to your own processing situations.

End of Chapter

Chapter 5
The INFOS II Utility Programs

In this chapter and the next we describe the utility programs, which help you create and maintain INFOS II files through the CLI. Be sure that you only run the latest revisions of the utilities with your new version of INFOS II. We assume that you are familiar with the information in Chapters 1 through 4 before you begin this chapter.

With the INFOS II utilities, you can gain access to files that you created previously through a language, or those that you have created with the ICREATE utility.

With INFOS II system utility programs, you can do everything that is possible with the INFOS II commands. You can create, modify, and delete files, and retrieve information about a file's structure. The utility programs also let you load, dump, and rename files, and retrieve statistics on a file's use.

The file inquire utility, INQUIRE, contains a full menu of its own commands and command modifiers that you use to examine and modify an INFOS II file. Its uses are quite extensive. For this reason, we have described INQUIRE by itself in the following chapter.

You use the INFOS II CHECKPOINT utility during backup and recovery procedures after a system failure. For this reason, we will describe this utility in Chapter 7, "INFOS II Backup and Recovery Options."

Table 5-1 summarizes the INFOS II utility programs and their purposes.

Table 5-1. INFOS II Utility Programs

Generic Name	Name	Description
File Create	ICREATE	Creates an INFOS II file at an interactive terminal. You can specify any of the parameters described in Chapter 2, or use the appropriate switches to create a standard INFOS II file.
File Information Retrieval	IFILE	Retrieves information about an index, a database, and the associated volumes of an INFOS II file. The system will display the information on an interactive terminal, or you can direct the output to the generic LIST file or another disk file.
File Rename	IRENAME	Changes the name of any INFOS II index or database.
File Dump	IDUMP	Creates a backup copy of any INFOS II file on magnetic tape or another disk. You can load the backup copy via the AOS load facility.
File Dump	DDUMP	Provides incremental or archival backup copies of any INFOS II file.
File Load	DLOAD	Loads the backup copy of an INFOS II file created by DDUMP.

(continues)

Table 5-1. INFOS II Utility Programs

Generic Name	Name	Description
File Delete	IDELETE	Deletes any INFOS II file. It deletes the database, all associated indexes, and all volumes of the file.
File Verification	IVERIFY	Ensures that your file is structurally valid. It also retrieves file use statistics for any INFOS II database and all its indexes.
File Inquiry	INQUIRE	Opens an INFOS II file for you to examine or modify. You can use relative or keyed access to move through the file to read, write, delete, etc. Chapter 6 describes this utility in detail.
Index Load	IXLOAD	Generates an entire INFOS II main index or subindex from a sequential file.
File Update	CHECKPOINT	Updates a master file with differential file data. (See Chapter 7 for more information.)
File Structure Evaluation	INDEXCALC	Predicts the structure of an INFOS II file when you specify certain parameter values.

(concluded)

How to Use the INFOS II Utility Programs

You invoke the utility programs through CLI macros that are supplied with the INFOS II system. You can issue these macros at an interactive terminal or in a batch stream. Each call must contain the name of a utility and any optional switches appropriate to that utility. Each utility also expects to see an index or database name when you invoke it. If you don't include a name on the command line, the utility will prompt you for it. If, when prompted, you press NEW LINE without entering a name, the system will return you to the CLI.

The following is a sample utility command line:

```
IDELETE TESTFILE.DB
```

In this example, the IDELETE utility will delete the database file named TESTFILE.DB, all its associated index files, and all their volumes.

How to Respond to Utility Prompts

Certain utilities, such as ICREATE and INQUIRE, ask questions about the values you want for the features they offer. If you press the NEW LINE key in response, you indicate that you want the standard (default) value for these features. The default value is usually included in square brackets, [ ], in the prompt. If you don't want the default, you must enter the value you want.

In some cases, the standard value is a numeric value; in others it's the letter Y or N. For example, ICREATE has the following prompt:

```
PAGE SIZE (BYTES) [2048]:
```

If you want the 2048-byte page size, respond by pressing the NEW LINE key (you could also enter 2048 and press the NEW LINE key). If you want the larger page size, however, you must enter 4096 followed by the NEW LINE key. Another ICREATE prompt is:

```
ENABLE SPACE MANAGEMENT (Y OR [N]):
```

If you don't want space management, press the NEW LINE key (or press N and then the NEW LINE key). If you do want space management, however, you must type Y and press NEW LINE.

Audit Trailfiles

An *audit trailfile* is a record of your interaction with a utility for a given session. It contains all the prompts the utility issued and your responses to them. On subsequent sessions of the utility, you can use the contents of the trailfile as input to the utility, rather than having to enter all your responses interactively.

To create a trailfile for a given session with the utility, use the global switch /T=trailfile. Do not enter any arguments, such as a database name, in the command line. The utility will prompt you for them. To use the trailfile as input for a subsequent session, use the global switch /B=trailfile.

For example, you can describe an INFOS II file and keep your interaction with ICREATE in a trailfile, while not actually creating the file. To do this, invoke the ICREATE utility with these switches:

```
ICREATE/N/T=TRAIL
```

You will then have a trailfile that you can examine and edit. At some later time you can use TRAIL as input for the ICREATE utility. To create such a file later, invoke the ICREATE utility with the /B=trailfile switch:

```
ICREATE/B=TRAIL
```

INFOS II Utility Switches

The following global switches apply to most utilities. Some utilities have additional global switches and local switches specific to themselves. We describe the switches that are unique to a particular utility in the description of that utility.

Switch	Description
/L	Directs the output of the utility to the generic LIST file.
/L=filename	Directs the output of the utility to *filename*.
/N	Accepts all input, but does not actually execute the utility. Most often used with the /T switch to produce a batch input file to the utility.
/T=trailfile	Stores your interaction with the utility in *trailfile*.
/B=trailfile	Uses the contents of *trailfile* as input to the utility.

Most of the utility programs display a completion message when they terminate, before returning you to the CLI. Generally, the system displays this message on the terminal screen, but you can direct it to a list file if you wish.

ICREATE

Use the File Create utility, ICREATE, to create an INFOS II file or to create an inverted index to an existing file. With it you can define a system standard ISAM or DBAM file, or specify the parameter values you want for a nonstandard file. Chapter 2 describes the INFOS II file creation parameters, their default values, and alternative values. At the successful completion of ICREATE, you have a fully defined, empty file, which you can then begin to process.

ICREATE is easy to use. You can enter all the information required to create a file on a single CLI command line. Or, through interactive dialog, you can enter values for those parameters you want changed and use the system-assigned values for the rest of the parameters.

Note that INFOS II database and index files *must* have the same parent directory. You must not separate them, even by providing links in their place. You can, however, relocate INFOS II volumes and create links to them if you want to place them on faster devices for better performance.

ICREATE Syntax

ICREATE *[/N] [/T=trailfile] [/B=trailfile] [/S] [[filename] [/I] [/D]]*

The following list describes the local ICREATE switches. See the list of global switches for an explanation of the */N, /B,* and */T* switches.

Switch	Description
/S	Creates a standard DBAM file. All parameters except the index filename will have default values. ICREATE will prompt you for the index *filename.*
/S filename	Creates a default DBAM file, called *filename.*
/S filename/I	Creates a default ISAM file, called *filename.*
filename	Creates a DBAM file, called *filename,* and asks you for parameter values.
filename/D	Creates a DBAM file, called *filename,* and asks you for parameter values.
filename/I	Creates an ISAM file, called *filename,* and asks you for parameter values.

ICREATE Dialog

The ICREATE utility prompts you for parameters. Depending on your responses to these prompts, ICREATE prompts you for other parameters. Note that each prompt, except the filename prompt, includes the default value for that option in square brackets. Remember that if you respond to a prompt by pressing the NEW LINE key, the system will use the default value shown for that option. There is no default value for the filename.

This section discusses the ICREATE dialog in detail.

ICREATE Prompt	Your Action
NAME OF FILE TO BE CREATED:	Enter the name of the index file you want to create. This is the name you will use to open the file for processing. ICREATE checks to be sure that the name you specify is not in use in the directory where the file is being created. If it is, you'll get an error and ICREATE will repeat the filename prompt. If you press NEW LINE without specifying a name, ICREATE will return you to the CLI.
ACCESS METHOD (I=ISAM, D=DBAM) [D]:	To create an ISAM file, enter I. The default is DBAM. If you want to create a DBAM file, enter D or just press NEW LINE. If you choose DBAM, ICREATE will ask about subindex levels. It will skip this question if you choose ISAM.
****** DEFINE INDEX FILE ***** MAXIMUM NUMBER OF INDEX LEVELS [2]:*	The INFOS II system permits up to 32 index levels for a DBAM file. If you respond by pressing NEW LINE, ICREATE will restrict you to two index levels — 0 and 1. You can subsequently define one subindex level below the main index. Remember that when you respond to ICREATE prompts you initially create only the level 0 index. You'll create all lower level subindexes dynamically with the DEFINE SUBINDEX command.

PAGE SIZE (BYTES) [2048]:	You can specify a 4096-byte page for the index, but we recommend that you read Chapter 10 of this manual before choosing this size over the default.
PARTIAL RECORD LENGTH [0]:	You can specify a partial record length from 0 to 255 bytes long. If you enter 0 or press NEW LINE, the INFOS II system will not allow partial records in the main index. However, your response to this prompt does not affect your ability to create partial records in lower level subindexes.
ROOT NODE SIZE [2042]:	Chapter 10 describes root nodes. Unless you are familiar with them, we suggest that you always use the default root node size. If you do specify an alternative root node size, it must be at least large enough to contain three maximum length index entries. However, it cannot be greater than 6 bytes less than your page size. The system allocates space equivalent to the size you specify (either explicitly or by default) at file creation. Thus, the root node is limited to that size. See Chapter 10 for more information.
MAXIMUM KEY LENGTH [255]:	The default length of 255 bytes is the maximum that the system allows. Responding with a NEW LINE allows you to enter keys from 1 to 255 bytes long into the main index. If you want to make the upper limit smaller than 255, then enter the largest number of bytes you want to allow.
ALLOW DUPLICATE KEYS IN THIS INDEX? (Y OR [N]):	If you want to be able to write duplicate keys in the main index, respond with Y. If not, enter N or press NEW LINE.
ENABLE SPACE MANAGEMENT? (Y or [N]):	If you want to enable space management for the index, enter Y. Otherwise, enter N or press NEW LINE.
ENABLE KEY COMPRESSION? (Y OR [N]):	If you want key compression for the index, enter Y. If not, enter N or press NEW LINE.
OPTIMIZE RECORD DISTRIBUTION? (Y OR [N]):	See Chapter 10 for information on optimized record distribution. If you want optimized record distribution, enter Y. The system will then respond with this prompt:
ROOT NODE MERIT FACTOR: [1]:	Select the merit factor you want, or press NEW LINE to accept the default. Chapter 10 contains more information about the root node merit factor.

Next, the ICREATE utility will prompt you for information about the index volumes.

****** DEFINE INDEX VOLUME(S) ******
NUMBER OF VOLUMES TO DEFINE [1]:

If you aren't going to use differential file mode, you can specify up to 32 volumes for the index. If you process in differential file mode, you can specify a maximum of 16 volumes; the INFOS II system will subsequently create the corresponding 16 differential volumes. (See Chapter 7 for more information on differential file mode.)

The system repeats the following series of prompts for each volume you specified in response to the last prompt:

VOLUME 1 NAME [VOL01]:

If you don't want the default filename, you can use any legal AOS filename, except DVLnn, where nn is the volume number. If you did name any of your volumes DVLnn and processed your file in differential file mode, you would receive a file consistency error and could lose your file.

SPECIFY MAXIMUM SIZE? (Y OR [N]):

The N or NEW LINE response causes the system to define the largest volume it can. The system can use up to 1,048,576 disk blocks for an index with a 2048-byte page size. It can use twice as many disk blocks for an index with a 4096-byte page size.

If you respond to this prompt with a Y, ICREATE displays the following prompt:

SIZE (DISK BLOCKS):

Enter the number of disk blocks you want the system to allocate for the index on the volume being defined. If the index uses the 2048-byte page size, you can specify up to 1,048,576 blocks. For the 4096-byte page size, you can specify up to 2,097,152 blocks. Remember that the INFOS II system allocates space as it is needed, so you may not get all the blocks you request. They may have been previously allocated by the operating system.

Note that if you make volume size and element size equal, you will get a contiguous volume (if that much contiguous space is available on the disk). The largest contiguous volume you can specify is 65,535 blocks, regardless of your index page size.

SPECIFY FILE ELEMENT SIZE? (Y OR [N]):

The standard element size is 32 blocks. You can specify an element size up to the size of a logical disk (LD), or 65,532, whichever is less. To do so, respond Y to this prompt and the system will display the next prompt.

SIZE IN DISK BLOCKS (MULTIPLE OF n):

The character *n* will appear as 4 or 8 depending on your page size. For the smaller page size, specify a number of disk blocks that is a multiple of 4. For the larger page size, specify a multiple of 8.

VOLUME MERIT FACTOR: [1]:	This prompt appears only if you selected optimized record distribution. Select the volume merit factor you want for each volume. Do not assign volume merit factors in increasing order.

Next, the ICREATE utility prompts you for information about the database file.

****** DEFINE DATABASE FILE ***** DATABASE FILE NAME [indexfilename.DB]:*	By default, the database has the same name as the index with the extension .DB added. But you can give the database any name you want, as long as you follow AOS naming conventions. Do not specify a pathname to a database file; just use the filename. ICREATE will reject a pathname. The database file must be in the same directory as its associated index files.

The remainder of the database file prompts are the same as those for the index file. Respond to them in the same way you responded to the index file prompts.

```
PAGE SIZE (BYTES) [2048]:
ENABLE SPACE MANAGEMENT? (Y OR [N]):
ENABLE DATA RECORD COMPRESSION? (Y OR [N]):
OPTIMIZE RECORD DISTRIBUTION? (Y OR [N]):
****** DEFINE DATABASE VOLUME(S) ******
NUMBER OF VOLUMES TO DEFINE [1]:
VOLUME 1 NAME [VOL01]:
SPECIFY MAXIMUM SIZE? (Y OR [N]):
SIZE (DISK BLOCKS):
SPECIFY FILE ELEMENT SIZE? (Y OR [N]):
SIZE IN DISK BLOCKS (MULTIPLE OF n):
VOLUME MERIT FACTOR [1]:
```

Sample ICREATE Session with an ISAM File

Suppose you want to create an ISAM file with a maximum key length of 40 bytes. You want to be able to use duplicate keys, but you don't need partial records. The largest data record is 300 bytes long and you want a contiguous database of 40,000 blocks. The name of the file is P_STATS. You will have the following dialog with ICREATE at your terminal:

ICREATE Response	Your Action
	ICREATE ⤸
*******INFOS FILE CREATION mm/dd/yy hh/mm/ss *******	
NAME OF FILE TO BE CREATED:	P_STATS ⤸
ACCESS METHOD (I=ISAM,D=DBAM)[D]:	I ⤸
****** DEFINE INDEX FILE *******	
PAGE SIZE (BYTES) [2048]:	⤸
PARTIAL RECORD LENGTH [0]:	⤸
ROOT NODE SIZE [2042]:	⤸
MAXIMUM KEY LENGTH [255]:	40 ⤸
ALLOW DUPLICATE KEYS IN THIS INDEX? (Y OR [N]):	Y ⤸
ENABLE SPACE MANAGEMENT? (Y OR [N]):	⤸

```
ENABLE KEY COMPRESSION? (Y OR [N]):                    ↵
OPTIMIZE RECORD DISTRIBUTION? (Y OR [N]):              ↵
****** DEFINE INDEX VOLUME(S) ******
NUMBER OF VOLUMES TO DEFINE [1]:                       ↵
VOLUME 1 NAME [VOL01]:                                 ↵
    SPECIFY MAXIMUM SIZE (Y OR [N]):                   ↵
    SPECIFY FILE ELEMENT SIZE? (Y OR [N]):             ↵
****** DEFINE DATABASE FILE ******
DATABASE FILENAME [P_STATS.DB]:                        ↵
PAGE SIZE (BYTES) [2048]:                              4096 ↵
ENABLE SPACE MANAGEMENT? (Y OR [N]):                   ↵
ENABLE DATA RECORD COMPRESSION: (Y OR [N]):            ↵
OPTIMIZE RECORD DISTRIBUTION (Y OR [N]):               ↵
****** DEFINE DATABASE VOLUME(S) ******
NUMBER OF VOLUMES TO DEFINE [1]:                       ↵
VOLUME 1 NAME [VOL01]:                                 ↵
    SPECIFY MAXIMUM SIZE? (Y OR [N]):                  Y ↵
      SIZE (DISK BLOCKS):                              40000 ↵
    SPECIFY FILE ELEMENT SIZE? (Y OR [N]):             Y ↵
      SIZE IN DISK BLOCKS (MULTIPLE OF 8):             40000 ↵
```

Sample ICREATE Session with a DBAM File

Suppose you want to create a DBAM file with a main index that is a selector index. A root node of 512 bytes can contain all the keys of the main index. You will eventually define four levels of subindexes below the main index, some of which will also be selector subindexes. You want the INFOS II system to automatically place as many root nodes on a page as it can, so you'll need space management for the index. But you don't need partial records or duplicate keys.

The name of the index is RECEIVABLES and the name of the database is ACCOUNTS. Eventually, you'll define a second index for the database, which you'll name PAYABLES. You want a two-volume database; the first volume will be 65,532 contiguous blocks and the second can be any size; the system will only use it to hold overflow database records. You will have the following dialog with ICREATE.

ICREATE Response	Your Action
	ICREATE RECEIVABLES ↵
******INFOS FILE CREATION mm/dd/yy hh:mm:ss ******	
ACCESS METHOD(I=ISAM,D=DBAM)[D]:	↵
****** DEFINE INDEX FILE ******	
MAXIMUM NUMBER OF INDEX LEVELS [2]:	5 ↵
PAGE SIZE (BYTES) [2048]:	↵
PARTIAL RECORD LENGTH [0]:	↵

```
ROOT NODE SIZE [2042]:                          512 ↵
MAXIMUM KEY LENGTH [255]:                       6 ↵
ALLOW DUPLICATE KEYS IN THIS INDEX? (Y OR [N]): ↵
ENABLE SPACE MANAGEMENT? (Y OR [N]):            Y ↵
ENABLE KEY COMPRESSION? (Y OR [N]):             ↵
OPTIMIZE RECORD DISTRIBUTION? (Y OR [N]):       ↵
****** DEFINE INDEX VOLUME(S) ******
NUMBER OF VOLUMES TO DEFINE [1]:                ↵
VOLUME 1 NAME [VOL01]:                          ↵
    SPECIFY MAXIMUM SIZE? (Y OR [N]):           ↵
    SPECIFY FILE ELEMENT SIZE? (Y OR [N]):      ↵
****** DEFINE DATABASE FILE ******
DATABASE FILENAME [RECEIVABLES.DB]:             ACCOUNTS ↵
PAGE SIZE (BYTES) [2048]:                       ↵
ENABLE SPACE MANAGEMENT? (Y OR [N]):            ↵
ENABLE DATA RECORD COMPRESSION? (Y OR [N]):     ↵
OPTIMIZE RECORD DISTRIBUTION? (Y OR [N]):       ↵
****** DEFINE DATABASE VOLUME(S) ******
NUMBER OF VOLUMES TO DEFINE [1]:                2 ↵
VOLUME 1 NAME [VOL01]:                          ↵
    SPECIFY MAXIMUM SIZE? (Y OR [N]):           Y ↵
      SIZE (DISK BLOCKS):                       65532 ↵
    SPECIFY FILE ELEMENT SIZE? (Y OR [N]):      Y ↵
      SIZE IN DISK BLOCKS (MULTIPLE OF 4):      65532 ↵
VOLUME 2 NAME [VOL02]:                          ↵
SPECIFY MAXIMUM SIZE? (Y OR [N]):               ↵
    SPECIFY FILE ELEMENT SIZE? (Y OR [N]):      ↵
```

Creating a Second Index for a Database

In the preceding example, we created an index named RECEIVABLES and a database named ACCOUNTS. Now suppose you want to create a second index named PAYABLES for the same database. Invoke ICREATE with the following:

```
)ICREATE PAYABLES ↵
```

After you specify parameter values for PAYABLES, respond to the prompt for database filename as shown below:

```
****** DEFINE DATABASE FILE ******
DATABASE FILENAME [PAYABLES.DB]:        ACCOUNTS ↵
```

You now have two index files for the ACCOUNTS database.

IFILE

Use the file information retrieval utility, IFILE, for two purposes:

- To retrieve information about an index, a database, or any volume of either an index or a database.
- To select a recovery option for a database file. (We discuss how to do this in Chapter 7.)

IFILE returns information about INFOS II files in two modes: normal and verbose. The difference between them is the amount of information they return. In the default normal mode, IFILE returns information about the specified index and its volume(s), or about a database and its volume(s). If you use the /V switch for verbose mode, IFILE will return extra information. Specifically, if you use the /V switch and specify an index filename, IFILE will supply information about the index and its associated database. If you specify a database name, it will return information about the database and all its associated indexes.

You can run IFILE to retrieve information on an INFOS II file even if that file is open to INFOS II or another utility.

How to Invoke IFILE

The syntax to invoke IFILE for information retrieval follows. Please note:

- IFILE conducts no dialog.
- IFILE has only one prompt: if you do not include a filename on the command line, IFILE will ask you for one.

Syntax

IFILE *[/V] [/T=trailfile] [/L] [/L=filename] [filename] [filemode]*

Since the */V* switch specifies verbose mode, omitting this switch indicates normal mode. See the list of global switches earlier in this chapter for a description of the other switches.

We describe how to use IFILE to select a file processing mode in Chapter 7, so we will not discuss the *filemode* option here.

The following example shows an IFILE output in verbose mode.

```
) IFILE/V BOOKS ↵

FILE: BOOKS
FILE TYPE: INDEX FILE
DATABASE FILE:  BOOKS.DB
ACCESS METHOD: ISAM
PAGESIZE: 2048 BYTES
NUMBER OF VOLUMES: 1
NUMBER OF INDEX LEVELS: 1

        *** VOLUMES ***

        VOLUME NO. 1
        VOLUME NAME: VOL01
        VOLUME ELEMENT SIZE: 32 BLOCKS
        VOLUME MAXIMUM SIZE: 1048576 BLOCKS
        DISK BLOCKS USED: 12
```

```
*** DATABASE FILE ***

FILE: BOOKS.DB
FILE TYPE: DATABASE FILE
ACCES METHOD: ISAM
PAGE SIZE: 2048 BYTES
NUMBER OF VOLUMES: 1
NUMBER OF INDICES: 1
MINIMUM RECORD LENGTH: 45
MAXIMUM RECORD LENGTH: 54

        *** VOLUMES ***

        VOLUME NO. 1
        VOLUME NAME: VOL01
        VOLUME ELEMENT SIZE: 32 BLOCKS
        VOLUME MAXIMUM SIZE: 1048576 BLOCKS
        DISK BLOCKS USED: 12
```

If we had not asked for verbose output in the sample above, we would have received only the output for the index, and not the last two sections, which specify information about the database file.

IRENAME

Use the file rename utility, IRENAME, to change the name of any index or database. IRENAME checks the directory in which the present filename exists to ensure that a file with the new filename does not already exist there. If one does exist, IRENAME will prompt you for another filename. You can then respond by entering a new filename or by pressing NEW LINE, which returns you to the CLI.

If the file you want to rename is not in your current directory, you must specify a complete pathname to it. However, you need not specify the complete path to the new filename; specify only the new filename.

CAUTION: Do NOT use the CLI RENAME command to rename an INFOS II file. The INFOS II index and database contain each other's names. The IRENAME utility updates this internal information, whereas the CLI RENAME command does not.

How to Invoke IRENAME

There is normally no dialog with the IRENAME utility, because you can specify both the current and new filenames on the CLI command line. However, if you specify only one filename, IRENAME will assume that it's the present filename and prompt you for the new filename. If you don't specify any filenames on the command line, IRENAME will first ask for the present filename, and then for the new filename. There are no other prompts in IRENAME.

Syntax

IRENAME *[/N] [/L] [/L=filename] [/T=trailfile] [/B=trailfile] [present-filename] [new-filename]*

We explained these switches earlier in this chapter, in the list of global switches.

Examples

1. Renaming an Index

```
)IRENAME ↓
PRESENT FILENAME: TESTFILE ↓
NEW FILENAME: INFOS_FILE ↓
INDEX FILE NAMED TESTFILE RENAMED INFOS_FILE
```

2. Renaming a Database

```
)IRENAME ↓
PRESENT FILENAME: TESTFILE.DB ↓
NEW FILENAME: INFOS_FILE.DB ↓
DATABASE FILE TESTFILE.DB RENAMED INFOS_FILE.DB
```

IDUMP

The file dump utility, IDUMP, enables you to create a backup copy of an INFOS II file. You must have the privilege to bring up a son (subordinate process) in order to use IDUMP.

Files you dump via IDUMP can be loaded with the AOS Load facility. Normally, you'll dump to a magnetic tape so that you can safely store the file off-line. You can, however, dump to another disk file.

You can specify the names of the dump file and the INFOS II file on the CLI command line. If you do this, be sure to give the name of the dump file first, then the INFOS II filename. If you specify only one filename, IDUMP will assume it's the dump file and prompt you for the INFOS II filename. If you don't specify any filenames on the command line, IDUMP will prompt you for them.

IDUMP will always dump a database and all its associated indexes, regardless of whether you specify an index name or a database name.

The INFOS II DDUMP and DLOAD commands, which we describe in the next section, offer more features than the IDUMP command. We recommend that you take advantage of these INFOS II options rather than use IDUMP.

IDUMP Syntax

The syntax of the CLI command to invoke IDUMP follows:

IDUMP *[/N] [/L] [/L=filename] [/T=trailfile] [/B=trailfile] [dumpfilename] [filename]*

We describe these switches earlier in this chapter, in the list of global switches.

IDUMP Examples

In each of the three following examples, the database INVENTORY.DB and all its indexes will be dumped to file 0 on MTA0. In example 2, we created a trailfile named DUMPER.

1. `)IDUMP @MTA0:0 INVENTORY.DB ↓`

2.
```
)IDUMP/T=DUMPER ↓
NAME OF DUMP FILE:    @MTA0:0 ↓
NAME OF INFOS II FILE TO DUMP:    INVENTORY.DB ↓
```

3. `)IDUMP/B=DUMPER ↓`

DDUMP/DLOAD

To protect your database files, we recommend that you copy them onto tape and physically remove them from the computer area. This protects your information from destruction by system failures and other calamities. However, copying your files at periodic intervals does take time. To facilitate this backup method, the INFOS II system provides two utilities called DDUMP and DLOAD.

Please note that while it is possible to use CLI DUMP commands to dump INFOS II system files, we don't recommend it. The INFOS II DDUMP/DLOAD facilities offer greater efficiency and more features than the CLI DUMP command. DDUMP performs consistency checks, such as searching for all the index files associated with a database. DDUMP also ensures that a file is closed to the INFOS II system and all other utilities before performing a dump, and returns an error message if it is not. A CLI DUMP will not do any of these checks. If, however, you decide to use the CLI DUMP command, you must specify the # template after the index or database name to be sure that all volumes in the files are dumped.

Note that the DDUMP utility can resolve filenames through your searchlist. Therefore, you must take appropriate care in specifying filenames with DDUMP, so that you are certain to dump the correct file.

The DDUMP and DLOAD utilities use large block sizes for high speed. They write to disk, unlabeled tape, or labeled tape. Do not attempt to use the CLI LOAD facility to load a file that you have dumped with DDUMP or you will receive an error and the system will not load the file.

Archival and Incremental Dumps

Switches on the DDUMP utility allow you to dump an entire INFOS II file and then make subsequent dumps of only those pages that you have modified since the original dump. These switches are /ARCHIVE and /INCREMENTAL. Using them saves you significant time and space on dump media.

Each page of an INFOS II file has a bit reserved for update status information. When you modify a page, this bit is set. If you then perform an incremental dump, the system copies only those pages marked with the status update bit. When you perform an archival dump, the system clears the status update bit on every page in the file.

To recover a corrupted database file, first load the archival file with DLOAD, and then load the last incremental file. When you load the incremental file, only the updated pages in the archival file will be replaced. The rest of the pages in the files will not be affected.

Because the incremental file contains all modified pages, it will grow steadily larger. You must decide at what point the file becomes unwieldy, and then perform another archival dump. We describe a situation involving an incremental file in the "Examples" section of the DDUMP/DLOAD utilities.

DDUMP Syntax

DDUMP *[/ARCHIVE][/INCREMENTAL][/F][/L] [/L=listfilename] [/IBMLABEL] [/DENSITY=n][/RETAIN=n][/OVERRIDE]* dumpfilename databasefilename

See the list of global switches earlier in this chapter for any switches not described below.

Switch	Description
/ARCHIVE	Dumps the entire contents of the database and its associated volumes onto a dumpfile. When you use DLOAD on an archival file, the INFOS II system recreates the entire file. (You must include either this switch or the /INCREMENTAL switch on the DDUMP command line.)
/DENSITY=n	Specifies the tape density, where *n* is 800, 1600, or 6250. The default value for *n* is the density of the machine on which you have mounted the tape.
/INCREMENTAL	Copies only those pages that have been modified since the last archival dump onto the dumpfile. (You must include either this switch or the /ARCHIVE switch on the DDUMP command line.)
/F	Dumps the INFOS II index(es), the database, and their corresponding volumes, but does not dump any information involving the linked directory structure. If you dump your file with the /F switch, you can reload this file only as a flat structure.
/IBMLABEL	Allows you to dump the file onto a tape with an IBM format label.
/OVERRIDE	Permits you to dump files that were open during an INFOS II system or operating system failure. If you don't use this switch, DDUMP will only dump files that closed normally. Bear in mind, however, that a file dumped with /OVERRIDE may contain severe structural errors. DDUMP will dump differential files when you use this switch. (See Chapter 7 for more information on differential files.)
/RETAIN=n	Specifies a retention time on labeled magnetic tape, where *n* indicates the number of days. (See example, below.)

NOTE: If you use the /L or /L=listfile switch, the output is the DDUMP/DLOAD dialog, not the INFOS II file.

DLOAD Syntax

DLOAD *[/DELETE] [/DENSITY=n] [/F] [/HEADER] [/L] [/L=listfilename] [/N] [/OLD] [/OVERRIDE] [/RETAIN=n]* dumpfilename *[databasefilename]*

See the list of global switches earlier in this chapter for any switches not described below.

Switch	Description
/F	Loads all volumes *flat,* that is, all files are loaded into the same directory. It does not maintain the linked directory structure. If you do not use the /F switch on the dump, you can subsequently load the file flat (without maintaining the directory structure) or load it with the previously dumped linked structure.
/DELETE	Deletes the database if it already exists before loading the dumpfilename. The system displays a warning message for each INFOS II file loaded that did not previously exist. The /DELETE switch is ignored for incremental DLOADs. If you do not specify the /F switch, /DELETE will not delete links to volumes.
/DENSITY=n	Specifies a tape density, where *n* is 800, 1600, or 6250.

/HEADER Prints the initial header information on the load file and checks the database name on the loaded file against the name you specified. DLOAD will not read or load the file if you use this switch.

/IBMLABEL Allows you to load a file that was dumped from a tape with an IBM format label. Note that you also must use the IBMLABEL switch when you dump the file with the DDUMP command.

/N Works the same way as the /HEADER switch, except that the system will read the file.

/OLD Permits you to use an archival dump other than the last one performed.

/OVERRIDE Load from a dumpfile made with the DDUMP/OVERRIDE switch. Loads the differential volume information as well as the standard volume information. (See the INFOS II Backup and Recovery Options, Chapter 7.)

NOTE: Be sure to use revision 4.31 or later of DLOAD to load differential files dumped with revision 4.31 or later of DDUMP. If you use an earlier revision of DLOAD, you will receive an INVALID TAPE MOUNTED or INVALID DISC FILE SPECIFIED error. However, you can use revision 4.31 or later of DLOAD to load differential files you dumped with an earlier version of DDUMP.

DLOAD accepts the + template for the database filename in the same manner as AOS. The + sign alone loads the file without checking the filename. If you use +<chars> then DLOAD checks the trailing characters; with <chars>+ it checks the leading characters.

Examples

Suppose that every Friday you do an archival dump of your inventory database by executing a command such as

```
)DDUMP/ARCHIVE/L=JUNE_20_ARCHIVE @MTA0:0 INVENTORY.DB ↵
```

And, at the end of each day, you execute the following:

```
)DDUMP/INCREMENTAL/L=INCR @MTA0:0 INVENTORY.DB ↵
```

Suppose that then, late Wednesday evening, you have a hardware failure and your database is destroyed on disk. You would then have to load in the most recent archive as follows:

```
)DLOAD/DELETE/L=ARCHIVE_LOAD @MTA0:0 INVENTORY.DB ↵
```

Then, load the most recent incremental dump as follows:

```
)DLOAD/L=INCR_LOAD @MTA0:0 ↵
```

This will update your INFOS II database to the last incremental dump. For details on how to recover any updates made to the file after the incremental dump, refer to Chapter 7, INFOS II Backup and Recovery Options.

In the following example, we use the /ARCHIVE, /RETAIN=n, and /DENSITY=n switches with DDUMP to specify a retention period of 10 days on an archive tape of density 1600.

```
)DDUMP/ARCHIVE/RETAIN=10/DENSITY=1600 @MTB0:0 ↵
```

Using DDUMP and DLOAD with Labeled Tape

The /IBMLABEL switch on DDUMP and DLOAD allows you to write and read tapes with IBM format labels. You must have two terminals in order to use this option. In the following sample sessions, we show user input in monospace type on a line beginning with a prompt. If the input is at the operator's console, the prompt has an asterisk. We show output at the operator's console in *slant monospace* type. We include comments in regular type to tell what happens at certain points in the dialog.

Sample DDUMP Session

In this session, we dump DATABASE_NAME into FILE1 of the tape, at unit MTA0. We indicate to the system that we want it to create a link between the labeled tape and our user directory, and to call it LINK_NAME.

```
*) CONTROL @EXEC OPERATOR ON ↓
   FROM PID N : (EXEC) HH:MM:SS

) MOUNT/IBM/VOLID=VOL1/VOLID=VOL2 LINK_NAME PLEASE MOUNT ME ↓


FROM PID   N : (EXEC) HH:MM:SS
FROM PID   N : (EXEC) ** UNIT MOUNT **
FROM PID   N : MID=X, USER=NAME, PID=Y, VOLID=VOL1
FROM PID   N : VOLID(S) ARE: VOL1, VOL2
FROM PID   N : REQUEST IS 'PLEASE MOUNT ME'
FROM PID   N : RESPOND: CONTROL @EXEC MOUNTED @UNITNAME
FROM PID   N : OR:      CONTROL @EXEC REFUSED


*) CONTROL @EXEC MOUNTED @MTA0 ↓
   FROM PID    N : (EXEC) HH:MM:SS

) DDUMP/ARCHIVE/IBMLABEL [!DIR/I]:LINK_NAME:FILE1 DATABASE_NAME ↓
```

At this point you receive normal DDUMP output. The system will prompt you for VOL2 at the operator's console if the tape runs out. Then your system prompt will return at the other console.

```
) DISMOUNT LINK_NAME ↓


FROM PID   N : (EXEC) HH:MM:SS
FROM PID   N : (EXEC) ** UNIT DISMOUNT **
FROM PID   N : MID=X
FROM PID   N : UNIT(S) ARE: @MTA0
FROM PID   N : RESPOND: CONTROL @EXEC DISMOUNTED


*) CONTROL @EXEC DISMOUNTED ↓
   FROM PID    N : (EXEC) HH:MM:SS
```

Sample DLOAD Session

In this session we will load the file we dumped in the above session. Once again we will mount the tape on the unit MTA0.

```
) MOUNT/IBM/VOLID=VOL1/VOLID=VOL2 LINK_NAME PLEASE MOUNT ME ↓


FROM PID   N : (EXEC) HH:MM:SS
FROM PID   N : (EXEC) ** UNIT MOUNT **
FROM PID   N : MID=X, USER=NAME, PID=Y, VOLID=VOL1
FROM PID   N : VOLID(S) ARE: VOL1, VOL2
FROM PID   N : REQUEST IS 'PLEASE MOUNT ME'
FROM PID   N : RESPOND: CONTROL @EXEC MOUNTED @UNITNAME
FROM PID   N : OR:      CONTROL @EXEC REFUSED


*) CONTROL @EXEC MOUNTED @MTA0 ↓
   FROM PID    N : (EXEC) HH:MM:SS

) DLOAD/ARCHIVE/IBMLABEL [!DIR/I] :LINK_NAME:FILE1 DATABASE_NAME
```

Now you receive normal DLOAD output. VOL2 may be prompted for at the operator's console if the tape runs out. Then your prompt returns.

```
) DISMOUNT LINK_NAME ↓

FROM PID   N : (EXEC) HH:MM:SS
FROM PID   N : (EXEC) ** UNIT DISMOUNT **
FROM PID   N : MID=X
FROM PID   N : UNIT(S) ARE: @MTA0
FROM PID   N : RESPOND: CONTROL @EXEC DISMOUNTED

*) CONTROL @EXEC DISMOUNTED ↓
   FROM PID    N : (EXEC) HH:MM:SS
```

IDELETE

Use the file delete utility, IDELETE, to delete any INFOS II file. The IDELETE utility does the following:

- Uses your searchlist to find the INFOS II database if it is not in your current directory.
- Deletes a database and all its volumes, all its associated indexes, and all their volumes.
- Prevents you from deleting any file that is currently open to INFOS.

Do not use the CLI DELETE command to delete INFOS II databases. It doesn't perform in the same way as IDELETE. It will not use your searchlist to find the database. It won't stop you from deleting a file that is open to INFOS. In addition, if you used the CLI DELETE command, you would have to specify not only the database, but also all of its volumes, all of its indexes, and all of the index volumes. You could easily forget some of these. If you then tried to recreate and rebuild your INFOS II file using a macro, you could fail. Because you would still have parts of the original file left, the system would generate an error message as it attempted to assign the new files the same names as the old files.

CAUTION: You also must not use the CLI DELETE command to delete an individual index file of an INFOS II file. If you did, you would corrupt the rest of the INFOS II file.

How to Invoke IDELETE

If you don't specify a database name on the CLI command line, IDELETE will prompt you for a filename. If you then press NEW LINE instead of supplying a filename, IDELETE will not execute, and the system will return you to the CLI.

Syntax

IDELETE *[/N] [/V] [/L] [/O] [/L=filename] [/T=trailfile] [/B=trailfile] [databasefilename]*

See the description of global switches earlier in this chapter for any switch not described here.

The /V switch tells the system to verify the results of the delete operation.

The /O switch permits you to delete an open file. You would use this switch if you did not want to preserve a file left open in the event of an AOS or INFOS II system failure. If you did want to salvage such a file, you could do so with IVERIFY. We describe how to use IVERIFY in this chapter and in Chapter 7.

Example

The following example uses the /N and /T switches to make sure that IDELETE will execute without errors, but does not perform the delete.

```
)IDELETE/N/T=TRAIL ⤸
FILE TO BE DELETED: INVENTORY.DB ⤸
```

If IDELETE can successfully delete the INVENTORY database and all of its indexes, the system will return you to the CLI. You could then delete the file in this way:

```
)IDELETE/V/B=TRAIL ⤸
```

Since we included the /V switch on the command line, the system will verify the delete as follows:

```
INDEX INVENTORY DELETED DATABASE FILE INVENTORY.DB DELETED
```

IVERIFY

The file verification utility, IVERIFY, has two purposes: to verify that your INFOS II file is structurally valid, and to retrieve statistics on database and index usage. The statistics will be more meaningful once you are familiar with Chapter 10. In addition, if a system failure occurs, you must run IVERIFY on files you were processing in Standard File Mode. IVERIFY will close the files. We explain how to do this in Chapter 7.

IVERIFY follows AOS conventions to locate the database file. It checks your working directory first and then the directories in your searchlist. IVERIFY does, however, expect to find your index file in the same directory as the database file.

When you invoke IVERIFY, it automatically builds two temporary work files in the current working directory. The files are ?nnn.PLB.IV.TMP and ?nnn.PLS.IV.TMP, where nnn is a three-digit number. IVERIFY also builds two additional work files, ?nnn.DUN.IV.TMP and ?nnn.DUC.IV.TMP, which it uses when it validates record use counts. At the successful completion of IVERIFY, the system automatically deletes the work files. However, if IVERIFY terminates abnormally, you will have to use the AOS delete facility to get rid of the work files.

Remember that the IVERIFY utility verifies the structural integrity of your file, but not its contents. It is up to you to ensure the validity of your data.

How to Invoke IVERIFY

You must specify a database filename for IVERIFY, either on the CLI command line or in response to a filename prompt. IVERIFY then checks the database and all its indexes for validity and produces statistics for that database and all its indexes.

Syntax

IVERIFY *[/A] [/X] [/Y] [/L] [/L=filename] [databasefilename]*

Normally, IVERIFY will try to continue processing in spite of nonfatal errors. You can alter this by using one of IVERIFY's switches.

Switch	Description
/A	Specifies that you want IVERIFY to terminate on any error (including those that IVERIFY could correct). This switch is especially useful when you have a large file to verify and you want to stop the verification process as soon as the system discovers an error.

/X Specifies that you want IVERIFY to ignore data record use counts. You should only use this switch if you have already run IVERIFY and received a use count error and discovered the abnormal condition that caused the error. Otherwise, IVERIFY could "correct" the use count numbers and you wouldn't be alerted to the condition that had caused the use counts to be incorrect.

/Y Directs IVERIFY to add one to every occurrence number in the file. You might want to do this if you have any keys with zero occurrence numbers (see the note below). You should back up the database prior to running IVERIFY/Y, because if there is a system failure during the IVERIFY/Y, the database will be corrupted and you will have to load the backup.

NOTE: Earlier versions of the IXLOAD utility sometimes assigned a zero occurrence number to the first key loaded in a file. AOS INFOS finds no file consistency problems with zero occurrence numbers. IVERIFY will merely detect the condition and generate a warning message. But since AOS/VS INFOS will not tolerate zero occurrence numbers, you cannot transport files from AOS to AOS/VS unless you first correct any zero occurrence numbers.

Correcting a zero occurrence number may require extensive changes to the file, therefore it is not done automatically. It can be corrected either by rebuilding the file or by using IVERIFY with the /Y switch (see above).

Observe the following cautions with regard to the /Y switch:

- The switch applies only to files created with revisions of AOS INFOS earlier than 4.31. If you built your database with revision 4.31 or later, you will never need to use the /Y switch.
- If you need to use the /Y switch, only run it once on any given file.

Note that if you store occurrence numbers in either your partial or database records, the /Y switch will not increment them; you must correct them yourself. Also, IRECOVER will not recover any changes that IVERIFY makes to your file. Thus, if you are using INFOS II request logging, be sure to make a backup of your database after running IVERIFY/Y to use with subsequent recovery sessions. See Chapter 7 for more information on backup and recovery.

How to Interpret the IVERIFY Statistics

The Database Statistics column in Table 5-2 represents the statistics produced by IVERIFY. The numeric value that IVERIFY returns for each statistic is in decimal. The Meaning column gives a general explanation of each statistic.

Table 5-2. IVERIFY Statistics

Database Statistics	Meaning
*** START DATABASE: name ***	
NUMBER OF DATABASE PAGES:	Total number of 2048- or 4096-byte pages currently allocated to the database.
DATABASE PAGE SIZE:	The size of the page (2048 or 4096) you selected for the database with ICREATE.
TOTAL NUMBER OF RECORDS:	This number includes logically deleted and permanent records, as well as those with a use count greater than 0.
MAXIMUM RECORDS PER PAGE:	The maximum number of records on a single 2048- or 4096-byte page.

(continues)

Table 5-2. IVERIFY Statistics

Database Statistics	Meaning
MINIMUM RECORDS PER PAGE:	The minimum number of records on a single 2048- or 4096-byte page.
MAXIMUM RECORD LENGTH:	The length, in bytes, of the largest record in the database.
MINIMUM RECORD LENGTH:	The length, in bytes, of the smallest record in the database.
MAXIMUM RECORD USE COUNT:	The largest number of keys linked to a single data record.
MINIMUM RECORD USE COUNT:	The smallest number of keys linked to a single data record.
DATABASE ALLOCATED SPACE:	The actual number of bytes allocated to store the data records.
DATABASE CONTROL SPACE:	The total number of bytes allocated for data record vectors.
DATABASE COMPRESSED SPACE:	The number of bytes you saved by using data record compression.
DATABASE AVAILABLE SPACE:	The number of bytes remaining for use in the pages currently allocated to the database.
DATABASE WASTED SPACE:	The number of bytes in the page(s) allocated to the database that neither you nor the INFOS II system can use. This space often results from data records with odd byte lengths.
EMPTY DATABASE PAGES:	The number of pages allocated to the database that the INFOS II system has not yet used for allocated or control space.
LOGICALLY DELETED RECORDS:	The number of data records currently marked as logically deleted.
PERMANENT RECORDS:	The number of data records with a use count exceeding 255.
INDIRECT RECORDS:	The number of data records relocated through rewriting.
DELETED RECORDS:	The number of data records physically deleted from the database. The vectors for these records still exist in the database control space. If you are using space management, this space can be reused.
TOTAL USE COUNT:	The sum of all the use counts.
AVERAGE RECORD LENGTH:	The average length of all records in the database.
AVERAGE RECORD USE COUNT:	The average of the use counts of all records in the database.

(continued)

Table 5-2. IVERIFY Statistics

Cumulative Index Statistics	Meaning
*** START INDEX: indexname ***	
NODE COUNT:	The total number of nodes currently in use in the entire index.
PAGE COUNT:	The total number of pages currently in use in the subindex.
INDEX PAGESIZE:	The size of the page (2048 or 4096) that you selected for the index with ICREATE.
SUBINDICES:	The total number of subindexes currently defined for the index.
LEVEL 0 ENTRIES:	The total number of index entries that have a Data Record Link Field.
TOTAL ENTRIES:	The total number of entries in the entire index.
DATA BYTES:	The sum of the lengths of all the keys at all index tree levels.
COMPRESSED BYTES:	The number of bytes you saved by using compressed keys.
CONTROL BYTES:	The total number of bytes allocated for index entry vectors.
AVAILABLE NODE BYTES:	The number of bytes available in all the index nodes.
AVAILABLE PAGE BYTES:	The number of bytes available in the page(s) allocated to the index.
WASTE BYTES:	The number of unused bytes in index nodes that have split. These bytes probably cannot be used (see Chapter 10 for an explanation of index nodes). Also the wasted bytes resulting from odd lengths of keys and partial records.
LOGICAL DELETIONS:	The number of keys currently marked as logically deleted.
DUPLICATE KEYS:	The total number of duplicate keys in the entire index.
MAX KEY LENGTH:	The length, in bytes, of the largest key in the index.
MIN KEY LENGTH:	The length, in bytes, of the smallest key in the index.
MAX PARTIAL RECORD LENGTH:	The length, in bytes, of the largest partial record in the index.
MIN PARTIAL RECORD LENGTH:	The length, in bytes, of the smallest partial record in the index.
EMPTY INDEX NODES:	The total number of empty nodes in the entire index. Nodes would be empty if you haven't used the index yet, or if you have physically deleted all previously existing entries.

(concluded)

If your database is structurally valid, IVERIFY will terminate the output with the following line:

**** INFOS II FILE STRUCTURE HAS BEEN VERIFIED AND IS CORRECT ****

IXLOAD

The IXLOAD utility allows you to build an INFOS II subindex using significantly less processing time than usual. The utility accepts a sequential file and uses it to construct a subindex. You can build the subindex in a newly created INFOS II file or in an empty subindex of a multilevel file.

To prepare to use the utility, first create the INFOS II subindex that will receive the information. This subindex must be an empty subindex; it cannot contain any keys. If you want to create a new INFOS II file, create the file from the top down. Again, as you add subindexes, they must be empty. Set up whatever structure you want in the file to the point where you begin using IXLOAD.

Next, you will need a SAM file. You can build separate inverted subindexes from the same SAM file. However, if you plan to build multiple subindexes that are not inversions, you must have a separate file for each subindex. If you want to create a multilevel file, you must build your file from the top level down.

The SAM file can have variable- or fixed-length records. When you invoke the IXLOAD utility, you will specify the format of the SAM file record. Each record will contain your primary key, alternate keys (if any), partial records (if any), the data record, and a 4-byte blank field. The INFOS II system might need this field for entering feedback information while it is processing the file. You must provide these 4 bytes whether or not you use alternate keys.

With Optimized Record Distribution, the primary key field must reserve 1 byte for the record merit factor. If you omit this information, but have optimized record distribution enabled in the INFOS II file, the system default merit factor is 1. If you have variable-length keys, you must have a 1-byte field containing the length of the key. If you have variable-length database records, you must have a 2-byte field containing the length of the database record.

Once you have made these preparations, you can build the INFOS II subindex with the IXLOAD utility.

Syntax

IXLOAD *[/N][/T][/T=trailfile][/B=batchfile] [/O] [/S][/M]* samfilename

See the description of global switches earlier in the chapter for any switch not described below.

Switch	Description
/O	Overrides the error *(IONIO) KEY NOT IN ORDER* on the primary (first) load.
/S	Sorts the SAM file by primary key before executing the utility.
/M	Redefines the between-key delimiter. The default delimiter is a comma (,).

NOTE: If IXLOAD aborts during its execution it might affect your database use counts. Also, it may waste space in the INFOS II file. While this space will not affect the function or structure of your file, you can never reuse or delete the space.

To avoid this, we strongly recommend that you archive your file with the DDUMP utility before you start IXLOAD.

If you do not use the /S switch or the /O switch, and IXLOAD has errors on the primary load, IXLOAD will process a SORT on the primary key and reload the subindex. The /S switch processes the SORT before the utility attempts to load the subindex; the /O switch prevents the utility from processing the sort if the error occurs. IXLOAD terminates immediately and displays an error message if the sort terminates abnormally.

IXLOAD Dialog

When you successfully invoke IXLOAD, it begins a dialog in which you specify the structure of your SAM file record. Note that when IXLOAD asks for an offset, you supply a zero-byte position (the second byte of a record has an offset of one). The following is a sample IXLOAD dialog.

IXLOAD Prompt	Your Action
KEY FILE:	Enter the pathname of the SAM file. The default is to press the NEW LINE key, which will return you to the CLI.
MAXIMUM KEYFILE RECORD LENGTH:	This question applies to variable SAM files. There is no default value.
** * * PRIMARY FILE * * **	
INDEX TO BE LOADED:	Enter the index filename. There is no default value.
KEY PATH TO SUBINDEX [MAIN INDEX]:	Enter the key path to the blank subindex. The default is the main index.
KEY LENGTH (ENTER LENGTH OR [V]):	Enter the length of the primary key. The default is variable length keys.
OFFSET OF KEY LENGTH:	This question applies to variable length keys only. If you answered V to the previous prompt, you must now supply the offset to the 1-byte field that contains the varying keys' length. There is no default value.
MAXIMUM LENGTH OF KEY FIELD:	Enter the maximum length. There is no default value.
OFFSET OF KEY:	Enter the offset. There is no default value.
PARTIAL RECORD LENGTH [0]:	Enter the partial record length, in bytes. The default is zero — or no partial records.
OFFSET OF PARTIAL RECORD:	If you choose partial records, enter the offset. There is no default value.

DATABASE RECORD LENGTH [ENTER LENGTH OR [V]]:	Enter the length you want, in bytes. The default is variable length.
OFFSET OF DATABASE RECORD LENGTH:	If you responded to the previous prompt with V, enter the offset of the variable records' 2-byte length description. There is no default value.
OFFSET OF DATABASE RECORD:	Enter the offset. There is no default value.
OFFSET OF RECORD MERIT FACTOR [DEFAULT MERIT FACTOR]:	Enter the merit factor you want. The default is the lowest merit factor.
OFFSET OF FEEDBACK AREA:	The length of this field must equal 4 bytes. There is no default value. You must have this field blank in your SAM file records.
HOW MANY ALTERNATE KEY INDICES TO LOAD? [0]:	Enter the number of alternate key indexes you want. The default is zero.

If you choose to have alternate key indexes, the IXLOAD utility will ask you to supply information about each index.

** * * ALTERNATE KEY * * **

INDEX FILENAME TO BE LOADED [indexfilename]:	The default is the index filename. If you want the index to have a different name, enter it here.
KEY PATH TO SUBINDEX [MAIN INDEX]:	Enter the keypath to the other blank subindex, unless you want the keypath to the main index.
KEY LENGTH (ENTER LENGTH OR [V]):	Enter the key length in bytes. The default is the variable key length.
OFFSET OF KEY LENGTH:	If your response to the previous prompt was V, enter the offset to the 1-byte field that contains the lengths of the varying keys. There is no default value.
MAXIMUM LENGTH OF KEY FIELD:	Enter the maximum length. There is no default value.
OFFSET OF KEY:	Enter the alternate key position in the SAM file records. There is no default value.
PARTIAL RECORD LENGTH: [0]:	Enter the length in bytes. The default is zero (no partial records).
OFFSET OF PARTIAL RECORD:	Enter the offset. There is no default value.

** * * ALTERNATE KEY * * **

The above ALTERNATE KEY dialog will repeat as many times as necessary, depending on how many ALTERNATE KEY INDICES you specified.

After you give these specifications, IXLOAD builds an INFOS II subindex. The utility saves time from its lowered IPC communication overhead, since it need issue only one INFOS II request.

NOTE: A single IXLOAD request will use 100 percent of your INFOS II system processing time until it completes. Other users will be forced to wait; it could appear as though they are hanging. Therefore, we strongly recommend that you do not issue an IXLOAD request during peak hours, and that you warn other users before running the utility.

Restrictions

You should note the following restrictions of the IXLOAD utility.

- The INFOS II main index or subindex that will contain the loaded file must be empty.
- The maximum SAM file record size that IXLOAD will accept is 8,192 bytes.
- The maximum size limit of key file records is the page size minus the page header size plus two words.
- If you use the alternate key feature, or the /S or /O switches, IXLOAD will invoke the AOS Sort/Merge utility. Sort/Merge might use considerable space in IXLOAD's working directory for temporary file space, and could slow down the loading process. See the *Sort/Merge Utility User's Manual (AOS, AOS/VS)* for more information on this utility.
- You cannot enter inverted keys within the loaded subindex.
- The SAM file must be a fixed or variable length file only.
- Either the SAM file keys must be in sorted order, or you must use the /S switch. Using the /S switch performs a sort on the file before it is loaded.
- You cannot load more than 255 subindexes or 254 alternate keys. Otherwise you exceed the limit for the data record use count.

Errors

Table 5-3 lists the IXLOAD error returns. Please note that if IXLOAD aborts while performing an inverted load, the database use counts will be incorrect. This will cause IVERIFY to return an error unless you specify IVERIFY/X. We recommend that you rebuild your INFOS II file if this happens. While IXLOAD will not alter the index and database externally in the event of an error, you might find your file size has increased, as the utility could have partially created an index node structure.

Table 5-3. IXLOAD Error Returns

Error	What Happens
Keys not in sorted order. No subindex defined. Subindex not empty. INFOS II errors related to writing keys and records. A key, partial record, or data record is too long. The key file record length is too large. Key (SAM) file not found. Specified index not found.	IXLOAD aborts.
Too many alternate loads requested. Index specified not an INFOS II index. Invalid SAM file type.	IXLOAD repeats the question.

IXLOAD Output

On the normal completion of the IXLOAD utility, your SAM file information will be in your INFOS II file. The SAM file could contain information in the feedback fields, and the system would sort this information on the last key loaded from it.

If a sort was processed, it will create a file called samfilename.SORT, containing the SORT utility's output.

INDEXCALC

Use INDEXCALC to predict the size and performance of an INFOS II file before you create it. When you supply the required information to INDEXCALC, the utility recommends file creation parameters for optimum performance, and lists the maximum limits for those parameters with respect to file size and growth.

How to Invoke INDEXCALC

Use the following command line to invoke INDEXCALC from the CLI:

```
INDEXCALC
```

INDEXCALC has no switches.

The utility then begins a dialog to determine your hypothetical file's parameters.

Dialog Summary

The INDEXCALC dialog consists of a series of questions about your processing needs. The questions vary, depending upon your previous responses. We list the questions in Table 5-4. This list is not a sample dialog, but a summary of all possible questions the system might display.

There are no defaults in the INDEXCALC dialog. If you press NEW LINE or CR in response to a question, the utility will either repeat the question or wait until you give a valid response. Figure 5-1 shows a sample INDEXCALC output.

Table 5-4. INDEXCALC Question Summary

INDEXCALC Prompt	Your Response	Comments
HARD COPY OF ALL RESULTS, Y OR N:	Y	You'll get the prompt that asks you to specify a HARD COPY HEADER.
	N	Your output will not have a header, but you will be able to selectively print any results you want at the line printer by answering Y to the prompt HARD COPY OF THESE RESULTS, Y OR N: after your results are on the terminal.
SPECIFY THE HEADING YOU WANT ON YOUR HARD COPY HEADER:	header	Enter an alphanumeric string to identify your results.
SPECIFY KEY LENGTH:	number	Enter a decimal number from 1 to 255 that gives the largest key length in the index (or subindex) in bytes (characters).
SPECIFY PARTIAL RECORD LENGTH:	number	Enter a decimal number giving the longest partial record length in the index. Enter 0 if partial records will not be used.
DO YOU WANT TO ALLOW SUB-INDEX? (Y or N)	Y	INDEXCALC adjusts index entry size to accommodate a pointer to another subindex.
	N	Index size is not adjusted for a subindex pointer.

(continues)

Table 5-4. INDEXCALC Question Summary

INDEXCALC Prompt	Your Response	Comments
WHAT INFORMATION SHOULD THE PROGRAM SOLVE FOR? NUMBER OF LEVELS (L), PAGE SIZE (P), OR NUMBER OF INDEX ENTRIES (E):	L	When you solve for the number of index tree levels in the file, INDEXCALC asks you for the page size and the number of index entries.
	P	When you solve for the page size, INDEXCALC asks you for the number of index entries and levels.
	E	When you solve for the number of index entries, INDEXCALC asks you for the page size and the number of index levels.
SPECIFY THE PAGE SIZE YOU WANT (2048 OR 4096 BYTES):	number	The AOS INFOS II system uses only two page sizes, 2048 and 4096 bytes. The 2048 size is the ICREATE default.
SPECIFY NUMBER OF INDEX ENTRIES:	number	Enter total number of index entries for the index.
SPECIFY NUMBER OF LEVELS:	number	Give a decimal number between 1 and 32 for the number of tree levels.
ITERATE (I), CHOOSE ANOTHER PARAMETER (C), RESTART (R), OR STOP (S):	I	When you solve for the number of index tree levels, INDEXCALC asks you to specify the number of entries. It will reuse the page size. When you solve for page size, INDEXCALC expects you to specify the number of index entries and levels. When you solve for the number of entries, INDEXCALC asks you for the number of tree levels. It will reuse the page size.
	C	INDEXCALC will repeat the question, WHAT INFORMATION SHOULD THE PROGRAM SOLVE FOR?
	R	The program restarts at the beginning.
	S	INDEXCALC returns you to the CLI.

(concluded)

```
SOLVING FOR NUMBER OF LEVELS

KEY LENGTH =         6   PART. REC. LENGTH =   0  SUB-INDEXES ALLOWED =       Y
PAGE SIZE =       2048   NODE SIZE =        2042  MIN. INITIAL NODE SIZE =   112
NUMBER OF LEVELS =   4
DESIRED ENTRIES =    1000000.                    MAXIMUM ENTRIES =   85688400.

ROOT NODE B.F. =   100   INTERM. NODE B.F. = 101  LEVEL 0 NODE B.F. =        84

 LEVEL   !       NODES      PAGES    !   MAX NODES  MAX PAGES    !
    0    !       11905.      11905.   !    1020100.    1020100.   !
    1    !         118.        118.   !      10100.      10100.   !
    2    !           2.          2.   !        100.        100.   !
    3    !           1.          1.   !          1.          1.   !
-------------------------------------------------------------------------------
TOTALS   !       12026.      12026.   !    1030301.    1030301.   !
 BYTES   !                  24629248.   !             2110056448.   !
-------------------------------------------------------------------------------

ROOT SIZE(BYTES)                80.   !                 2040.   !

SOLVING FOR PAGE SIZE

KEY LENGTH =         6   PART. REC. LENGTH =   0  SUB-INDEXES ALLOWED =       Y
PAGE SIZE =       4096   NODE SIZE =        4090  MIN. INITIAL NODE SIZE =   112
NUMBER OF LEVELS =   3
DESIRED ENTRIES =    1000000.                    MAXIMUM ENTRIES =    6930014.

ROOT NODE B.F. =   202   INTERM. NODE B.F. = 203  LEVEL 0 NODE B.F. =        169

 LEVEL   !       NODES      PAGES    !   MAX NODES  MAX PAGES    !
    0    !        5918.       5918.   !      41006.      41006.   !
    1    !          30.         30.   !        202.        202.   !
    2    !           1.          1.   !          1.          1.   !
-------------------------------------------------------------------------------
TOTALS   !        5949.       5949.   !      41209.      41209.   !
BYTES    !                  24367118.   !              168792064.   !
-------------------------------------------------------------------------------

ROOT SIZE(BYTES)              640.   !                 4080.   !

SOLVING FOR NUMBER OF ENTRIES

KEY LENGTH =         6   PART. REC. LENGTH =   0  SUB-INDEXES ALLOWED =       Y
PAGE SIZE =       4096   NODE SIZE =        4090  MIN. INITIAL NODE SIZE =   112
NUMBER OF LEVELS =   4
DESIRED ENTRIES = 1406792842.                    MAXIMUM ENTRIES = 1406792842.

ROOT NODE B.F. =   202   INTERM. NODE B.F. = 203  LEVEL 0 NODE B.F. =        169

 LEVEL   !       NODES      PAGES    !   MAX NODES  MAX PAGES    !
    0    !     8324218.    8324218.   !    8324218.    8324218.   !
    1    !       41006.      41006.   !      41006.      41006.   !
    2    !         202.        202.   !        202.        202.   !
-------------------------------------------------------------------------------
TOTALS   !     8365427.    8365427.   !    8365427.    8365427.   !
 BYTES   !                34264788992.   !            34264788992.   !
-------------------------------------------------------------------------------

ROOT SIZE(BYTES)              4080.   !                 4080.   !
```

DG-15264

Figure 5-1. Sample INDEXCALC Output

End of Chapter

Chapter 6
The INQUIRE Utility

Use the file inquire utility, INQUIRE, to examine and modify an INFOS II file. Through INQUIRE, you can execute all INFOS II system commands (READ, WRITE, DEFINE SUBINDEX, etc.), and you have the full range of keyed and relative access techniques described in Chapter 2. INQUIRE also provides a group of command modifiers that function like general processing features, including suppress database access, file inversion, and lock, among others. Additional, non-INFOS II, commands are also available with INQUIRE. These allow you to reproduce processing packets, to change input and output files, and to set repeat counts. We suggest that you read Chapters 9 and 10 before using the advanced features of this utility.

By default, INQUIRE sets the current position on the accessed key at the completion of the requested operation. However, for each command you issue, you can request that INQUIRE not set the current position.

On a READ command, INQUIRE will display the database record if one is present. If the key accessed for the READ does not have a database record, or if the key is a duplicate, the system will inform you of this.

INQUIRE allows you to perform numerous functions, including the following:

- Write keys, data records, and partial records at any subindex level.
- Rewrite data records and partial records.
- Use inversion with the WRITE and REWRITE commands to link new or existing keys to existing data records.
- Delete keys and data records either physically or logically.
- Reinstate logically deleted keys and their data records.
- Define, link, and delete subindexes.
- Acquire data record feedback.

Although you can write keys and data records through INQUIRE, this utility is not designed for loading large amounts of data into an INFOS II file. It would take a lot of time compared to your other options. The AOS SORT/MERGE utility, the IXLOAD utility, or an application program would allow you to load large amounts of data more easily.

INQUIRE Syntax

Use the following CLI command syntax to invoke INQUIRE:

INQUIRE *[/M] [/T=trailfile] [/B=trailfile] [indexfilename]*

Switch	Purpose
/M	Allows you to change the default between-key delimiter and the indicators for occurrence number, generic key, and approximate key. The /M switch is unique to INQUIRE. We explain this switch in more detail in the "Keyed Access" section, later in this chapter.
/T=trailfile	Stores your interaction with INQUIRE in *trailfile.*
/B=trailfile	Uses the contents of *trailfile* as input to INQUIRE.

INQUIRE Dialog

INQUIRE prompts you for an index filename if you do not provide one on the command line. It then issues the following prompt:

NUMBER OF LOCKS: [0]:

The default number of locks is 0, but you can specify up to 32 outstanding locks. It's a good idea to specify some locks if other users are also processing the file you are modifying. Locking a record while you modify it ensures that no other users will gain access to it until you are finished and have unlocked it. If you specify an invalid number of locks, INQUIRE will repeat the prompt until you enter a valid number. See Chapter 3 for more information on locks.

INQUIRE then issues this prompt:

ENTER C TO GET LIST OF COMMANDS:

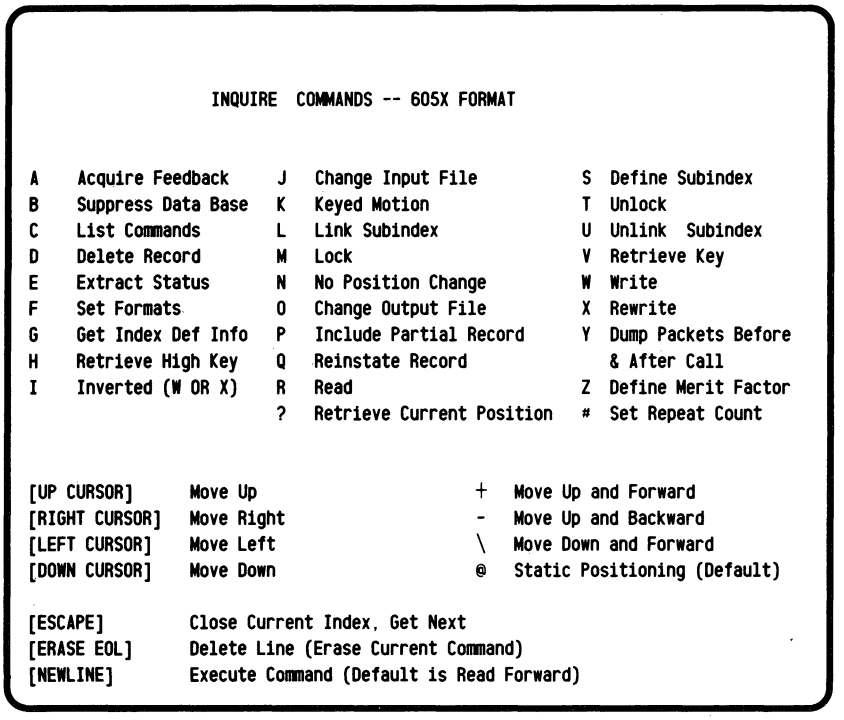
COMMANDS:

If you want a display of all the INQUIRE commands and command modifiers, press C (do not press the NEW LINE key afterwards). Figures 6-1, 6-2, and 6-3 show what this display looks like on various terminal screens. (INQUIRE determines what type of terminal you have and gives you the appropriate display.) After it displays the list, INQUIRE will give you the *COMMANDS:* prompt again. The ensuing dialog will depend on what command you issue next. The following sections describe the INQUIRE commands and command modifiers.

```
                    INQUIRE COMMANDS

A       ACQUIRE FEEDBACK          N       NO POSITION CHANGE
B       SUPPRESS DATA BASE        O       CHANGE OUTPUT FILE
C       LIST COMMANDS             P       INCLUDE PARTIAL RECORD
D       DELETE RECORD             Q       REINSTATE RECORD
E       EXTRACT STATUS            R       READ
F       SET FORMATS               S       DEFINE SUBINDEX
G       GET INDEX DEF INFO        T       UNLOCK
H       RETRIEVE HIGH KEY         U       UNLINK SUBINDEX
I       INVERTED (W OR X)         V       RETRIEVE KEY
J       CHANGE INPUT FILE         W       WRITE
K       KEYED MOTION              X       REWRITE
L       LINK SUBINDEX             Y       DUMP PACKETS BEFORE & AFTER CALL
M       LOOK                      Z       DEFINE MERIT FACTOR
                                  ?        RETRIEVE CURRENT POSITION
^       MOVE UP                   ESCAPE   CLOSE CURRENT INDEX GET NEXT
->      MOVE FORWARD              RUBOUT   DELETE LINE
<-      MOVE BACK                 #        SET REPEAT COMMAND
LF      MOVE DOWN                 CR       EXECUTE COMMAND (DEFAULT=READ FWD)
+       MOVE UP AND FORWARD       -        MOVE UP AND BACKWARD
@       STATIC POSITIONING        \        MOVE DOWN AND FORWARD
        DEFAULT COMMAND IS READ, DEFAULT MOTION IS STATIC
```

DG-15259

Figure 6-1. INQUIRE Command Display (6012, 4010I)



DG-15260

Figure 6-2. INQUIRE Command Display (605X Format)

```
        INQUIRE COMMANDS FOR HARDCOPY TERMINALS


A    ACQUIRE FEEDBACK          N       NO POSITION CHANGE
B    SUPPRESS DATA BASE        O       CHANGE OUTPUT FILE
C    LIST COMMANDS             P       INCLUDE PARTIAL RECORD
D    DELETE RECORD             Q       REINSTATE RECORD
E    EXTRACT STATUS            R       READ
F    SET FORMATS               S       DEFINE SUBINDEX
G    GET INDEX DEF INFO        T       UNLOCK
H    RETRIEVE HIGH KEY         U       UNLINK SUBINDEX
I    INVERTED (W OR X)         V       RETRIEVE KEY
J    CHANGE INPUT FILE         W       WRITE
K    KEYED MOTION              X       REWRITE
L    LINK SUBINDEX             Y       DUMP PACKETS BEFORE & AFTER CALL
M    LOCK                      Z       DEFINE MERIT FACTOR
                               ?        RETRIEVE CURRENT POSITION
^    MOVE UP                   ESCAPE   CLOSE CURRENT INDEX, GET NEXT
>    MOVE FORWARD              DEL      DELETE LINE
<    MOVE BACK                 #        SET REPEAT COMMAND
!    MOVE DOWN                 CR       EXECUTE COMMAND (DEFAULT=READ FWD
+    MOVE UP AND FORWARD       -       MOVE UP AND BACKWARD
@    STATIC POSITIONING        \       MOVE DOWN AND FORWARD
     DEFAULT COMMAND IS READ, DEFAULT MOTION IS STATIC
```

DG-15261

Figure 6-3. INQUIRE Command Display (Hard Copy)

Exiting INQUIRE

When you are finished with INQUIRE and want to close the file, press the ESC key in response to the *COMMANDS:* prompt, and then press the NEW LINE key when you are prompted for an index name. The ESC closes the file and NEW LINE returns you to the CLI.

Overview of INQUIRE Commands

The INQUIRE commands shown in Figures 6-1, 6-2, and 6-3 fall into the following four categories.

1. Function Commands: These are the equivalent of the INFOS II commands in Chapter 3. For example, R is the same as READ and V is the same as RETRIEVE KEY.

2. Command Modifiers: These work in conjunction with an INQUIRE command. For example, you can write a key and use B to suppress database access.

3. Motion Control Specifiers: These indicate a direction of relative motion. For example, use → for forward motion and ← for backward motion.

4. Unique INQUIRE Commands: These commands are unique to the INQUIRE utility. For example, C gives you a list of INQUIRE commands and F lets you set the record formats.

Table 6-1 lists the commands in each of these categories.

Table 6-1. INQUIRE Command Categories

INFOS II Function Commands	INFOS II Command Modifiers	Motion Control Specifiers				Unique Inquire Commands	
		605X and 6012	4010I	Hard Copy	Command		
R - Read	B - Suppress Database	↓	↓	!	Move Down	A	Acquire Feedback
*NL(↓) - Read Forward	I - Invert	↑	↑	^	Move Up	C	List Commands
W - Write	K - Keyed Access	→	→	>	Move Forward	F	Set Formats
X - Rewrite	M - Lock	←	BS	<	Move Backward		
D - Delete	T - Unlock	+	+	+	Move Up and Forward	J	Change Input File
Q - Reinstate	N - No Position Change	-	-	-	Move Up and Backward	O	Change Output File
S - Define Subindex	P - Include Partial Record	\	\	\	Move Down and Forward	Y	Dump Packets
L - Link Subindex	Z - Volume Merit Factor	@	@	@	Static	#	Set Repeat Count
						?	Retrieve Current Position
U - Delete Subindex						NL	Execute Command
E - Retrieve Status						CTRL-K ERASE EOL	Delete Line (605X)[1,2]
V - Retrieve Key						DEL	Delete Line (4010I)[1]
H - Retrieve High Key						DEL	Delete Line (Hard Copy)[1]
G - Retrieve Subindex Definition					CTRL-U RUBOUT	Delete Line (6012)[1]	
ESCAPE - Close File (Get next index)							

[1]Typing any undefined character causes INQUIRE to respond with ??, and terminate the line.

[2]Typing a CTRL-K or ERASE EOL on a 605X terminal will echo a CTRL-U on the screen.

You can respond to the *COMMANDS:* prompt with a string of one or more single key stroke commands, terminated by NEW LINE (⤸). Note that you do not type in a space between key strokes; INQUIRE automatically inserts a blank after each key you press.

Default Values

The default command is READ FORWARD, which you can specify by pressing NEW LINE in response to the *COMMANDS:* prompt.

That is:

```
COMMANDS:  ⤸
```

retrieves the record for the next higher key relative to the current position.

At the completion of the requested operation, INQUIRE automatically sets the current position on the key to which it gained access. Also, note that for every command except READ, the system will assume static relative motion if you do not include a motion control specifier or the keyed access command (K) in the command string. (For READ, the default motion is forward.) Therefore, if your response to the *COMMANDS:* prompt is

```
COMMANDS:  V ⤸
```

the system will display the key for your current position.

We discuss access techniques and the INQUIRE function commands and command modifiers in the following sections.

Access Techniques

INQUIRE allows you to use all three INFOS II system access techniques: keyed, relative, and the combination of keyed and relative motion.

Keyed Access

Keyed access is the default access method for the WRITE and DELETE commands only. For all others, the default is relative access. The command modifier K specifies keyed access. Whenever the command string includes K, INQUIRE prompts you for the key. For example, you can specify a keyed read in this way:

```
COMMANDS:  K R ⤸
KEY:
```

You can respond to the *KEY:* prompt with one or more keys in a keypath, and then terminate your key specification by pressing NEW LINE. The system searches for keys from the top of the index, regardless of the current position. Use a between-key delimiter to separate individual keys in a keypath leading down through the levels of a multilevel index. The default delimiter is the comma, but you can change the delimiter to any character you want. To change the delimiter, or any of the other default values for key characteristics, use the /M switch when you invoke INQUIRE.

The /M Switch

When you use the /M switch on your INQUIRE command line, INQUIRE prompts you for new values for the following key characteristics:

```
NEW BETWEEN-KEY DELIMITER [,]:
NEW OCCURRENCE NUMBER INDICATOR [#]:
NEW GENERIC KEY INDICATOR [-]:
NEW APPROXIMATE KEY INDICATOR [+]:
```

The default values appear in brackets. You can change any of these values as they appear on your screen. If you just press NEW LINE, then INQUIRE will assume that you want to keep the default.

As you can see, the default between-key delimiter is the comma. But your keys might use the comma as a character, such as in a date. In this case, you wouldn't use the comma as a delimiter. You might change it, for example, to a slash (/). When you receive the prompt for a new delimiter, respond in this way:

```
NEW BETWEEN-KEY DELIMITER [,]:    /↵
```

You can now do a keyed access to the purchase order 1P206A written on August 4, 1983, shown in Figure 6-4. Specify the multilevel key as follows:

```
KEN/AUGUST 4, 1983/1P206A↵
```

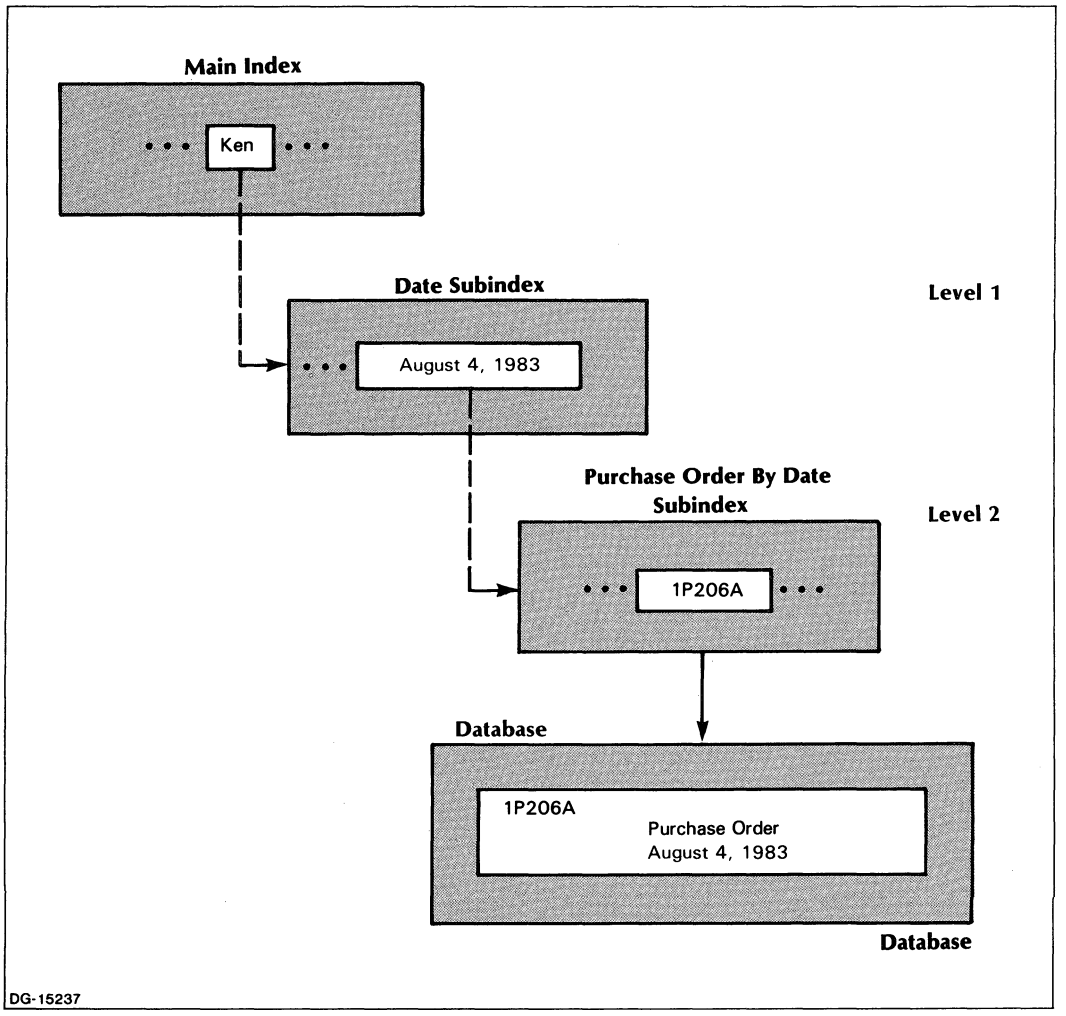


Figure 6-4. Multilevel INFOS II File

Change any of the indicators — occurrence number, generic key, or approximate key — in the same way.

Relative Access

The WRITE and DELETE commands are the only commands for which you cannot use relative access alone; they require keyed access (the default) or combined keyed and relative access. For all other commands, however, INQUIRE assumes relative access unless K appears in the command string.

You have eight motion control specifiers, one for each direction of motion (see Table 6-1). For a READ command, which you specify by entering R or pressing NEW LINE, the default motion is forward. For all other INFOS II commands and for appropriate INQUIRE commands, the system assumes static motion. Therefore, if the current position is above the index and you want to read the record for the first key, specify a back slash:

```
COMMANDS:  \ ↵
```

Note that INQUIRE displays this command as:

```
COMMANDS: DOWN FORWARD
```

If you want to retrieve the next lower key relative to current position, specify:

```
COMMANDS:  ← V ↵
```

NOTE: Motion control specifiers vary with terminal types (see Table 6-1). Be sure you have properly set your terminal type in the CLI with the CHARACTERISTICS command. INQUIRE will check your terminal command list. If your characteristics are wrong, INQUIRE could function incorrectly.

Combined Keyed and Relative Access

The combination of keyed and relative access is appropriate in a multilevel file only. It is most advantageous when you are processing at a lower level subindex, since by using it you don't need to specify a multilevel key for each request. For example, suppose you use the static motion specifier in the following command string:

```
COMMANDS:  K W B @ ↵
KEY: 021 ↵
```

The system will write key 021 into the subindex in which the current position is set. (We will explain B — the Suppress Database command modifier — later in this chapter.)

Suppose instead you had entered the following line:

```
COMMANDS:  K W B ↵
KEY: 021 ↵
```

In this case, the system would have written key 021 into the main index (level 0), because you didn't include the motion control specifier for static relative motion and INQUIRE assumed keyed access alone.

The only other meaningful motion specifiers for a keyed access are ↑ (for upward motion) and ↓ (for downward motion). For example, if you've just finished defining a subindex and want to write a key into it, specify the following:

```
COMMANDS:  K W ↓ ↵
KEY: keyname ↵
```

Note that INQUIRE displays your command as

```
COMMANDS: K W DOWN
```

Remember that a command string containing the W command cannot use relative motion alone. A keyed WRITE (K W) or a WRITE (W) command can use keyed motion alone, or combined keyed and relative motion. In this latter case, you would specify your relative motion on the *COMMANDS:* line, and define your key (from your resulting position) on the KEY: line.

INQUIRE's Command Modifiers

INQUIRE's command modifiers permit you to select certain options when you use INQUIRE's function commands. This section explains all the command modifiers.

Suppress Database (B)

The B command modifier instructs INQUIRE to suppress the database during the specified command. This prevents an INFOS II system error from occurring if someone has deleted or locked the database record. It also prevents you from overwriting an existing record when performing an inverted Write request (see the Invert modifier below). The B command modifier is also useful if you want to write a selector key, which has no data record.

Invert (I)

Use the I command modifier to attach more than one key to the same database record. When you use this modifier, specify the key you want to invert (the alternate key). INQUIRE will then ask for the octal position code of the data record associated with the original key. You can find this code with the ACQUIRE FEEDBACK (A) command, discussed later in this chapter. Be sure to specify Suppress Database when you create an inverted key. Otherwise, INQUIRE will expect you to rewrite the database record, and it will destroy the old contents of the record.

Lock (M)

The M command modifier locks the record you are using. No other user will be able to gain access to the record while you are working on it. See Chapter 3 for more information on locks. Remember to use the Unlock command (T) on the record when you finish, so that others can use it.

In the following example, INQUIRE reads (R) the record for the first (lowest value) key in the main index, locks it (M), and includes the partial record (P) in the read.

```
COMMANDS:  R M P \ ↵
```

INQUIRE will then ask you whether you want to lock the partial record, the data record, or both. You will receive these prompts:

```
DATABASE RECORD LOCK? (Y OR [N]):
PARTIAL RECORD LOCK? (Y OR [N]):
```

Respond with a Y for either or both records, depending on which you want to lock. Enter N or press NEW LINE if you do not want to lock the record.

Unlock (T)

Use the T command modifier to unlock the record on which you are currently positioned. Unlock all locked records when you are finished with them. If you don't, you could prevent other database users from completing their work.

No Position Change (N)

The N command modifier maintains your position in the index file at your current location, regardless of the results of your command.

Include Partial Record (P)

Use the P command modifier to gain access to a partial record in addition to (or instead of) the database record. INQUIRE will not display the contents of the partial record if you do not use the P modifier.

Define Merit Factor (Z)

Use the Z command modifier to assign a merit factor to a record. The merit factor is meaningful only if you use optimized record distribution (see Chapter 10). When you use the Z command modifier, the system places your record on the volume whose merit factor is equal to or less than the record merit factor you specify. If you want to assign or reassign a merit factor to an existing record, you must rewrite the record.

INQUIRE's Function Commands

The leftmost column of Table 6-1 lists INQUIRE's function commands. These commands perform the same processing functions as the INFOS II commands described in Chapter 3, except that INQUIRE has no OPEN or CLOSE equivalents. You open the file by invoking INQUIRE and close the file by pressing the ESC key.

This section describes each of these function commands and indicates the command modifiers and motion control specifiers appropriate for each command.

Read Command (R)

Read is the default function command. INQUIRE will perform a read operation if you do not include a function command in the command string, but do include one or more command modifiers or a motion control specifier. For example, if your current position is in a lower level subindex and you want to examine the partial record for the key that owns that subindex, you can specify the following:

```
COMMANDS:  P ↑ ↓
```

When you respond to the *COMMANDS:* prompt with either R or NEW LINE, INQUIRE performs a read with down and forward motion. You can use any combination of the following command modifiers with the R command: B, K, N, P, and either M or T (not both). If you use the modifier K, the only allowable motion control specifiers will be @, ↑, or ↓. If you don't include K in the command string, you can use any of the motion control specifiers. You can also specify any combination of the INQUIRE commands A, Y, or # with the R command.

Write Command (W)

Use the Write command (W) to enter a new key into the main index or a subindex, with or without a partial or data record. The new key can be a duplicate if you have allowed duplicate keys in that index. The W command will enter a new data record into the database, unless you have included the Suppress Database command modifier (B) in your command string. You must include the P command modifier if you want to write a partial record.

You can specify any combination of the following command modifiers: B, I, K, N, P, Z, and either M or T (not both). We describe the Invert (I) command modifier further in the section "Linking Keys to Existing Data Records." You can also specify any combination of the INQUIRE commands A, #, and Y with the W command. Unless you include a relative motion specifier in the command string, INQUIRE assumes keyed access on a Write operation.

Write Command Examples

1. In this example, we use the W command to write key 01 into the main index and RECORD ONE into the database.

```
COMMANDS:    K W ⤸
KEY:    01 ⤸

RECORD
FIELD # 1:    RECORD ONE ⤸
```

   (We will describe record formats under the unique INQUIRE command Set Formats (F).)

2. In this example, we illustrate relative motion. We will write into a lower level subindex from directly above, using down and forward motion.

```
COMMANDS:    W \ ⤸
```

   (Note that this line appears on your screen as the following:)

```
COMMANDS:    W DOWN-FORWARD
KEY:    HARRIS ⤸
FIELD # 1:    G(TB)**2 ⤸
```

3. If the current position is in a lower level subindex, you can use the @ motion control specifier to write a key there. For example:

```
COMMANDS:    K W P @ ⤸
KEY:    11 ⤸

RECORD
FIELD # 1:    RECORD TWO ⤸
PARTIAL RECORD:    SUBINDEX LEVEL 01 ⤸
```

   This sequence writes the key 11, the record RECORD TWO, and the partial record SUBINDEX LEVEL 01, to the subindex in which current position is set. It is important to note that without the static motion control specifier, @, the system would write the key and partial record to the main index (level 0 subindex).

Rewrite Command (X)

Use the Rewrite command (X) to modify a data record or partial record. You can use any combination of the following command modifiers: B, I, K, N, P, Z, and either M or T (not both). We describe the Invert (I) command modifier later under the section "Linking Keys to Existing Data Records." If the command string contains K, the only allowable motion control specifiers will be ↑, ↓, and @. If you do not include K in the command string, you can use any of the motion control specifiers. You can also specify any combination of the INQUIRE commands A, Y, and #.

Rewrite Command Examples

1. A simple Rewrite command could look like the following:

```
COMMANDS:    X ⤸

RECORD
FIELD # 1:    RECORD 1 ⤸
```

2. In this example, we rewrite a partial record, but not its associated data record.

```
COMMANDS:    X B P ⤸

PARTIAL RECORD:    PART REC ONE ⤸
```

Linking Keys to Existing Data Records

When you write a new key you can link it to an existing data record with an inverted Write command (K W I). You can also link an existing key to a data record with an inverted Rewrite command (X I). However, before you issue an inverted Rewrite command, the key cannot already be linked to a data record. During these linking operations, you also have the option of modifying the contents of the data record, since it already exists. That is, you may rewrite the record on either a Write Inverted or on a Rewrite Inverted operation. Note that if you don't want to rewrite the record, you must include the Suppress Database command modifier (B) in the command string.

The inverting procedure requires that you supply the INFOS II system with two 6-digit octal numbers, referred to as the data record feedback. INQUIRE provides the Acquire Feedback command (A) so that you can retrieve the feedback to the data record you want. (See "Unique INQUIRE Commands," later in this chapter, for more information on the A command.)

Inverting Example 1

Figure 6-5 shows a simple single-level file with key 001 linked to the record, Record One. Let's assume you've just opened the file, so the current position is above the index. You want to write the key 002, linking it to the data record, Record One. You will then write another key, 003, and rewrite the data record at the same time.

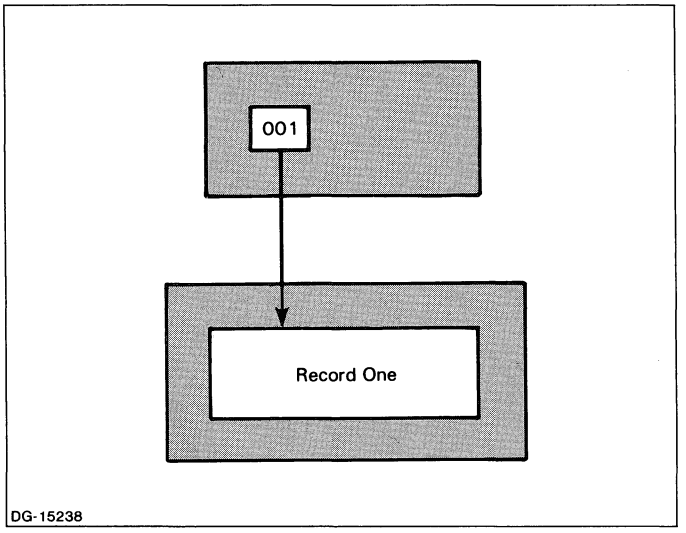


Figure 6-5. Simple INFOS II File–I

First, you have to obtain the octal link between key 001 and the data record. To do this, you must gain access to key 001 and specify the INQUIRE command A.

```
COMMANDS:  K A ↵
KEY:  001 ↵
```

INQUIRE returns the data record and its feedback information:

```
FIELD #1: Record One
DATA RECORD LINK (OCTAL): 000400/000002
INDEX LEVEL: 0
```

Now you can issue the Write Inverted command. Let's assume you do not want to modify the data record right now. Suppress the database with the B command modifier.

```
COMMANDS:    K W I B ⤶
KEY:    002 ⤶
FEEDBACK HIGH WORD (OCTAL):    400 ⤶
FEEDBACK LOW WORD (OCTAL):    2 ⤶
```

The file now looks like the one in Figure 6-6.

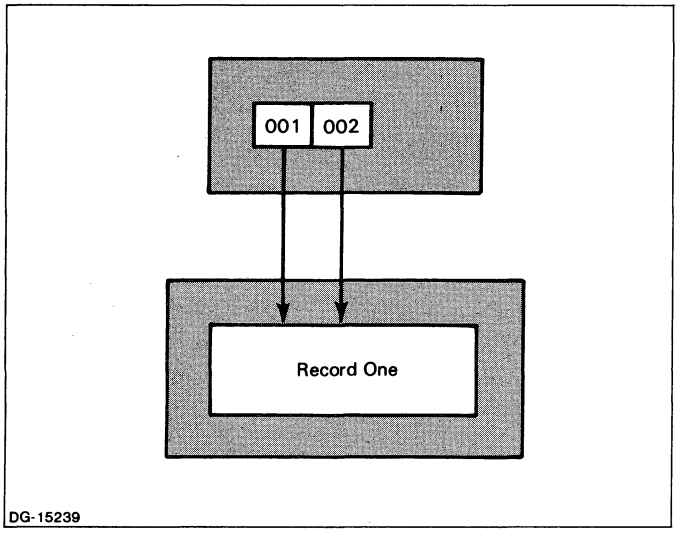


Figure 6-6. Simple INFOS II File–II

Next you want to write key 003. Suppose you also want to rewrite the data record now. You will not use the suppress database command modifier, B, this time.

```
COMMANDS:    K W I ⤶
KEY:    003 ⤶

FIELD #1:    First Record ⤶
FEEDBACK HIGH WORD (OCTAL):    400 ⤶
FEEDBACK LOW WORD (OCTAL):    2 ⤶
```

The file now looks like the one shown in Figure 6-7.

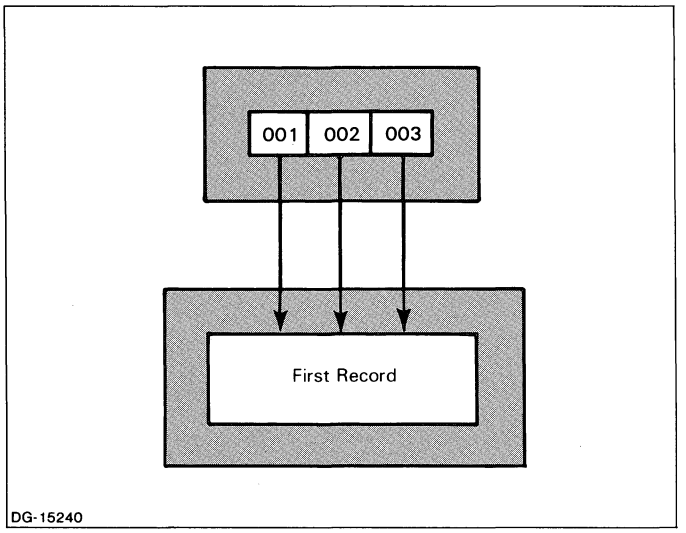


Figure 6-7. Simple INFOS II File–III

Inverting Example 2

In this example, we will link an existing key to a data record. We will then link another existing key to the data record and rewrite the data record at the same time. Figure 6-8 illustrates the file we will use.

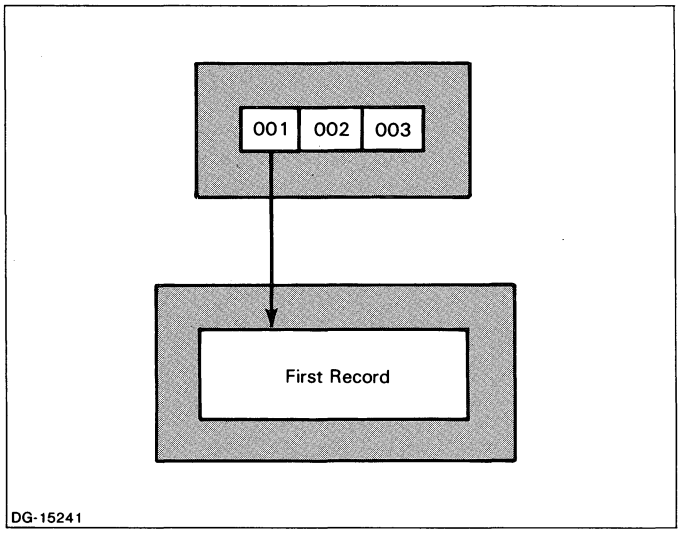


Figure 6-8. Simple INFOS II File–IV

First, we will link key 002 to the data record. After we obtain the data record feedback, we can issue a Rewrite command with inversion. Here again, we will use the suppress database command modifier, B, in the command string, since we don't want to modify the data record at this time.

```
COMMANDS:    K X I B ↵
KEY:    002 ↵
FEEDBACK HIGH WORD (OCTAL):    400 ↵
FEEDBACK LOW WORD (OCTAL):    2 ↵
```

The file now looks like the one shown in Figure 6-9.

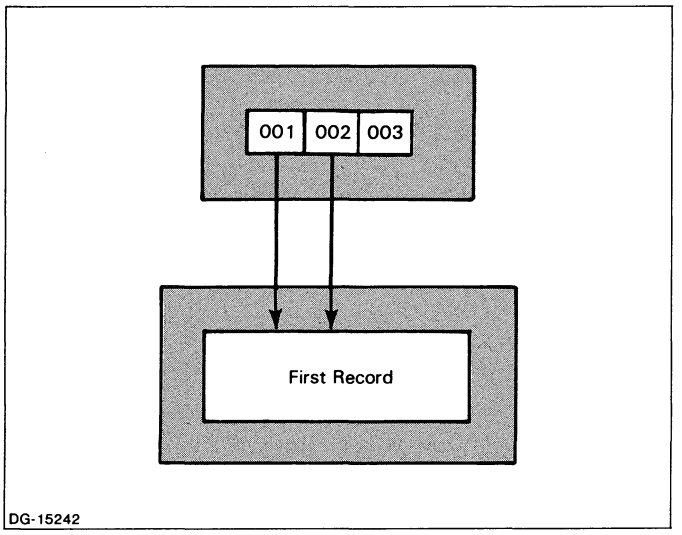


Figure 6-9. Simple INFOS II File–V

To link key 003 to the data record and rewrite the record at the same time, we will issue a Rewrite command with inversion.

```
COMMANDS:    K X I ↵
KEY:    003 ↵

FIELD #1:    Rewritten Record ↵
FEEDBACK HIGH WORD (OCTAL):    400 ↵
FEEDBACK LOW WORD (OCTAL): 2 ↵
```

The file now looks like the one in Figure 6-10.

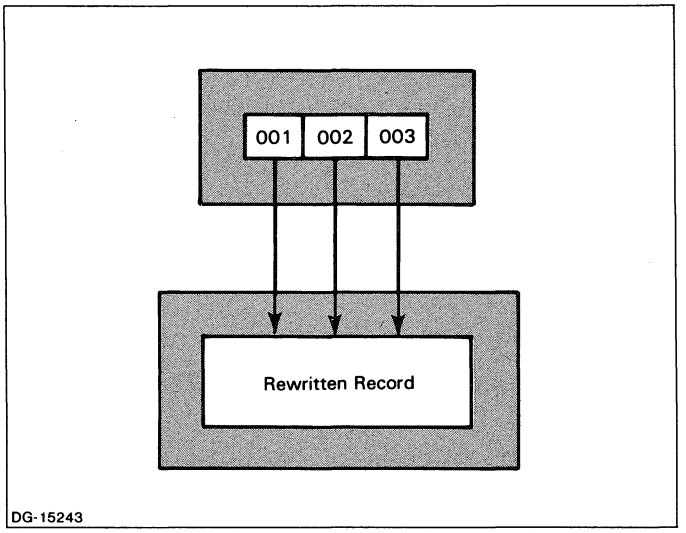


Figure 6-10. Simple INFOS II File–VI

Entering Duplicate Keys with the W Command

Using the INQUIRE W command, you can make a duplicate for any key by specifying the key name followed by the # command modifier. As we explained in Chapter 4, the INFOS II system uses occurrence numbers to differentiate between duplicate keys. You need to specify a duplicate key's occurrence number to access it.

For example, let's look at a typical dialog that creates duplicate keys. You create the first key in an ordinary manner, without using the # modifier.

```
COMMANDS:   K W ⤸
KEY:   LIDIA ⤸
FIELD # 1:   EPSILON PRODUCTS INC ⤸
```

When you specify the second key, you must use the # modifier to indicate that the key is a duplicate; otherwise, you will get an error.

```
COMMANDS:   W ⤸
KEY:   LIDIA# ⤸
FIELD # 1:   LAMBDA MANUFACTURING ⤸
```

(Note that K W and W can be used interchangeably here).

This dialog creates two keys — both named LIDIA — that name two different data records, EPSILON PRODUCTS INC and LAMBDA MANUFACTURING.

When you ask INQUIRE to read the first record, it returns the following:

```
COMMANDS:   R ⤸
FIELD # 1:   EPSILON PRODUCTS INC
DUPLICATE KEY EXISTS
```

Using the V command to retrieve the key, you will see the following:

```
COMMANDS:    V ↵
RETRIEVED KEY:    LIDIA
OCCURRENCE NUMBER: 000000/000001
DUPLICATE KEY EXISTS
```

Notice that INQUIRE now shows us LIDIA's occurrence number. (We describe the V command in more detail in a later section.)

Assuming you already know all the duplicate keys' occurrence numbers (described in the next section), you have the following dialog to read and retrieve the second key.

```
COMMANDS:    V ↵
RETRIEVED KEY:    LIDIA
OCCURRENCE NUMBER: 000000/000002
DUPLICATE KEY EXISTS

COMMANDS:    K R ↵
KEY:    LIDIA#2 ↵

FIELD # 1: LAMBDA MANUFACTURING
DUPLICATE KEY EXISTS
```

The next section describes in more detail what these occurrence numbers are, and how they work.

Occurrence Numbers

Occurrence numbers are unique 12-digit octal numbers that the INFOS II system associates with every key name. Remember that the system automatically assigns consecutive occurrence numbers to every key within the same subindex as it creates them. When a key becomes a duplicate, these previously unknown numbers become a logical part of the key name. If you then want to specify a duplicate key for an INQUIRE request, you must identify the key with its name *and* its unique occurrence number.

You can ask INQUIRE to return a key's occurrence number to you with the Retrieve Key (V) command. When using occurrence numbers, the following rules apply:

1. Unlike ordinary key names, you need not specify the entire 12 digits of a duplicate key's occurrence number. You may omit any number of the leading zeros. When you specify the occurrence number, you must omit the slash that INQUIRE places between the number's sixth and seventh digits. INQUIRE uses this to mark the separation between the high and low words that comprise the number.
2. If you do not specify an occurrence number, or if you specify occurrence #0, INQUIRE will find the first occurrence of that key.

   You can also use relative motion to retrieve duplicate keys. To find duplicate keys in a sequence, first use occurrence number 0, then use relative motion (forward) to find the occurrence numbers of the other duplicate keys. To illustrate the use of occurrence number 0, we will use the example above. First we will perform a keyed Retrieve Key and then a Retrieve Key with forward motion.

   ```
   COMMANDS:    K V ↵
   KEY:    LIDIA#0 ↵
   RETRIEVED KEY: LIDIA
   OCCURRENCE NUMBER: 000000/000001
   DUPLICATE KEY EXISTS

   COMMANDS:    V → ↵
   RETRIEVED KEY: LIDIA
   OCCURRENCE NUMBER: 000000/000002
   DUPLICATE KEY EXISTS
   ```

3. The INFOS II system never reuses an occurrence number in the same subindex. However, since it assigns occurrence numbers on a subindex basis, keys in different subindexes can have the same occurrence number. As a result, it is possible to create two sets of duplicate keys with exactly the same names and occurrence numbers. Because these keys are in different subindexes, however, they will have different pathnames. The following examples show some valid and invalid key specifications, and the operation(s) in which they apply.

Operation	Valid Statements	Invalid Statements
READ	*KEY:* LIDIA#1	*KEY:* #1 (no key name)
READ	*KEY:* LIDIA#0106	*KEY:* LIDIA0106 (no # sign)
All	*KEY:* LIDIA#	
All	*KEY:* LIDIA	

Delete Command (D)

Use the Delete command, D, to physically delete a key and its associated data record and partial record. Or you can use D to mark a partial or data record as logically deleted. The default is physical deletion. See Chapter 3 for more information about physical and logical deletion. INQUIRE uses K as the default command modifier on delete operations.

Physical Deletion

For a physical deletion, you simply tell INQUIRE which key you want to delete. Since physical deletion is the default type of deletion, you can respond to the *LOGICAL DELETION (Y OR [N]):* prompt by pressing NEW LINE. For example:

```
COMMANDS:  D ↵
LOGICAL DELETION (Y OR [N]):  ↵
KEY:  123 ↵
```

INQUIRE first locates the index entry containing the specified key (123) and sets the current position on that entry. It then unlinks the key from its associated data record, if one exists. If unlinking the key reduces the record's use count to 0, the record is deleted. INQUIRE then deletes the key if it is not linked to a subindex.

If INQUIRE deletes the key, it will set the current position on the next lower value key in the subindex, if there is one. If the key that is deleted is the lowest key in the main index, INQUIRE will set the current position in front of the main index. If the key that is deleted is the lowest key in a lower level subindex, INQUIRE will set the current position in front of that subindex.

On a physical deletion, you can include the No Position Change (N) command modifier in the command string. If you do, then INQUIRE will not set the current position according to the key you delete. After the deletion, your current position will be the same as it was prior to the delete operation. For example:

```
COMMANDS:  D N ↵
LOGICAL DELETION (Y or [N]):  ↵
KEY:  123 ↵
```

Logical Deletion

On a logical deletion, INQUIRE asks what you want to mark as logically deleted. Your dialog would look like this:

```
COMMANDS:    D ↵
LOGICAL DELETION (Y or [N]):    Y ↵
KEY:    123 ↵
```

For logical deletions, you can use any combination of the following command modifiers: B, K, N, and P, and either M or T (not both). INQUIRE uses K as the default modifier. With K, the allowable motion control specifiers are ↑, ↓, and @. If you don't specify K, you can use any of the motion control specifiers: ↑, ↓, ←, →, @, +, -, or \.

Deleting Duplicate Keys with the D Command

You can use the D command to physically or logically delete a duplicate key, its partial record, and its data record. Deleting duplicate keys is a fairly straightforward process. Let's look at our LIDIA example again. Assume you have created a third key, LIDIA#3, which contains the record CHANCELLOR CO. Now you want to logically delete the partial record and data record associated with LIDIA#1. The dialog for this deletion looks like this:

```
COMMANDS:    K D ↵
LOGICAL DELETION (Y OR [N]):    Y ↵
KEY:    LIDIA ↵
LOCAL (PARTIAL RECORD) REINSTATE/DELETE? (Y OR [N]):    Y ↵
GLOBAL (DATABASE RECORD) REINSTATE/DELETE? (Y OR [N]):    Y ↵

COMMANDS:    R ↵
FIELD # 1: EPSILON PRODUCTS INC
PARTIAL RECORD LOGICALLY DELETED (LOCAL)
DATABASE RECORD LOGICALLY DELETED (GLOBAL)
DUPLICATE KEY EXISTS

COMMANDS:    V ↵
RETRIEVED KEY: LIDIA
OCCURRENCE NUMBER: 000000/000001
PARTIAL RECORD LOGICALLY DELETED (LOCAL)
DUPLICATE KEY EXISTS
```

In this case, the key's occurrence number still exists because we deleted the key logically instead of physically. Reading LIDIA#0 will return the field EPSILON PRODUCTS INC with the same messages as the R command above.

Now, let's physically delete this key:

```
COMMANDS:    K D ↵
LOGICAL DELETION (Y OR [N]):    ↵
KEY:    LIDIA ↵
```

Reading LIDIA#0 now returns this dialog:

```
COMMANDS:    K R ↵
KEY:    LIDIA ↵
FIELD # 1: LAMBDA MANUFACTURING
DUPLICATE KEY EXISTS
```

If we had not created LIDIA#3 earlier, this physical deletion would leave only one LIDIA key remaining. INQUIRE would then drop all references to duplicate keys when we subsequently reference LIDIA, as shown here:

```
COMMANDS:    K R ↵
KEY:    LIDIA#0 ↵
FIELD # 1: LAMBDA MANUFACTURING
```

Note that specifying #0 has no effect when a key has no duplicates. The #0 modifier is merely a place holder. While you never need to specify this modifier, using #0 can make your code easier to follow; for example, in the case of a stored input file, where someone else will execute previously written commands. INQUIRE accepts occurrence number 0 for any key you specify on any command. It provides 0 itself if you do not indicate an occurrence number.

Reinstate Command (Q)

Use the Reinstate command, Q, to remove the logically deleted marks from partial records and data records. In the following example, we use Q to remove the logically deleted marks on the partial record and data record at the current position.

```
COMMANDS:    Q ↵
LOCAL (PARTIAL RECORD) REINSTATE/DELETE? (Y OR [N]):    Y ↵
GLOBAL (DATABASE RECORD) REINSTATE/DELETE? (Y OR [N]):    Y↵
```

You can include any combination of the modifiers B, K, N, and P, and either M or T (not both) in the command string. If you use K, then the allowable motion control specifiers are ↑, ↓, and @. If you do not specify K, you can use any one of the motion control specifiers.

Define Subindex Command (S)

Use the Define Subindex command, S, to specify parameter values for a subindex. You can use the S command only when the current position is set on a key that can be linked to a subindex. You can use keyed, relative, or the combination of keyed and relative motion to access the desired key. You may also use either of the command modifiers M or T in conjunction with the S command.

INQUIRE prompts you for the following subindex parameter values:

```
ROOT NODE SIZE:
MAXIMUM KEY LENGTH: [255]:
PARTIAL RECORD LENGTH: [0]:
SUBINDICES ALLOWED? (Y OR [N]):
ALLOW DUPLICATE KEYS IN THIS SUBINDEX? (Y OR [N]):
```

The default value for root node size depends on your page size. (This is the page size you specified when you first created the INFOS II file.) When you have pages of 2048 bytes, the default is 2042. When you have the larger page size, 4096 bytes, your default root node size is 4088. If you do not want the default value, you must specify a size (in bytes) that is large enough to contain at least three maximum length index entries, but is not greater than 6 bytes less than page size. The default is the maximum size you can specify.

INQUIRE remembers the parameter values you use with the S command. If you subsequently issue another S command, INQUIRE will ask if you want to use the parameter values of the previously defined subindex.

```
COMMANDS:    S ↵
USE PREVIOUS INFO? (Y OR [N]):
```

If you respond with a Y, INQUIRE uses the parameter values for the previously defined subindex. If you answer with N or NEW LINE, INQUIRE prompts you for new subindex parameter values.

Link Subindex Command (L)

Use the Link Subindex command, L, to link an existing key to an existing subindex. The key must not currently be linked to a subindex.

Immediately before issuing the L command, you must gain access to a key that is linked to the subindex to which you want to link another key. You must also set the current position on it.

This key you access is the source key. The source key may be the key under which the subindex was originally defined, or it can be a key that you linked to the subindex with a previous link subindex request.

When you issue the L command, you must specify the information that the INFOS II system needs to locate the key that will be linked to the subindex. This key is the destination key. You do this by responding to the prompts INQUIRE generates. For example:

```
COMMANDS:    K B R ⤸
KEY:    JRS ⤸

KEY HAS SUBINDEX

COMMANDS:    L ⤸
DESTINATION KEY:    TJR ⤸
DESTINATION CCW-MUST SPECIFY POSITIONING (OCTAL):    xxxxxx ⤸
```

The first part of the command is a keyed read to the source key (with database record suppression), where INQUIRE automatically sets the current position. INQUIRE then tells you that the accessed key is linked to a subindex.

The second part of the command, L, brings up the *DESTINATION KEY:* prompt. Specify a complete key path to the destination key, separating each key in the path by a valid between-key separator. (Recall that the comma is the default between-key separator.) If the destination key is in level 0, you need one key; if it's in level 1, you need two keys, etc. In our example above, we chose a key in the main index for our destination key.

After you specify the destination key, INQUIRE prompts you for the destination Command Control Word (CCW). This tells the INFOS II system how to position the destination key. Table 6-2 shows the octal numbers you can select for the destination CCW, which we indicated with xxxxxx above.

To combine keyed and relative access to the destination key, you must use 1 as the first digit of the six-digit octal number. For relative motion alone, the first digit will be 0.

Table 6-2. Destination CCW - Octal Numbers

Destination Access Type	Leave Current Position on Source Key		Set Current Position on Destination Key	
	Relative Motion (No Destination Key)	Keyed/ Relative	Relative Motion (No Destination Key)	Keyed/ Relative
(Keyed Only)	XXXX	100000*	XXXX	102000*
Forward Relative	040000	140000	042000	142000
Backward Relative	044000	144000	046000	146000
Downward Relative		150000		152000
Down/Forward Relative	054000	154000	056000	156000
Up/Forward Relative	060000	160000	062000	162000
Up/Backward Relative	064000	164000	066000	166000
Upward Relative		170000		172000
Static Relative		174000		176000

*No Relative Motion

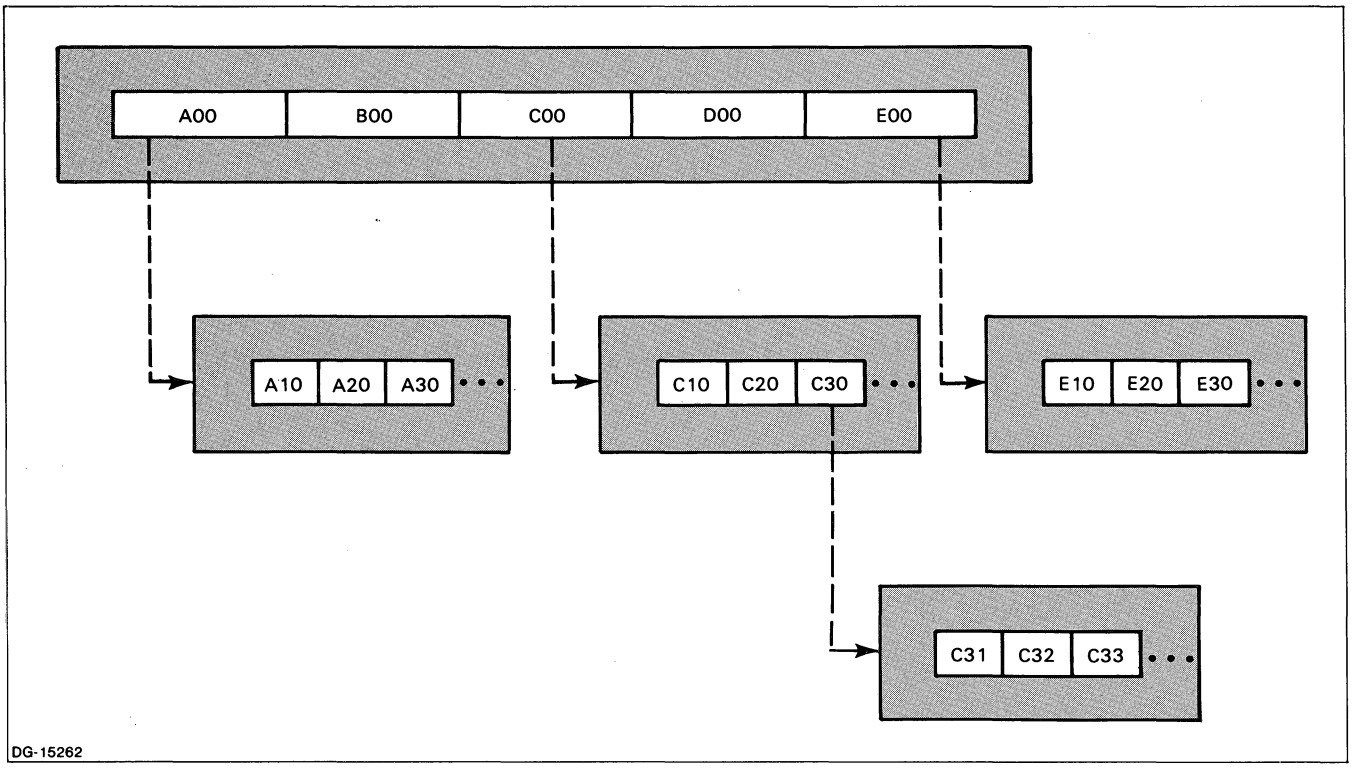


Figure 6-11. Sample Index–I

Link Subindex Examples

The following examples are based on Figure 6-11. They will help to clarify the use of the Destination CCW.

1. Let's assume you want to link the key B00 to the subindex below the key A00. First, gain access to key A00:

```
COMMANDS:    K B R ⤸
KEY:    A00 ⤸

KEY HAS SUBINDEX
```

   Then enter the LINK SUBINDEX command.

```
COMMANDS:    L ⤸
DESTINATION KEY:    B00 ⤸
DESTINATION CCW - MUST SPECIFY POSITIONING (OCTAL):    042000 ⤸
```

   The INFOS II system sets the current position on the source key (A00) when you gain access to it. The Destination CCW specifies forward motion from the current position (from key A00 to key B00). It then sets the current position on B00.

2. Let's assume you want to link the key D00 to the subindex below the key C30. First, gain access to key C30:

```
COMMANDS:   K B R ↵
KEY:   C00,C30 ↵
```

Then enter the LINK SUBINDEX command, specifying up and forward relative motion.

```
COMMANDS:   L ↵
DESTINATION KEY:   D00 ↵
DESTINATION CCW - MUST SPECIFY POSITIONING (OCTAL):   060000 ↵
```

3. Suppose you want to link key C10 to the subindex below key C30. First, gain access to key C30. Then, after entering the L command, specify the combination of keyed and static relative motion as the Destination CCW.

```
COMMANDS:   K ↵
KEY:   C00,C30 ↵

KEY HAS SUBINDEX

COMMANDS:   L ↵
DESTINATION KEY:   C10 ↵
DESTINATION CCW - MUST SPECIFY POSITIONING (OCTAL):   174000 ↵
```

Note that the first digit, 1, specifies keyed access. The next digits, 7 and 4, specify static relative motion, without setting the current position. If you had specified 176, you would have defined keyed access, static relative motion, and Set Current Position (see Table 6-2).

4. To link the key E30 to the subindex below key C30, first gain access to key C30. Then, after entering the L command, specify keyed access as the Destination CCW.

```
COMMANDS:   K ↵
KEY:   C00,C30 ↵

KEY HAS SUBINDEX

COMMANDS:   L ↵
DESTINATION KEY:   E00,E30 ↵
DESTINATION CCW - MUST SPECIFY POSITIONING (OCTAL):   100000 ↵
```

Delete Subindex Command (U)

Use the Delete Subindex command, U, to unlink a subindex from a key or to physically delete a subindex. To use the U command, you must do one of the following:

- Have the current position set on a key linked to a subindex.
- Move to a key that is linked to a subindex by including a command modifier in the command string along with the U command.

If another key in the index is linked to the subindex, the key you access for the U command will be unlinked from the subindex, but the subindex will not be physically deleted. The system will not physically delete the subindex if any of the following conditions are true:

- Any other key in the index is linked to it.
- Another user has a current position in that subindex.
- Any records (partial or data) that are associated with the keys in that subindex are locked.

If either of the last two conditions exists when you issue the U command, you'll get an error message and INQUIRE will not unlink the subindex from the key.

When you physically delete a subindex, each data record associated with the keys in the subindex has its use count decremented by one. The INFOS II system physically deletes from the database any record whose use count is reduced to zero as a result of the Delete Subindex process.

The following examples are based on Figure 6-12, in which we detail a Delete Subindex operation.

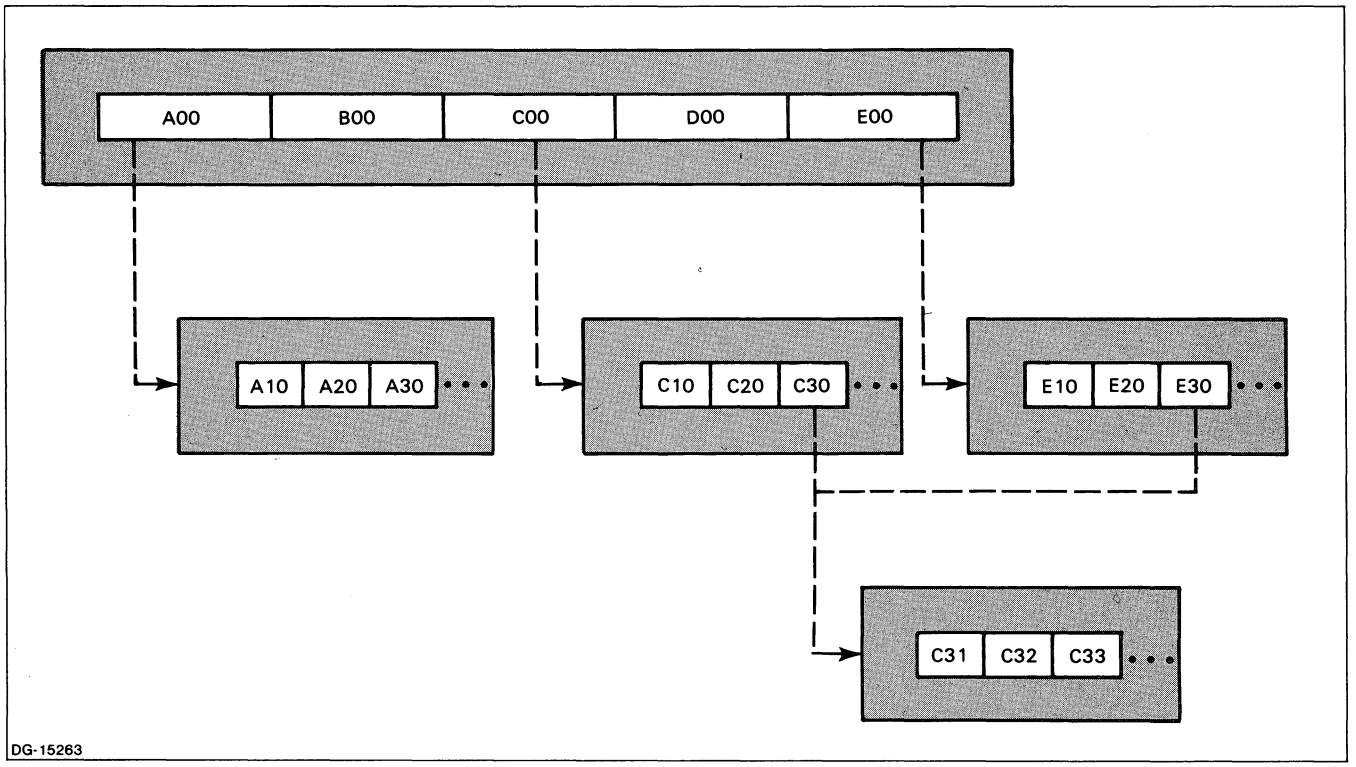


Figure 6-12. Sample Index–II

Delete Subindex Examples

The following examples are based on the sample index in Figure 6-12.

1. Let's assume that the current position is set on key E20 and you want to unlink key E30 from its subindex. Enter the following:

   ```
   COMMANDS:  U →⤸
   ```

2. If the current position is on key E30 and you want to unlink the key C30 from its subindex (which could physically delete that subindex), enter:

   ```
   COMMANDS:  U K⤸
   KEY:  C00,C30⤸
   ```

3. If the current position is on key A00, enter the following to unlink A00 from its subindex:

   ```
   COMMANDS:  U⤸
   ```

Retrieve Status Command (E)

Use the Retrieve Status command, E, to retrieve information about a key and its associated data record. You can use the command modifiers K and N, and either M or T. The command modifier B returns the record's length as 0 and the modifier P has no effect.

You can also use any of the motion control specifiers to gain access to a key. If you don't include a motion control specifier in the command string, you will receive information for the key at your current position. After the command executes, the current position will be on the key you specified for the E command, unless you included the N command modifier in the command string.

The status information that INQUIRE returns on an E command includes all relevant portions of the following:

```
COMMANDS:   E ↵

PARTIAL RECORD LOGICALLY DELETED (LOCAL)
DATABASE RECORD LOGICALLY DELETED (GLOBAL)
DUPLICATE KEY EXISTS
KEY HAS SUBINDEX
RECORD LENGTH:nnnn
VOLUME MERIT FACTOR:nn
KEY LENGTH:nnn
PARTIAL RECORD LENGTH:nnn
OCCURRENCE NUMBER: nnnnnn/nnnnnn
```

INQUIRE expresses record and key length in bytes. Partial record length is also given in bytes, but the number returned reflects the maximum partial record length allowed in the subindex.

Retrieve Key Command (V)

Use the Retrieve Key command, V, to find the value of a key.

When V is the only entry in the command string, it returns the value of the key at your current position. If you include the command modifier K or any of the motion control specifiers in the command string along with the V command, it will return the value of the key it reaches after the positioning. Your current position after execution of the V command will be on that key, unless you also included the N command modifier in the command string.

The V command returns all relevant parts of the following information:

```
COMMANDS:   V ↵

RETRIEVED KEY: keyname
OCCURRENCE NUMBER: nnnnnn/nnnnnn
PARTIAL RECORD LOGICALLY DELETED (LOCAL)
DUPLICATE KEY EXISTS
KEY HAS SUBINDEX
```

Retrieve High Key Command (H)

Use the Retrieve High Key command, H, to find the value of the highest key in a subindex.

When H is the only entry in the command string, it returns the value of the highest key in the current subindex and sets the current position to that key. If you include the K command modifier or any of the motion control specifiers in the command string along with the H command, INQUIRE will return the highest key in the current subindex that you directed it to. It will also reset the current position on the accessed key, unless you included the N command modifier in your command string.

The H command returns the following information:

```
COMMANDS:    H ⤸

HIGH KEY: keyname
OCCURRENCE NUMBER: nnnnnn/nnnnnn
```

Retrieve Subindex Definition Command (G)

Use the Retrieve Subindex Definition command, G, to examine the parameter values of a subindex.

When G is the only entry in the command string, INQUIRE returns the parameter values for the current subindex and does not change your current position. If you include the command modifier, K, or any of the motion control specifiers in the command string along with the G command, it will return the parameter values for the specified subindex. It will also set your current position, unless you specified the N command modifier in your command string.

The subindex parameter values returned for the G command follow:

```
COMMANDS:    G ⤸

ROOT NODE SIZE:nnnn
VOLUME MERIT FACTOR:nn
MAXIMUM KEY LENGTH:nnn
PARTIAL RECORD LENGTH:nnn
SUBINDICES ARE ALLOWED
DUPLICATES ALLOWED IN THIS INDEX
```

Note that if this subindex does not allow duplicate keys or subindexing, the corresponding parameters will be

```
SUBINDICES NOT ALLOWED
DUPLICATES NOT ALLOWED IN THIS INDEX
```

Also note that the *VOLUME MERIT FACTOR* will only appear if you are using optimized record distribution.

Close Command (ESC)

Use the Close command, ESC, to close the current file. When you press the ESC key in response to the *COMMANDS:* prompt, INQUIRE closes the file and asks for another index filename.

```
COMMANDS:    ESC ⤸
INDEX:
```

You may respond to the *INDEX:* prompt with either of the following:

- An index filename, in which case INQUIRE will open the named file.
- A NEW LINE (⤸), in which case INQUIRE will return you to the CLI.

Unique INQUIRE Commands

The rightmost column of Table 6-1, appearing earlier in this chapter, shows the unique INQUIRE commands. Of these commands, the following have special characteristics.

C List Commands

F Set Formats

J Change Input File

O Change Output File

The system executes these commands immediately after you press the appropriate key, so do not press CR or NEW LINE or any other characters afterwards. Otherwise, you could make a mistake. For example, if you type F and press NEW LINE, INQUIRE will immediately start the Set Formats dialog upon receiving the F, and will take the NEW LINE as a default response to the next prompt, which is *FORMATS IN FILE? (Y OR [N])*. Thus, even if the format you want *is* in the file, INQUIRE will assume it isn't, because you answered [N], and you will have to go through the entire create-a-new-format dialog. You can nullify the F command by pressing NEW LINE for every prompt, but you cannot terminate the question sequence.

You also cannot abort a C command once you type it, but that is no problem. The list of commands will merely appear on your screen, followed by another *COMMANDS:* prompt. To interrupt the J and O commands, press NEW LINE when you receive the first prompt.

We described the following unique INQUIRE commands earlier in this chapter — the C command in the "How to Invoke INQUIRE" section, and the NEW LINE command in the "Overview of INQUIRE Commands" section.

Acquire Feedback Command (A)

The Acquire Feedback command (A) provides information that you will need if you want to perform inversion. (See the "Linking Keys to Existing Data Records" section, earlier in this chapter.) The information includes the fields of the data record, the index level, and two six-digit octal numbers. You use these octal numbers for inversion.

If you use the A command on a command line by itself, you will receive feedback information for the key on which the current position is currently set. You can also use the A command with the command modifier K and the motion control specifiers. For example:

```
COMMANDS:   K A ⤸
KEY:   TJR ⤸

FIELD #1: TJR RECORD
DATA RECORD LINK (OCTAL): 000600/000002
INDEX LEVEL: 0
```

Set Formats Command (F)

Use the Set Formats command, F, to change the default format conventions for the contents of INFOS II files. Normally, INQUIRE interprets the contents of keys, partial records, and data records as ASCII characters. It interprets data records as ASCII characters with one field of 80 or fewer characters. If the file you want to examine with INQUIRE does not conform to these conventions, use the F command so that INQUIRE can properly retrieve and display the keys, partial records, and data records.

You need to define formats only once. If you want to save formats, INQUIRE will place your format specifications in a file you name. When you examine any file, you can specify the formats you want simply by naming the file containing the appropriate specifications. Set the formats according to the dialog shown in Table 6-3.

Table 6-3. INQUIRE Format Dialog

INQUIRE Prompt	Your Response	Result
FORMATS IN FILE? (Y OR [N])	Y	INQUIRE asks for the name of the file in which you have previously stored the formats you want.
	N	INQUIRE will prompt you to specify key or record formats.
NUMBER OF LEVELS TO BE DESCRIBED: [1]:	number	Enter the number of index levels in which you want to set formats.
LEVEL # n		INQUIRE displays this message once for each index level you requested. Each time this message appears, you define key, partial record, and data record formats for an index level.
KEY		
NUMERIC? (Y OR [N]):	N	The keys for this index level consist of ASCII characters, so INQUIRE doesn't ask you the next two questions.
	Y	INQUIRE responds with the next two questions.
RADIX (8, 10, OR 16):[8]	number	Enter the number for the desired radix.
BYTES (B) OR WORDS (W):[B]	B	Each byte in the key field is interpreted as a numeric value according to the radix you specified.
	W	Each word in the key field is interpreted as a numeric value according to the radix you specified.
PARTIAL RECORD		
NUMERIC? (Y OR [N]):	N	The partial records at this index level consist of ASCII characters, so INQUIRE doesn't ask you the next two questions.
	Y	INQUIRE responds with the next two questions.
BYTES (B) OR WORDS (W):[B]:	B	Each byte in the partial record field is interpreted as a numeric value according to the radix you specified.
	W	Each word in the partial record field is interpreted as a numeric value according to the radix you specified.

(continues)

Table 6-3. INQUIRE Format Dialog

INQUIRE Prompt	Your Response	Result
NUMBER OF FIELDS IN DATABASE RECORD:[1]:	number	Enter the decimal number that corresponds to the number of fields you want in the record. The number must be between 1 and 25. If you specify a number out of this range, you will be prompted again. You then define each field with the next two questions.
FIELD # n		INQUIRE displays this message before you define the format for each field in the record.
NUMERIC? (Y OR N):	N	This field consists of ASCII characters. INQUIRE will ask you the *STARTING CHARACTER POSITION* and *FIELD LENGTH* questions rather than the *RADIX* and *BYTES OR WORDS* questions.
	Y	The content of this field is numeric. You define it by answering the next four questions.
RADIX (8, 10, OR, 16):[8]:	number	Enter the number for the radix you want.
BYTES (B) OR WORDS (W):[B]:	B	Each byte in the field is interpreted as a numeric value according to the radix you specified.
	W	Each word in the field is interpreted as a numeric value according to the radix you specified.
STARTING CHARACTER POSITION (1-n):[1]	number	Enter the decimal number of the byte in the record at which this field begins.
FIELD LENGTH, BYTES:	number	Enter the length, in bytes, of this field.
SAVE FORMATS IN FILE? (Y OR N):	Y	You can save the key, partial record, and data record formats defined during the current execution of INQUIRE and use them in subsequent executions of INQUIRE.
	N	The formats you define are valid only for the current execution of INQUIRE, and you lose them when you dismiss INQUIRE or define new formats.
FILE NAME:	name	If you responded Y to the *SAVE FORMATS* question, INQUIRE will build an AOS file in which it will store the formats you specified. Enter a name for this file.

(concluded)

Change Input File Command (J)

The J command tells INQUIRE to accept commands from a file in the system instead of from your terminal. You must first use the CLI to enter your commands into a file. The last entry in the CLI file should be J@CONSOLE (no spaces). Then, invoke INQUIRE on your INFOS II file and type the J command (remembering not to press NEW LINE or CR). INQUIRE will then give you the INPUT FILE: prompt. Type in the name of your CLI file and the system will execute all its commands in order. Your J@CONSOLE command at the end of the CLI file returns control of INQUIRE from the system to your terminal upon completion.

Change Output File Command (O)

The output file is the file to which INQUIRE writes the commands and the results of INQUIRE operations. Normally, this is the @CONSOLE file. Use the O command to direct these results to a file other than the default. INQUIRE will then ask you to name a new output file, to which it will record all subsequent results of INQUIRE commands.

After you've used the O command, you can return the output to the terminal by typing a second O command. Then, when you're asked to name the output file, enter @CONSOLE.

Dump Packets Command (Y)

You should not use the Dump Packets command, Y, until you are familiar with the contents of Chapters 9 and 10. In Chapter 9, you will see that a packet is a group of logically related control parameters. Information passes between a user and the INFOS II system through packets.

If you include the Y command in the command string, INQUIRE will display the packet it sends to the INFOS II system for the requested function. It then displays the same packet with the information the system places in it, before it returns to INQUIRE.

This command displays the packets line by line, moving them up the screen quite rapidly. You can halt the display by typing CTRL-S, and can restart it by typing CTRL-Q. Alternatively, you can specify a trailfile when you invoke INQUIRE, to which it will write the lines of the packets. Later you can print the trailfile and examine the printed copy of the packets' contents.

The following example shows a processing packet and key table for a READ operation with forward relative motion. The first packet and table are what INQUIRE sends to the INFOS II system. The second packet and table contain information placed there by the system before returning to INQUIRE. Note that we indicated a trailfile upon invoking INQUIRE so we can look at a printed copy of the packet's contents later.

```
)INQUIRE/T=LO.TRAIL ⤸
INDEX:    TEST ⤸
NUMBER OF LOCKS:[0]:    ⤸
ENTER C TO GET LIST OF COMMANDS:

COMMANDS:    DOWN B ⤸

COMMANDS:    R ⟶ Y ⤸


TYPE     PROCESSING PACKET
PCHN     100335
PKPN     000001
PDAT
PLEN     000000
PPRA
PCCW     CFWD CREL CSCP CSPR
PECW
```

```
PLKH    000000
PLKL    000000
PSXL    000000
PSTU
PMKL    000000
RMF-PRL 000000
PSID    000000
PXPP    000000


TYPE    KEY DEFINITION PACKET
KTYP
KYLN    000036
KKYP
KDKH    000000
KDKL    000012
RESD    000014
RESD    000015

TYPE    PROCESSING PACKET
PCHN    100335
PKPN    000001
PDAT    SSSSSSSSSSSSSSSSSS
PLEN    000024
PPRA
PCCW    CFWD CREL CSCP CSPR
PECW
PLKH    035000
PLKL    000002
PSXL    000000
PSTU
PMKL    000003
RMF-PRL 000000
PSID    000000
PXPP    000000

TYPE    KEY DEFINITION PACKET
KTYP
KYLN    000036
KKYP
KDKH    000000
KDKL    000000
RESD    000014
RESD    000015

FIELD   # 1: SSSSSSSSSSSSSSSSSS
```

Delete Line Command

Use the Delete Line command to do either of the following:

- Delete an entire command string.
- Erase one or more of the characters you have typed in response to an INQUIRE command prompt.

Different terminals use different keys to delete a line, as we show in the following list.

Terminal Type	Delete Keys
Hardcopy	DEL
4010I	DEL
6012	RUBOUT, CTRL-U
605X	ERASE EOL, CTRL-K

Depending on which terminal you use, your command to delete a line looks like this:

↑ U, or like this: ^U on your screen. This results when you enter any of the appropriate keys. For example:

```
COMMANDS:   K W ↑ U
COMMANDS:
```

Suppose you want to write the key PERSONNEL to the main index, but you misspell it and type the following:

```
COMMANDS:   K W B ↵
KEY:   PERSONAL_
```

You can correct your error before pressing NEW LINE by entering the Delete command twice. Then, your entry will look like this:

```
KEY:   PERSON_
```

Now you can enter the remaining characters for the key and press NEW LINE.

```
KEY:   PERSONNEL ↵
```

Set Repeat Count Command (#)

You can use the Set Repeat Count command, #, when you want to gain access to several records sequentially. The keys for these records must all be in the same subindex. For example, suppose you enter the following commands:

```
COMMANDS:   ↓ ↵

COMMANDS:   → # ↵
REPEAT COUNT:   6 ↵
```

The results are:

```
FIELD # 1: RECORD ONE
FIELD # 1: RECORD TWO
FIELD # 1: RECORD THREE
FIELD # 1: RECORD FOUR
FIELD # 1: RECORD FIVE
FIELD # 1: RECORD SIX
```

Retrieve Current Position Command (?)

When you use the Retrieve Current Position command, INQUIRE returns information to you about the key where you are positioned. This information includes the key name for each subindex level down to the key at your current position, and the key's occurrence number. For example, your dialog could look like this:

```
COMMAND:   ? ↵

SUBINDEX LEVEL 0 KEY: NAMES
SUBINDEX LEVEL 1 KEY: ADAMSON

OCCURRENCE NUMBER: 000000/000005
```

End of Chapter

Chapter 7
INFOS II File Backup and Recovery Options

In commercial data processing, there is always the possibility of a system failure, from which you must be able to recover. Because INFOS II files have complex structures and are often very large, verifying them can be an involved procedure. To help you, the INFOS II system offers a series of options designed to keep data loss to a minimum after a system failure.

Specifically, the INFOS II system maintains two types of protection for your files:

- Structural Integrity — the INFOS II system maintains the correct internal structure of both your database and index files.
- Data Integrity — Information in all your records and their associated keys is correct up to the last modification made.

The INFOS II system offers this protection through four recovery options that you can set for any database file.

1. Modified Page Flush (MPF).
2. Differential File Mode (DF mode).
3. Buffered Request Logging.
4. Immediate Request Logging.

If you don't use one of these options, you will process your INFOS II file in Standard File Mode (SF Mode), which offers no backup or recovery protection.

Which recovery option you choose depends on your application and on how easily you can re-enter data. There is no "best" option; they all have tradeoffs. In general, the better recovery an option offers, the more it slows system performance time. Only you can decide where your priorities lie in this regard.

Figure 7-1 provides an approximation of where each option falls on the recovery/performance scale. It may help you in choosing a recovery option.

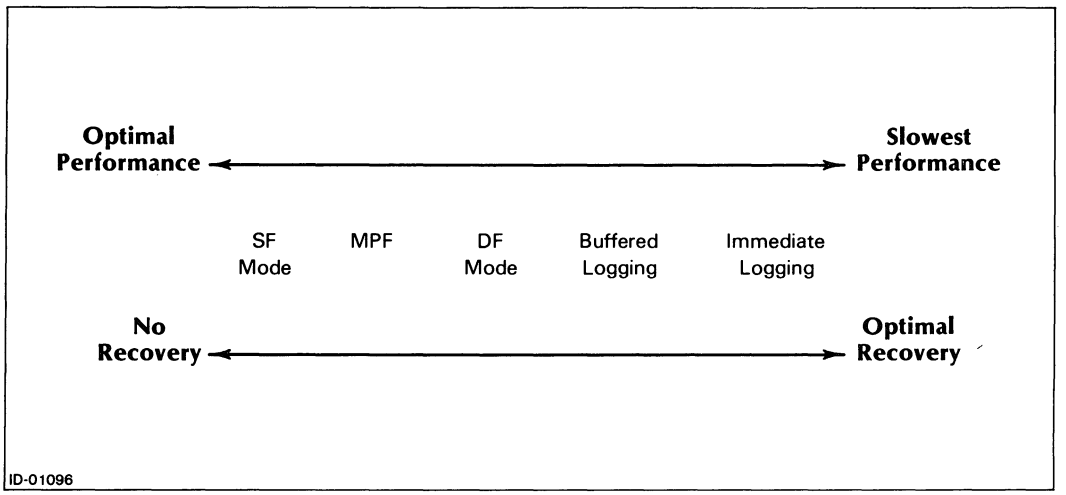


Figure 7-1. Recovery/Performance Scale

In addition, you might find the various options useful for the following situations:

Option	Situation
Standard File Mode	· Temporary INFOS II files. · Batch jobs. · Data easily re-entered.
Modified Page Flush	· Data not easily re-entered, but can be done.
Differential File Mode	· Data not easily re-entered, but can be done.
Buffered Request Logging	· Small amounts of data easily re-entered; large amounts of data not easily re-entered.
Immediate Request Logging	· Volatile data store. You cannot re-enter data; e.g., information taken over the telephone, not recorded.

You use the IFILE utility to select a recovery option. There are other INFOS II utilities that you must use in connection with the recovery procedure. For example, you use the IVERIFY utility to verify the structural integrity of your files. You can also use the DDUMP utility to back up your INFOS II databases. And if you choose one of the request logging recovery options, you will use the IRECOVER utility to restore your database after a system failure. We discuss all of these utilities in detail in this chapter.

The DDUMP Utility

In addition to using one of the INFOS II system's recovery options, we recommend that you periodically back up all of your databases with the DDUMP utility. This will help minimize data loss in the event of a physical media failure, regardless of which recovery option you select.

You might remember from Chapter 5 that you can use the DDUMP utility to make both archival and incremental dumps of your database. When you make an archival dump, you dump the entire INFOS II file; whereas with an incremental dump, you dump only those pages that you have modified since the last archival dump. So if you have recently made an archival dump of the database, you might only want to make an incremental dump. In any case, you should be sure that you have a tape dump of your database before you begin each session in which you use request logging.

See Chapter 5, "INFOS II Utility Programs," for more information on DDUMP.

Standard File Mode

Standard File Mode does not offer any recovery features; it does not provide protection for your files. It is the default processing mode that the INFOS II system gives you when you do not specify a recovery option with the IFILE utility. (We describe how you use IFILE to request recovery options later in this chapter.)

You will have better performance if you use Standard File Mode rather than a recovery option, but it offers you less chance of a full recovery from a system failure. It uses none of the available file backup options. With this mode, the system does not save your file modifications to disk right away, but passes your information to AOS.

AOS uses a feature called *shared page(s),* which means it stores every user's modifications in memory rather than writing them to disk right away. The system orders the modified pages in a "least recently used" (LRU) fashion, and when memory is full, it writes the least recently modified page out to disk. When a user wants to modify a page that is still contained in memory, INFOS pulls the page out of its place and modifies it. AOS then places the page at the head of the LRU line, since it has become the most recently used page. You have fast performance with Standard File Mode, because the shared pages feature minimizes the time-consuming process of writing to the disk.

Standard File Mode might be appropriate for the following situations:

- Building large INFOS II files for which you can easily re-enter all your information.
- Processing INFOS II files that you periodically save offline or delete.
- Using temporary INFOS II files.
- Using read-only databases.

Although Standard File Mode speeds your system performance time, if your system fails, you won't know how many of your modifications AOS actually entered into your database. In addition, if the system were to fail during a modification request, the file could become corrupted (i.e., inconsistent and therefore unusable). In this case, you would have to restore the file from a backup.

You must run IVERIFY to check the structural integrity of any INFOS II file that you were using in SF mode when the system failed. You cannot run the IRECOVER utility on a database that you were using in SF Mode.

Modified Page Flush

Modified Page Flush (MPF) provides nearly complete recovery in the event of a system failure. For each command you issue with the MPF option, the INFOS II system instructs AOS to write the modification to the disk immediately. After which, control returns to you. Thus, if the system should fail, you probably wouldn't lose any modifications. The only command that you might have to reissue would be the last one prior to the system failure, and only if INFOS II had not yet returned control to you.

There is, however, the possibility that your file was open for updates when the system failed. As a result, you would have to run IVERIFY on your MPF files to ensure their integrity.

MPF offers an obvious advantage in recovery over SF Mode. Remember that modifications made with SF mode remain in memory for an indefinite period, as AOS orders shared pages in an LRU fashion to minimize disk access time. In contrast, the system makes your MPF modifications immediately. Accordingly, using MPF slows down your entire system by increasing the number of times it writes to the disk. We recommend that you use MPF for on-line updating of your database only, since it does slow performance. It is useful, however, for users who find it difficult to re-enter data.

You should also be aware that if the system fails during a modify request when you are using MPF, your file could be corrupted. As with SF mode, you should always run IVERIFY on your file after a system failure, to check its integrity.

IVERIFY

In Chapter 5, we discussed how to use the IVERIFY utility to verify an INFOS II file's structural validity and to retrieve statistics on database and index use. Here, we will describe how to use IVERIFY in the backup and recovery of INFOS II files.

You can use the IVERIFY utility to check the structural integrity of an INFOS II file. IVERIFY cannot, however, assure the integrity of your data. Use IVERIFY primarily with Standard File Mode and Modified Page Flush.

The system has SF and MPF files open while it modifies and updates them. You must run IVERIFY on all of these open files after a system failure to ensure structural integrity. However, if a file's users have read-only access (an ACL of RE) or have the file opened with the read-only option, you will not need to run IVERIFY. Users performing just read requests cannot modify the file, so structural integrity is maintained.

On MPF files you need only check the last modification made by each user after running IVERIFY to ensure your data's integrity. But, if your file was in SF Mode and the system were to fail, you would have no way of knowing how many of the users' modifications were successful. You might have to re-enter the modifications from several sessions.

Differential File Mode

When you choose Differential File Mode, the INFOS II system builds a second INFOS volume, parallel in structure to your original INFOS volume. This second volume is the differential file. When you make a modification request, the system enters the changes into the differential volume only. This protects your master volumes because they are not open while you make your modifications. If the system were to fail, you would lose only the information in the differential file.

In the event of a physical media failure, however, you could lose your entire INFOS II file, depending on the severity of the failure. In the worst case, you would lose your differential files as well. You would then have to load a backup copy of the database file from tape and re-enter all of the modifications made since that backup.

The INFOS II system enters the changes from the differential file into the master volumes when you run the CHECKPOINT utility. The system does this automatically when INFOS II closes out an index file; it does this after the last user issues a CLOSE command. You can also initiate a checkpoint operation yourself at any time.

If the operating system or INFOS II were to fail, DF Mode would allow you to restore your file to the state it was in just after the last completed checkpoint simply by re-opening the file.

Creating an INFOS II Differential File

You can use Differential File Mode with any INFOS II file that has no more than 16 volumes. The file cannot have more than 16 volumes because Differential File Mode requires a duplicate for each volume, and the total maximum number of volumes you can have is 32.

Differential File Mode does not affect the structure of an INFOS II file or the application program. You can turn Differential File Mode on and off with the IFILE utility, as long as no INFOS user currently has the file open. From a user's perspective, programs run identically with or without Differential File Mode.

We describe the IFILE format you use to activate Differential File Mode for an INFOS II file in the "Choosing Recovery Options" section, later in this chapter.

Structure of a Differential File

Differential files are structurally similar to standard INFOS II files. The index and database files are actually two AOS control point directories containing AOS files. These files are your INFOS II file volumes. With DF Mode, each INFOS II volume has a corresponding differential volume, as shown in Figure 7-2.

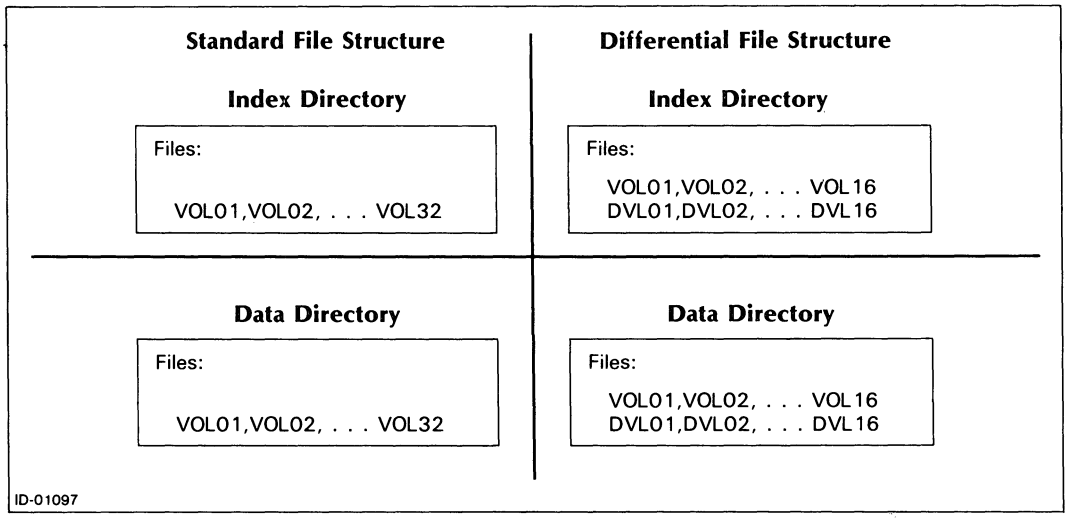


Figure 7-2. INFOS II Differential File Structure

When you enable DF Mode, the INFOS II system creates the volumes DVL01 through DVLnn, where nn is the number of standard volumes in your file. Remember that you are restricted to a maximum of 16 volumes in a differential index/database file.

CAUTION: To avoid unpredictable results:

- Do not name any of your standard volumes DVL01 through DVL16.
- Do not rename differential volumes.

When you use Differential File Mode, the INFOS II system writes the modification information into the differential volume for the file. With each modification request you make, the system reads 1- or 2-Kword pages from the master volume into the differential volume. It then performs your modifications on the pages in the differential volume, not on those of the master volume. Later, if you modify information on the same page, the system will not need to involve the master volume again; it will write directly to the differential file. All users share the same differential volumes when they simultaneously use the same data or index file.

Using Differential File Mode protects your INFOS II file because you do not modify the master volumes. If the system were to fail, you would lose only the information in the differential volumes. Note, however, that the system would allocate space in the master volumes for the new data. This ensures that the master volumes have room on the disk to hold the new data they receive at checkpoint time.

The CHECKPOINT Utility

The CHECKPOINT utility enters differential file data into the master volumes. Two situations will cause a checkpoint operation:

- When the last user closes an INFOS II file.
- When you run the CHECKPOINT utility.

If you want to close an INFOS II file, but don't know if you are its last user, then you should execute the CHECKPOINT utility before your CLOSE request. This ensures that your modifications will be entered into the system.

Whether you or the system initiates it, a checkpoint operation performs the following series of steps:

1. Prevents all users from gaining access to the INFOS II file while the checkpoint takes place.
2. Ensures that the system has written all file updates to the disk. Once this step is complete, you can restart the CHECKPOINT utility if the system should fail during the checkpoint.
3. Opens all the differential volumes for read-only access.
4. Opens the master volumes for modifications and updates.
5. Substitutes the modified pages in the differential volumes for the corresponding pages in the master volumes.
6. Deletes the information in the differential volumes, and revises internal databases to show that information formerly in the differential volumes is now in the master volumes.
7. Restores access to the INFOS II file.

If the system were to fail in the middle of a checkpoint, some of the information in the differential file might not be transferred to the master file. To protect against this, the system will not permit anyone access to the INFOS II file until you restart the checkpoint operation after a system failure. If anyone attempts to reopen the file for processing in this case, INFOS II will send the following error message:

```
(IOCPC) DIFFERENTIAL FILE CRASHED DURING CHECKPOINT- RUN CHECKPOINT UTILITY
```

Be sure that you are running the same version of INFOS II that was running at the time of the system failure when you restart the CHECKPOINT utility. Otherwise, you will receive the following error message:

```
(IOVER) FILE VERSION CONFLICT
```

If the system were to fail before a checkpoint, it would delete the differential volumes, and you would have to re-enter all the modifications made since the last checkpoint.

Command Format

You can manually issue a checkpoint operation with the CHECKPOINT utility. You must have owner (O) access to the database file to start a checkpoint. To restart a checkpoint after a system failure, you simply issue the CHECKPOINT command in the same manner as if the INFOS II file were open. The command format is as follows:

CHECKPOINT *[/L[=listfilename]]* databasefilename

Switch	Action
/L[=listfilename]	Writes the output to the specified list file. If you don't give a name, the system will use the default list file, @LIST.

When you run the CHECKPOINT utility, it returns the following information:

- The date and time the system performed the checkpoint.
- The number of pages modified in the checkpoint.

The system will not return the checkpoint date and time information if the checkpoint was initiated by a CLOSE request. However, when you execute the IFILE utility on the database, the utility returns the date and time of the last checkpoint.

Some Hints on How Often to Run CHECKPOINT

How often you run the CHECKPOINT utility depends on your data processing environment. When making this decision, consider how frequently the file is updated and the volume of the changes made to it in a given time period. (You can obtain this information from the listing output from the CHECKPOINT utility.) Consider the following points:

- Running CHECKPOINT requires some initial overhead time, and a certain amount of additional time for each modified page that CHECKPOINT enters into the master volumes.
- CHECKPOINT blocks access to the INFOS II file while it is running. Therefore, if CHECKPOINT has large amounts of data to enter, the delay could seriously disrupt system response time.
- Infrequent checkpoints would require you to re-enter a great deal of data if the system were to fail.
- Individual users cannot be aware of the total amount of information in the differential file at any one time. Therefore, we recommend that you don't allow all users access to the CHECKPOINT utility.

Consider the number of modified pages as an indicator of the amount of lost time if the system were to fail, versus the overhead time you would need to run CHECKPOINT.

We recommend that you run the CHECKPOINT utility frequently in an environment where files are constantly modified. You should do this whether you use Differential File Mode alone or with request logging. (We describe request logging in the next section.)

In the extreme case of an on-line terminal used a few times a day, you might want to do a checkpoint after every entry, but Modified Page Flush is probably better for this situation. If you are building a very large INFOS II file, you might want to use CHECKPOINT on the file periodically while you are processing it, provided you are able to specify a point from which to continue building. This would save portions of the file if the system were to fail. In environments with multiple application programs, it is important to remember that INFOS automatically runs a checkpoint when the last user program issues a CLOSE request on the file.

INFOS II Request Logging

Another type of INFOS II recovery you can choose involves request logging. For this option, INFOS II uses a logging facility that records in a log file all the update commands you issue to it. You can then use that log file to reprocess your database after a system failure. You must only use this log file for immediate file recovery; it is not meant for any other purpose. Also, note that log files you create with one version of INFOS II are not necessarily usable by later versions. Be certain to recover your files using the same version of INFOS II that was running at the time of the system failure.

Although your log file can be either a tape or disk file, for simplicity we will refer to it as a tape file in this chapter. Bear in mind that it can be either one.

The INFOS II logging facility uses the AOS Common Logger, a general-purpose utility, which can handle the logging process for several systems, including INFOS II and user applications. The Common Logger consists of a logging program, COMLOG, and an unlogging program, COMUNLOG. Briefly, COMLOG is the program that writes your requests to a log tape or disk, while COMUNLOG is the program that replays those logged requests at recovery time. You can find out more information about the AOS Common Logger in the "Controlling the Common Logger (COMLOG)" section, later in this chapter. See also the latest COMLOG Release Notice.

Any machine that supports COMLOG and COMUNLOG can use request logging. But, you must have revision 1.30 or later of COMLOG to log to disk.

The COMLOG utility allows multiple application programs to share a single log tape. You recover the logged information with the IRECOVER utility. We describe IRECOVER in detail later in this chapter.

In the event of a physical media failure, you must load the most recent backup tape of the database and reapply the update requests on the log tape to that version of the file. As we explained earlier, you should make an archival or incremental dump of the database before each logging session. We describe the recovery procedure in more detail, later in this chapter.

There are two COMLOG logging mode options available for your INFOS II databases. You can choose either Logon Immediate Mode or Logon Buffered Mode, which is the default.

Logon Immediate Mode

If you use Logon Immediate Mode, the INFOS II system writes all of the information necessary to perform your request to a log tape before allowing you to make another request. Therefore, if the system were to fail, you could restore your file to the state it was in after the last completed update request. Since each request must wait for the previous one to be logged, Logon Immediate Mode can slow system response time. It does, however, offer the more complete data recovery of the two logging modes.

Logon Buffered (Logon) Mode

If you specify Logon Buffered Mode, the INFOS II system writes request information to a COMLOG buffer rather than to tape or disk. Once it writes a request into the buffer, the INFOS II system allows you to issue another request. When the buffer is full, COMLOG writes its contents to the log tape. You specify the size you want for the buffer via the COMLOG utility.

If the system were to fail, you would lose any requests currently in the COMLOG buffer. The number of requests you'd lose would depend on the size of the buffer. For example, suppose you selected a buffer size of 1000 bytes, and you then issued a request that required 300 bytes, one that needed 500 bytes, and then two that required 150 bytes each. If the system were to fail after the third request, you'd lose all three requests, because they totaled only 950 bytes and were still contained in the buffer. You would then need to restore the file to the state it was in before the three update requests. If, however, the system were to fail after the fourth request, you'd lose only that fourth request. This would be the case because the buffer would be too small to accommodate the last request along with the first three. Therefore, the system would have written the first three requests to disk before storing the fourth in the buffer. You would only have to re-enter the last request in this case.

Logon Buffered Mode provides better performance than Logon Immediate Mode, but data recovery isn't as complete. If you are able to re-enter the last few updates which are lost from the buffer, then the better performance of this mode might appeal to you.

If the system were to fail, you would use the IRECOVER utility to unlog the data from the log tape. IRECOVER uses COMUNLOG to retrieve the INFOS II modifying commands previously issued to the database being recovered, to execute the logged commands, and to return the file to the state it was in before the system failure.

INFOS II request logging and recovery, therefore, offer you an additional level of file protection to that offered by Differential File Mode:

- Differential File Mode returns files to a known state, through the CHECKPOINT utility.
- Request Logging allows you to extend the integrity of the file up to the time of the last logged request.

Keep in mind that you sometimes issue two logically related requests, such as WRITE and DELETE, on the same piece of data. With the INFOS II system, you cannot remove previous requests unless you specifically issue the inverse operation. If you were to issue two logically related requests, and the system were to fail after logging the first request, but before the second, then INFOS II would reissue only the first request upon recovery. You would have to re-issue the second request.

Enabling Request Logging

When you enable logging for a database file, you are turning on one of the request logging modes. These modes have all the features of differential file mode plus the command logging capability.

Before you can open an INFOS file using request logging, COMLOG must be running. Otherwise, you will receive the message

```
(IONCL) COMLOG PROCESS NOT RUNNING
```

and INFOS II will deny access to the file. See the "Controlling the Common Logger (COMLOG)" section for information on how to start the COMLOG process.

As with the other recovery options, you use the IFILE utility to select one of the request logging options for each of your databases. We describe how to select one of the request logging modes in the "Choosing Recovery Options" section, later in this chapter.

The system can open up to 255 INFOS II files with logging enabled at the same time. If you attempt to open a file with logging enabled, and there are already 255 open files being logged, you will receive the following error message:

```
(IOLOP) LOG FILE OPEN REQUEST EXCEEDS MAXIMUM NUMBER ALLOWED
```

The 255 limit includes all files opened, as well as multiple opens of the same file.

How Request Logging Works

When a user issues the first open of a file with logging enabled, INFOS II initializes the interface to COMLOG. It then logs a message containing the time and date that the INFOS II system was brought up.

For every modify command you issue to that file, INFOS II sends a single message to COMLOG consisting of the following data:

- Packets
- Filenames
- Keys
- Partial records
- Data records

For each command, COMLOG records this information as one log record in the log file.

Each log record also contains a message with the time and date of the last checkpoint completed on the file. If the system were to fail, IRECOVER would use this information to determine which modifications were not made to the file (i.e., those issued since the last completed checkpoint). We discuss the IRECOVER utility in more detail later in this chapter.

If COMLOG were to terminate while the INFOS II system was logging a command, the system would send the following message to the terminal where INFOS II was brought up:

```
(IONCL) COMLOG PROCESS NOT RUNNING
```

INFOS II would also send an error code to the program of the user who issued the command.

If you receive this error message, get COMLOG running and then reissue the command.

Creating Alternate Index Files When Using Request Logging

In some applications, you might want to create a new index (alternate index file) for a database that has logging enabled. To do so, you must first turn logging off. You can then use the OPEN/CREATE command with inversion to create the new index, and turn logging back on afterwards. If you attempt to create an alternate index for a database without first turning off logging, you will receive error message 7060:

(IOCOI) CANNOT ISSUE INVERTED CREATE ON FILE IN LOGGING MODE

Controlling the Common Logger (COMLOG)

Because many users log to the same log file, someone must assume the operator's responsibilities of managing the log volumes and starting INFOS II and COMLOG. This section explains how to perform these functions.

AOS INFOS II uses standard COMLOG. Typically, you start the COMLOG utility before starting the INFOS II process; however, you can start COMLOG just before the first INFOS II user issues an open request to a file with logging enabled.

CAUTION: When restarting COMLOG, be sure to change volumes (i.e., a new tape or a new disk file), or COMLOG will write over the previously recorded requests.

Bringing Up the INFOS II and COMLOG Processes

Bring up INFOS II and COMLOG with PROCESS commands. For example, an UP macro to start INFOS II and COMLOG might include the following:

```
......
PROCESS/DEFAULT/DIRECTORY=@ COMLOG/switches
PAUSE 10
CONTROL @COMLOG START <unit>
PROCESS/DEFAULT/DIRECTORY=@/NAME=INFOS_II/RESIDENT INFOS_II.PR
......
CONTROL @EXEC ENABLE/ALL
......
```

COMLOG requires that you specify an AOS pathname for <unit>, such as @MTB0 or :LOGFIL:LOG_12_02_83. The file must already exist. If you are logging to disk, we recommend that you initialize the log file with the /INIT switch on the COMLOG command line.

See your COMLOG Release Notice for a detailed explanation of how to bring up COMLOG, the available switches, and how to increase buffered logging performance with COMLOG.

Remember that you cannot change the log file type when you switch volumes in the middle of a log file.

Opening a New Log File

COMLOG writes its own file and volume label records to log files; it does not use AOS labeled tapes. COMLOG writes a *date.time VOL1* label to a log file as soon as it becomes active. The log file becomes active when you bring up COMLOG and issue the START command. It remains active until the COMLOG process terminates.

End-of-Volume Condition

When the log file reaches the end of a tape volume, COMLOG writes an end-of-volume record to it, rewinds it, and prompts you (at the operator's console) to mount a new volume. COMLOG labels subsequent volumes *VOL2, VOL3,* etc. with the same *date.time* that it gave the first volume of that log file.

You should apply a label with the correct volume number and the date onto each tape volume.

Forcing an End-of-Volume Condition

If the INFOS II system fails and COMLOG does not shut down, force an end-of-volume condition by issuing the command:

```
)CONTROL @COMLOG FEOV ⤸
```

This will write an end-of-volume message on your log tape or disk, so that when INFOS comes back up you can restart logging on a new tape or disk. It does not stop COMLOG. However, issuing this command might not always succeed. If it does not, make sure no other processes are logging, and then issue the following command:

```
)CONTROL @COMLOG STOP ⤸
```

If there are no processes logging, this command will terminate COMLOG. However, if there are still processes that are customers to COMLOG, this command will not succeed, and you will have to issue the following command:

```
)CONTROL @COMLOG ABORT ⤸
```

This aborts COMLOG and will terminate logging. If you subsequently use COMUNLOG with IRECOVER to recover the logged entries, you will receive the message:

UNEXPECTED END-OF-FILE ON BLOCK <octal number>

You will then have to enter the CONTROL @COMLOG STOP command if you want to recover. This will force a normal end to the recovery session, and IRECOVER will replay the modifications made up to the indicated block number. (See "The IRECOVER Utility" section of this chapter.)

Initiating a Volume Switch

In environments with multiple application programs, a volume switch could inconvenience many users because all COMLOG users have to wait while you mount a new tape. But if you have at least two tape drives, you can avoid this problem by using the COMLOG PREMOUNT command in place of an anticipated START or MOUNTED command. To do this, use the following command line:

```
CONTROL @COMLOG PREMOUNT <unit name>
```

With the PREMOUNT command, COMLOG can check and queue up the next log file name before the actual end-of-volume condition occurs. This allows a smooth transition from one volume to another and avoids making logging users wait for a volume switch. PREMOUNT is an unsolicited command, meaning that COMLOG doesn't prompt you for it.

Alternatively, if you have only one tape drive, you can use the PREMOUNT command for the initial start request in the system UP.CLI macro. By doing so, you don't have to wait until COMLOG is ready and then issue a MOUNTED command. With PREMOUNT, COMLOG will do validity checks and reject the command on error; e.g., if the filename you specify does not exist.

You can only queue one PREMOUNT command at a time. If you specify more than one, COMLOG uses the most recent one. Therefore, if you make a typing mistake or use the wrong filename, you can issue the PREMOUNT command again, using the correct name, and COMLOG will use it instead of the previous request.

Note that you can use the PREMOUNT command with disk files too. Issue the CLI SPACE command to determine how much space is left in the log volume. When you see that it is becoming full, use the PREMOUNT command to queue up the next volume.

Closing the Log File

To close the log file, you must issue the following command:

```
)CONTROL @COMLOG STOP )
```

COMLOG will then write an end-of-logfile record to the log file. Be sure to carefully store all volumes belonging to the day's log file.

Note that COMLOG will accept the STOP command as soon as all logging files are closed to INFOS. This enables those INFOS applications that do not use logging to continue, while you safely store the log file.

Handling Hardware Errors

When COMLOG finds a hardware write error on the log file, it reports the error and the block number where it occurred. You should make a note of the block number that caused the error, in order to check for consistency at IRECOVER time. COMLOG then asks for instructions on what to do next. In most cases, you will want to close the current volume and mount the next one. If so, enter the following command and proceed as usual:

```
)CONTROL @COMLOG FEOV )
```

Alternatively, you could abort COMLOG. However, you should avoid using the CONTROL @COMLOG ABORT command if INFOS has files open with logging enabled. If COMLOG aborts, its users will receive an error message and will no longer be logging. In addition, COMLOG will not write an end-of-file marker on the log file, which you should have for unlogging.

Terminating the INFOS II and COMLOG Processes

AOS INFOS II creates an IPC file called @INFOS for operator communication. The only command it accepts is the STOP command:

```
CONTROL @INFOS STOP
```

If there are no files open, INFOS accepts the command and terminates with the following message:

```
(IOBYE) INFOS PROCESS SHUTDOWN
```

If there are files open, the system will respond with the following:

```
(IOACT) INFOS USERS ACTIVE, TERMINATION REFUSED
```

Allow all INFOS users to terminate before you try the command again.

A typical macro to bring down INFOS and COMLOG includes the following:

```
......
TERM OP:EXEC
PAUSE 10
CONTROL @INFOS STOP
PAUSE 10
CONTROL @COMLOG STOP
......
```

Choosing Recovery Options (IFILE)

In choosing a recovery option, you must be aware of the tradeoffs between speed and security. In general, the better recovery an option offers, the slower the system performance. Likewise, the better your performance, the less complete your recovery.

In Chapter 5, we discussed using IFILE to retrieve information about an index or database file. You can also use the IFILE utility to select one of the recovery options. If you do not choose one, INFOS will assign you the default, Standard File Mode. The INFOS II System automatically creates all databases in Standard File Mode, which provides no backup and recovery protection.

If the file is closed, you can use IFILE to change file modes without modifying current application programs. All the programs that gain access to the INFOS II file automatically perform their operations using the file mode you select.

You make your modifications to an INFOS II file in the processing mode that the file is currently using — Standard File Mode, Differential File Mode, Modified Page Flush, or one of the Request Logging Modes. You can reset the mode only through the IFILE utility, not through application programs. All the file modes are invisible to application programs. Use the IFILE utility after creating the file if you want to change its processing mode.

Command Format

You must have at least owner (O) access to a file in order to change processing mode options. The command format is as follows:

IFILE databasename *[keyword]*

You use the keyword clause with IFILE to set your desired option, where keyword can be any one of the following:

Keyword	Action
S	Retains or restores Standard File Mode (no recovery option).
MPF *FLUSHON*	Turns on Modified Page Flush.
DF	Turns on Differential File Mode.
LOGON IMMEDIATE	Turns on immediate request logging.
LOGON BUFFERED *LOGON*	Turns on buffered request logging.
LOGOFF	Turns off request logging.

To find out what file mode a database has, issue the IFILE command without a keyword.

If you turn on request logging, you also will enable Differential File Mode for the file. If you specify LOGOFF, you will disable request logging for the file; however, the file will remain in Differential File Mode. To disable logging *and* reset the file mode, use IFILE and specify either Standard File Mode (S keyword) or Modified Page Flush (MPF or FLUSHON keyword).

IFILE Output

We described the IFILE output in Chapter 5. This output will include the following statements for the file mode you have chosen:

Option	Statement
Standard File Mode	(No Statement)
Modified Page Flush	*MODIFIED PAGE FLUSH ENABLED*
Differential File Mode	*DIFFERENTIAL FILE MODE ENABLED*
Buffered Request Logging	*DIFFERENTIAL FILE MODE ENABLED* *REQUEST LOGGING (BUFFERED) ENABLED*
Immediate Request Logging	*DIFFERENTIAL FILE MODE ENABLED* *REQUEST LOGGING (IMMEDIATE) ENABLED*

For example, suppose you have a database named INVENTORY.DB, and you want to process it using Buffered Request Logging. You would use the following command:

```
)IFILE INVENTORY.DB LOGON BUFFERED ↓
```

The IFILE output will include the following:

```
DIFFERENTIAL FILE MODE ENABLED
REQUEST LOGGING (BUFFERED) ENABLED
```

Note that in the above command line, we could have omitted the word BUFFERED. Because buffered logging is the default logging mode, you can ask for it just by specifying LOGON.

With DF Mode or request logging enabled, IFILE also returns the date and time of the last checkpoint (see "The CHECKPOINT Utility" section, earlier in this chapter).

All indexes associated with a database file assume the mode of the database. This is true even if you create the index after the database (i.e., an inverted index). Initially, database files are always created in SF Mode.

You can run the IFILE utility itself at any time. However, if any user has the file open, you cannot change the file mode.

Note that resetting the file mode might not be in the best interests of other users of the file. Therefore, you might want to prevent some users from using IFILE to change the file mode. You can do so on a per file basis with the AOS access control list (ACL) facility. IFILE requires the user to have owner (O) access to the INFOS II file in order to change modes.

The IRECOVER Utility

IRECOVER is the INFOS II utility that you use to recover from a system failure when you choose request logging as your recovery option. Use IRECOVER if the failure occurred while you were logging; do not use it if the failure occurred during a checkpoint. IRECOVER replays the log files, applying their requests to the files you loaded from an archive tape or the file that was open at the time of the system failure. (Note that you should only use the latter if the disk wasn't corrupted.)

If the system were to fail while a checkpoint was taking place, you would have to restart the checkpoint. The system would prompt you to do so. In this case, you would not run IRECOVER.

Operating Environment

IRECOVER operates with one or more databases at the same time. You can run IRECOVER on the database that was in use when the system failed, or on an older version of the database restored from the backup. In the latter case, you must have all the log tapes that were produced since the time of the backup. You would want to use the backup version of the database to recover if the disk was corrupted at the time of the system failure.

Remember that you can only use IRECOVER on databases that have logging enabled. If you had used Standard File Mode or Modified Page Flush, you would have no record of the modifying requests. Likewise, you would have no record if you were using Differential File Mode without request logging.

Note that IRECOVER is restartable, provided the databases are in DF Mode (with or without logging) at recovery time. That is, you can abort IRECOVER at any time before it sends the message that the file is closed. You can then restart the recovery later.

You should also note the following:

- COMUNLOG *must* be running for you to use IRECOVER.
- COMLOG does not have to be running for you to operate IRECOVER.
- You cannot run IRECOVER in batch. It is an interactive utility, which under certain circumstances prompts you at the operator's console for instructions during its execution.

When IRECOVER opens a file, it temporarily disables logging for that file. It does this because all the commands it is issuing are already in a log file. IRECOVER opens the INFOS II file exclusively to prevent other users access to it during the recovery process.

When you use IRECOVER to recover from a system failure, you know that your file contains no partially executed commands. IRECOVER simply issues all commands logged since the last checkpoint. When IRECOVER closes a file, a checkpoint takes place, ensuring that all the modifications it made are entered into the database.

After a system failure, you must immediately recover your file using IRECOVER. If you attempt to reopen the file for processing before using IRECOVER, you will receive the following error message:

```
(IORCV) LOG MODE FILE OPEN DURING SYSTEM OR INFOS II CRASH- RUN IRECOVER
```

If you tried to run the CHECKPOINT utility on the file after a failure and a checkpoint had not been in progress, you would receive the following error message:

```
(IOCNC) CRASHED FILE WAS NOT CHECKPOINTING - RUN IRECOVER
```

IRECOVER Input and Output

IRECOVER uses the AOS utility COMUNLOG to replay the log files and retrieve your logged requests. It performs extensive *timestamp* checks to be certain that it is operating on databases that match the logfile. (See the "Replay and Timestamp Logic" section, later in this chapter.)

IRECOVER creates two output files, the *audit file* and the *status file*. You can print these files to examine their contents. We include samples of audit files and status files in our IRECOVER examples, later in this IRECOVER section.

Audit File

The audit file reports the final recovery status for all the databases on the log file. This status includes the following:

- Database pathname.
- Recovery status (valid or not valid).
- Index pathname.
- Recovery status (successful or unsuccessful opening of file).
- Status of channels that are active at the end of the logfile, if any.

The status of active channels includes the following information:

- Channel number.
- Last group number: This refers to TPMS transaction IDs and is applicable only to TPMS users. For non-TPMS applications, the field consists of dashes. (TPMS users should refer to the note below.)
- Username.
- Program name.
- User ID: We describe this number in detail in the "Status File" section, below.

You can see all of this information later in the sample audit file included with the IRECOVER example.

NOTE: TPMS users should note that IRECOVER reports the last TPMS transaction ID completed by each user in the audit file; therefore, recovery of *completed* TPMS-INFOS automatic transactions requires only one pass across the log file. However, if the file contains any incomplete TPMS transactions, IRECOVER will need to perform second accesses to undo the requests of the incomplete TPMS transaction group.

Also note that TPMS does not use the audit file. The information that it needs for restart is contained in the TPMS.TX in the database CPD.

In addition, you can use global switches on the IRECOVER command line to select a formatted display of the requests on the log file. You can have IRECOVER report just the open and close requests, or all the update requests. Note that if you ask for the full display, your audit file could be quite large, and recovery will take longer. This option is available mainly for debugging purposes. IRECOVER prints out this information *before* the final recovery status in the audit file.

If you ask to have your OPEN requests reported in the audit file, the report will include the following:

- Index name.
- Database name.
- Username.
- Superuser privilege, if it was in use at the time of the open.
- Process ID (PID) number.
- Program name.
- Process number.
- User ID number.

Each modifying request that IRECOVER reports includes the following information:

- Type of positioning used for the request.
- Key length (for the key at each level that was accessed).
- Partial record length, or statement indicating a suppressed partial record.
- Database record length, or statement indicating a suppressed data record.
- Statement as to whether or not IRECOVER successfully replayed the request.

In addition, the audit file includes a four-line picture of each key and record to which it gains access. We show an example of a 20-byte data record in Figure 7-3.

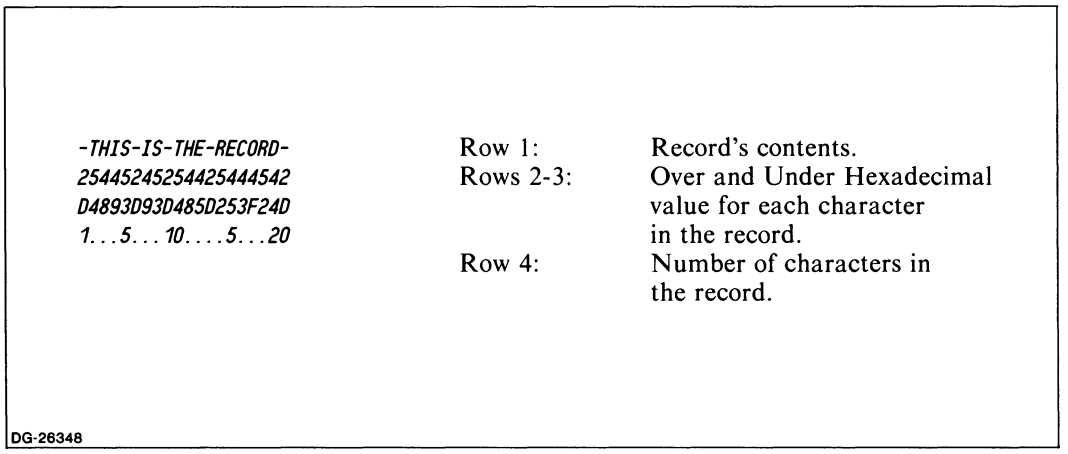


Figure 7-3. Data Record as Represented in an Audit File

You can instruct IRECOVER to send the audit file to another terminal if you want to examine it while IRECOVER is running.

Checking IRECOVER results in the Audit File

After running IRECOVER, you should always check the audit file to be certain that it reports valid recovery status in the section headed FINAL RECOVERY STATUS BY FILE. The audit file reports the status for the indexes and the databases that IRECOVER attempts to recover. It indicates valid status in the following way:

File	Message
Database	*VALID RECOVERY STATUS*
Index	*SUCCESSFULLY OPENED FOR RECOVERY*

If the recovery status is not valid, the audit file will report the errors it found. For example, the index status could be *FILE DOES NOT EXIST* or *FILE ACCESS DENIED*, and the database status might be *MISSING UPDATES*, or *PROPER INFOS SESSION NOT FOUND*. See the "Replay and Timestamp Logic" section, later in this chapter, for more information on these database errors.

Depending on which error the audit file reports, you might be able to fix the problem and recover the file. For example, if you receive *FILE ACCESS DENIED* status, due to an ACL problem, you can change the ACLs and run IRECOVER again. IRECOVER wouldn't have been able to open the file the first time you ran it, so the file would be in the same state as it was at the time of the system failure.

If your status is *DIFFERENTIAL FILE CRASHED DURING CHECKPOINT -- RUN CHECKPOINT UTILITY,* then you need not run IRECOVER. All you must do is restart the checkpoint.

NOTE: If you had logging enabled on only one database when your system failed, then you might want to check its status with the IFILE utility before you run IRECOVER. If IFILE reports *CHECKPOINT IN PROGRESS* at the time of the failure, then you won't have to run IRECOVER. Just restart the checkpoint with the following command line:

```
X CHECKPOINT databasename
```

At the end of recovery, IRECOVER displays messages at the terminal screen, reporting the names of databases it is closing. The output looks like this:

```
databasename1 SUCCESSFULLY CLOSED
databasename2 SUCCESSFULLY CLOSED
```

These messages do not mean that the recovery was successful. Rather they indicate that IRECOVER opened the named file, and subsequently closed it. You must still look at the audit file to determine the status of the database.

Status File

The status file shows the status of the files that IRECOVER opens. It also reports the username, program name, and user ID for all users that were active at the end of the last log file processed. The status file has no information in it until IRECOVER ends. If you run IRECOVER with the /ANALYSE switch (described in a later section), it will not create a status file.

The status file has fixed-length records, so that a program can easily read it. Each record has 131 characters, the last of which is a NEW LINE, which makes the file printable. The status file has three record types:

D Database record.

I Index record.

U User record.

Character position 1 of each record includes one of the above letters to indicate the record type.

Databases do not occur in any particular order in the status file, but records for each database occur in a given order, as shown below:

```
Database 1
Index 1         (First index for Database 1.)
User 1
User 2
...             (Other users, if any.)
Index 2         (Second index for Database 1.)
User 1
User 2
...             (Other users, if any.)
...             (Other indexes, if any, in same format.)
Database 2
Index 1         (First index for Database 2.)
...             (Users, additional indexes, as above.)
```

Record Format

The formats for database records, index records, and user records differ slightly.

Database records in the status file have the following format:

Character Position	Contents
1	D
2 - 129	Full pathname to the database file (128 characters, left justified).
130	A space.
131	NEW LINE

Index records in the status file have the following format:

Character Position	Contents
1	I
2 - 129	Full pathname to the index file (128 characters, left justified).
130	Y (if IRECOVER put the file in replay mode and updated it), otherwise a space.
131	NEW LINE

User records in the status file have the following format:

Character Position	Contents
1	U
2 - 7	Channel number (6 characters).
8 - 17	Last TPMS transaction (10 characters, right justified).
18 - 33	Username (16 characters, left justified).
34 - 65	Program name (32 characters, left justified).
66 - 69	User ID (4 characters, right justified).
70 - 130	61 spaces.
131	NEW LINE

The User ID Field

In some applications, you might require more identification to be reported in the status file than just the username and program name. For that reason, IRECOVER provides the ability to report a user ID, which the program passes to INFOS when it opens the INFOS file.

The user ID can be the user's console number, or some other value. The user process can retrieve its console number and send it to INFOS with the index File Definition Packet (FDP). (See Chapter 9 for a complete description of the INFOS packets.) Since there is not enough room in the index FDP to include the console number in ASCII characters, the program must retrieve its console number (or VCON, or batch stream) by an ?EXEC system call, and produce a small binary code.

Your user ID value is restricted in size because it must fit in some unused bits of the index FDP. You can store the binary user ID starting at the parameter ?SUID at offset ?FIFL in the index FDP (see Chapter 9). The user ID can be any positive number less than 2048.

For all non-TPMS programs, IRECOVER will report a zero user ID if your program does not supply one. Note that the passing of a user ID does not apply to TPMS programs.

NOTE: If you are using COBOL with INFOS II, refer to the appendix material of your latest *COBOL Reference Manual (AOS)* for information on how to modify INFOS II packets, so you will be able to pass the user ID.

IRECOVER Command Line

Use the following command line to start the recovery process:

X IRECOVER/switch(es) *[databasename]*

The *databasename* is optional on the command line. If you omit it, the system will recover all the databases on the log tape. Note that the database pathnames you specify must be exactly the same as they were when the application was running. IRECOVER will not find the database if you place it in another directory. The pathname must be the same one you used when you opened the file. Be sure you do not move any part of it before recovery.

IRECOVER Switches

You can type switches in any order. There are three kinds of switches:

- Audit file (required).
- Status file (optional).
- General Option (optional).

Audit File Switch

The audit file switch has this format:

/AUDIT=auditfilename *[/options]*

You must specify an **auditfilename**. Note that IRECOVER will delete any existing file with the same name in the working directory. If it finds **auditfilename** through the searchlist, IRECOVER will append the audit list to that file.

The audit file switch has the following options. You can only use these options if you also use the /AUDIT switch. Be sure you do not specify both the /BRIEF and /LONG options on the same command line.

Options	Purpose
/OPENS	Reports any OPEN, CLOSE, or CHECKPOINT.
/BRIEF	Reports requests in short form. Shows only the first 50 characters of keys, partial records, and data records.
/LONG	Reports requests in long form. Note that this option has the potential to produce a very large output file. It can also slow recovery time.

Status File Switch

The status file switch has the following format:

/STATUS=statusfilename

This switch is optional. If you do not include it and specify a name for *statusfilename,* IRECOVER will create a status file and assign it the default name according to the current date and time: IRCV.STATUS.dd_mm_yy.hh_mm_ss.

As with the /AUDIT switch, IRECOVER will delete a file with the same name as the *statusfilename* in the working directory, and will be able to find the status file through the searchlist.

This switch has no options.

General Option Switches

There are two general option switches you can use. They are optional.

Switch	Purpose
/ANALYSE	Use this switch if you want IRECOVER just to read the log file and make an audit list. No recovery takes place. IRECOVER does not produce a status file.
/RESTORE	Use this switch if you restored the database from backup (i.e., it is not the version that was open to INFOS at the time of the system failure). Of course it is possible that your backup database is identical to the one that you were using when the system failed, but if you recover using the backup, you *must* use the /RESTORE switch. If you don't, IRECOVER will find an error when it checks the timestamps (see the "Replay and Timestamp Logic" section, later in this chapter).

IRECOVER Examples

In the first example, we specify the name AUD3 for the audit file and STS3 for the status file name.

```
)X IRECOVER/AUDIT=AUD3/OPENS/STATUS=STS3/BRIEF ↵
```

IRECOVER will report to the audit file every OPEN, CLOSE, and CHECKPOINT that it finds, because we used the /OPENS switch. It will also include all of the INFOS requests in short form; it will abbreviate any long keys and records to their first 50 characters. We picture sample output from this IRECOVER session in Figures 7-4 and 7-5. The log file that we used included 2 OPEN requests and 40 WRITE requests, which wrote 40 level 1 keys named 0000000000 to 0000000039. Note that in Figure 7-4, we only included 3 of the 40 WRITE requests that IRECOVER reported in the audit file. The final logged request is a CLOSE.

```
++++++++++ INFOS INITIALIZATION - TIMESTAMP: 21-FEB-84 11:26:08

(104410) OPEN ------------------------------------------------ 21-FEB-84 15:38

       INDEX: :UDD:INFTREE:INFOS_TEST_INDEX
    DATABASE: :UDD:INFTREE:INFOS_TEST_INDEX.DB

    USERNAME: INFTREE          SUPERUSER
         PID: 31
     PROGRAM: OPWRITE.PR
     PROCESS: 031
     USER-ID: 0


(105010) OPEN ------------------------------------------------ 21-FEB-84 15:38

       INDEX: :UDD:INFTREE:INFOS_TEST_INDEX
    DATABASE: :UDD:INFTREE:INFOS_TEST_INDEX.DB

    USERNAME: LEW              SUPERUSER
         PID: 28
     PROGRAM: INQUIRE.PR
     PROCESS: 028
     USER-ID: 0
(104410) WRITE ----------------------------------------------- 21-FEB-84 15:38

    KEYED POSITIONING (1 LEVEL)

        LEVEL 1: LENGTH=10

            0000000000
            3333333333
            0000000000
            1...5...10

    SUPPRESS PARTIAL RECORD

    DATABASE RECORD - LENGTH=40

            0000000000----ABCDEFGHIJKLMNOPQRSTUVWXYZ
            3333333332222444444444444444455555555555
            00000000000DDDD123456789ABCDEF0123456789A
            1...5...10....2...20....5...30....5...40


    SUCCESSFULLY REPLAYED
```

Figure 7-4. Sample Audit File (AUD3) (continues)

```
(104410) WRITE ------------------------------------------- 21-FEB-84 15:38

    KEYED POSITIONING (1 LEVEL)

        LEVEL 1: LENGTH=10

            0000000001
            3333333333
            0000000001
            1...5...10

    SUPPRESS PARTIAL RECORD

    DATABASE RECORD - LENGTH=40

            0000000001----ABCDEFGHIJKLMNOPQRSTUVWXYZ
            33333333332222444444444444444455555555555
            0000000001DDDD123456789ABCDEF0123456789A
            1...5...10....2...20....5...30....5...40

    SUCCESSFULLY REPLAYED
```

```
(104410) WRITE ------------------------------------------- 21-FEB-84 15:38

    KEYED POSITIONING (1 LEVEL)

        LEVEL 1: LENGTH=10

            0000000039
            3333333333
            0000000039
            1...5...10

    SUPPRESS PARTIAL RECORD

    DATABASE RECORD - LENGTH=40

            0000000039----ABCDEFGHIJKLMNOPQRSTUVWXYZ
            33333333332222444444444444444455555555555
            0000000039DDDD123456789ABCDEF0123456789A
            1...5...10....2...20....5...30....5...40
```

Figure 7-4. Sample Audit File (AUD3) (continues)

```
    SUCCESSFULLY REPLAYED

(104410) CLOSE ---------------------------------------------- 21-FEB-84 15:38

********** NORMAL END OF LOG FILE


(Beginning of new page.)

            FINAL RECOVERY STATUS BY FILE

DATABASE: :UDD:INFTREE:INFOS_TEST_INDEX.DB
  STATUS: VALID RECOVERY STATUS
   INDEX: :UDD:INFTREE:INFOS_TEST_INDEX
  STATUS: SUCCESSFULLY OPENED FOR RECOVERY
          -- STATUS OF CHANNELS ACTIVE AT END OF LOGFILE --
  CHANNEL  LAST GRP#  USER            PROGRAM                        USER-ID
  105010      ----     LEW             INQUIRE.PR                       0
```

Figure 7-4. Sample Audit File (AUD3) (concluded)

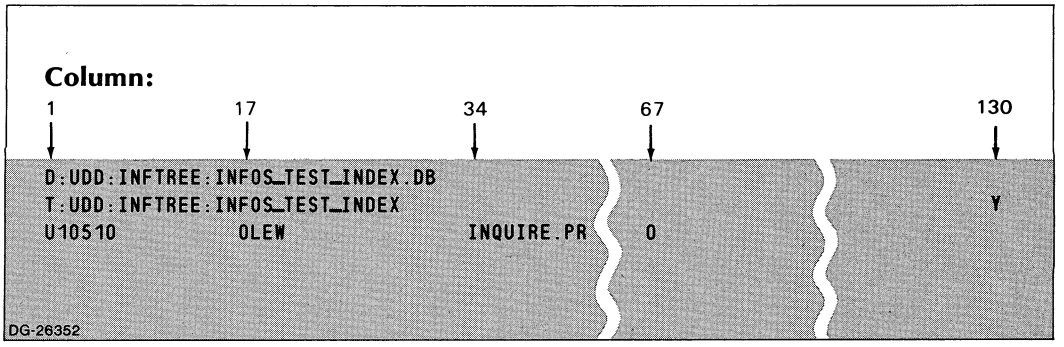


Figure 7-5. Sample Status File (STS3)

In the next example, we indicate that we want all instances of OPEN, CLOSE, and CHECKPOINT to be reported in FILE_A.

```
)X IRECOVER/AUDIT=FILE_A/OPENS ↓
```

When IRECOVER finishes, the status file information will be in the file IRCV.STATUS.04_01_84.08_22_58, since we did not specify a filename other than the default. (We assume, of course, that the recovery took place on April 1, 1984 just before 8:23 a.m.)

Replay Error Message

You will receive the following error message if IRECOVER gets an error when replaying a request:

```
REPLAY ERROR, SEE AUDIT LIST
```

You need not respond to this message. IRECOVER proceeds, and the audit file will contain a note of the error.

If IRECOVER reported any replay errors, it would ask you for confirmation before it would update the files at the end of the session. This is the message you would receive:

```
ERROR(S) ENCOUNTERED DURING RECOVERY
DO YOU WANT TO UPDATE FILES (Y OR N)?
```

Before answering this question you should look at the audit list and decide whether or not to update files in spite of the error(s). Always answer N (no) if you aren't certain whether you want the update to take place or not, because you can always run IRECOVER again later if you decide to update.

IRECOVER with Multiple Log Files

In some cases, you might want to restore a database from an old backup that would require several log files for recovery. (Note that we are referring to multiple log *files,* that is, one for each logging session; not multiple *volumes* of the same log file.)

For example, suppose you took a backup of a database, and then brought up INFOS and started request logging. You finished processing for the day, and came in the next day and started a new log file. Some time later, the system failed. You now have two log files with requests you want to recover.

If you have more than one log file, as you do in this example, you must process them in separate IRECOVER sessions. (But if one log file has more than one log volume, you must process all the volumes of this file in the same session.) Be sure to use the log files in the correct order; i.e., the order in which you made them. Remember that if you want to use the backup of the database that you took *before* you started logging, you must specify /RESTORE on the IRECOVER command line. Run the IRECOVER sessions one after another, and then continue processing.

IRECOVER Internal Data Structures

IRECOVER maintains a control block in memory for each of the following items that it finds on the log tape:

- Database file
- Index file
- User

IRECOVER creates the database and index file control blocks at the first opening of a file. At that time, IRECOVER opens the file to INFOS. It updates these file control blocks according to data it finds on the log files, such as checkpoints and the creation of new INFOS processes. IRECOVER never deletes these blocks. It will not close the INFOS files until the very end of the recovery session.

IRECOVER creates a user control block for each new user of a file. It deletes the block when the user closes the file.

Replay and Timestamp Logic

The INFOS II system records certain time-related data in a user data area (UDA). (A UDA is an area containing data associated with a file; see the *Advanced Operating System (AOS) Programmer's Manual* for more information.) The INFOS database UDA contains the following information:

- The date and time of creation of the last INFOS II process that opened the database.
- The date and time of the last checkpoint.

We call this time-related data the *timestamp* of the database.

When INFOS II connects to COMLOG for the first time, it logs a record that contains the date and time — the INFOS INIT timestamp.

Whenever you open a file, the "open" record that is logged contains the same information as the database UDA. In addition, all requests are logged with the date and time issued.

IRECOVER uses the following logic when it checks a database it is opening. If everything checks out normally, IRECOVER will put the file in REPLAY mode, and will replay all modifying requests that were successful when you originally issued them. It does not replay CLOSE or CHECKPOINT requests.

1. Date and Time of Creation
   IRECOVER expects the file on the disk to have the same date and time of creation as that of the logged file. If it doesn't, then IRECOVER will not put the file in REPLAY mode. The status file will report the status as ***INDEX CREATE TIME MISMATCH***.

2. INFOS INIT Timestamp
   IRECOVER expects the file on the disk and the logged file to have the same INFOS INIT timestamp. If they don't, one of the following situations will occur:

Situation	Message and Status
UDA timestamp > LOG timestamp	***WAITING FOR PROPER INFOS SESSION*** Not in REPLAY mode. You can update this file later if you find the correct INFOS session.
UDA timestamp < LOG timestamp	***NO_REPLAY*** This second situation occurs if you restore the database from a previous backup (i.e., it was not the version of the database that was open to INFOS when the system failed). IRECOVER will accept this condition and replay your requests only if you used the /RESTORE switch on the IRECOVER command line. Otherwise, it will read the entire log tape and make an audit file, but it will not replay the requests.

3. Checkpoint Timestamp
   IRECOVER expects the date and time of the last checkpoint recorded in the database file UDA to be the same as the logged checkpoint timestamp. If they are the same, IRECOVER will put the file in REPLAY mode. Otherwise, one of the following situations will occur:

Situation	Message and Status
UDA checkpoint < LOG checkpoint	*MISSING UPDATES* IRECOVER will not set REPLAY mode. It will never attempt to recover this database, since there might be some updates missing between the disk version of the data and the version you opened to modify. This is why it is important that you do not change file modes between making a backup and starting a new logging session.
UDA checkpoint > LOG checkpoint	*WAITING FOR CHECKPOINT* If IRECOVER subsequently finds a checkpoint record on the log file that matches the one in the UDA, it might then update the status to REPLAY mode.

4. Errors
   If the system returns any error when IRECOVER opens the file, the file will be in NO_REPLAY mode with an error code status.

Recovery Procedures After a System Failure

The recovery procedure differs depending on which recovery option you use on a file. Regardless of which you use, however, you need to restart the INFOS II system. If a disk failure occurs, you also must restore any corrupted databases from backup.

If you were using request logging, follow these steps to recover after a system failure:

1. Run ESD (emergency shutdown).
2. Write an end-of-file message to the log tape, if you need to (see information below).
3. Reboot the system.
4. Restart INFOS II (but do not enable consoles).
5. Run IRECOVER with the log file of the day.

See Appendix G, "Operating the INFOS II process," for information on how to restart INFOS II. Refer to the *Advanced Operating System (AOS) Operator's Guide* for information on running ESD and rebooting the system.

Writing an End-of-File Message on a Log Tape

If the system were to fail when you had request logging enabled, you might have to write an end-of-file message on your log tape. (COMLOG writes this message itself at the end of a log file that terminates normally.) If you do not write an end-of-file message, you can still use COMUNLOG to recover the logged requests, but it will expect to find the end-of-file message at the end of your logged requests. Instead, it will find "nonsense" data from previous uses of the tape, and it will send you an error message saying it has found bad data. We recommend that you avoid this error by writing the end-of-file message to your tape. This section explains how to do it. (If you were logging to disk rather than tape, you don't need to follow this procedure.)

CAUTION: If the system terminated as a result of a power failure, or if for any other reason the tape lost vacuum or was rewound, you must *not* write an end-of-file message. If you do, you could lose the entire log file. Instead, use the tape for recovery as usual, and COMUNLOG will unlog the requests until it reaches the "nonsense" data at the end.

Depending on what type of machine you are using, you must enter the end-of-file message with instructions from your terminal, or by flipping switches on the computer console. We explain both procedures below.

Using the System Control Processor (SCP) Operator's Console

Use the following procecure to write the end-of-file message when you are using an MV-class ECLIPSE, S120, or later machine.

CAUTION: Be sure the system is not up when you follow this procedure.

Your Action	Response
) RESET ⤸	(I/O Reset command.)
) 0/	Displays the current contents of location 0.
) 61122 <CR>	Places this value in location 0. Displays the current contents of location 1.
63077 ⤸	Places this value in location 1.
) 0/	Displays the new contents of location 0: *0 061122*
<CR>	Displays the new contents of location 1: *1 063077*
) EXAMINE AC 0 ⤸	Displays the current contents of accumulator 0.
000060 ⤸	Places this value in accumulator 0.
) START 0 ⤸	Writes end-of-file message to tape. The tape jumps as the message is written.

Using the DESKTOP GENERATION™ or ECLIPSE C/30 Computer

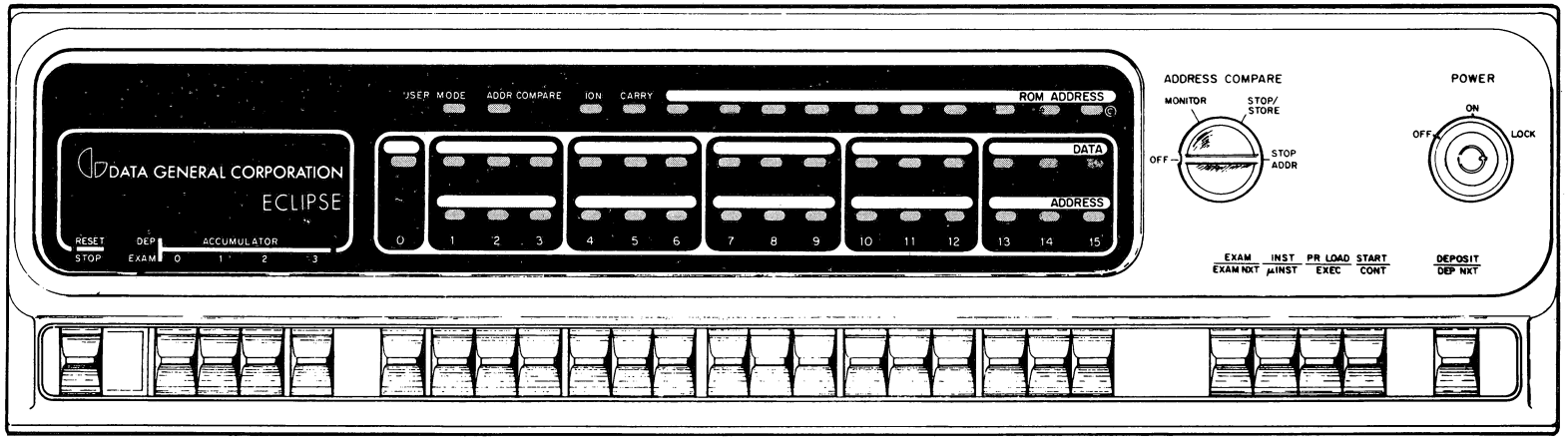
If you were logging to a cartridge on a DESKTOP GENERATION or an ECLIPSE C/30 computer, and the system failed, use the following procedure to write the end-of-file message.

CAUTION: Be sure the system is not up when you follow this procedure.

Your Action	Response
) I ⤸	(I/O Reset command.)
) 0/	Displays the current contents of location 0.
61122 <CR>	Places this value in location 0. Displays the current contents of location 1.
63077 ⤸	Places this value in location 1.
)0/	Displays the new contents of location 0: *0 061122*
<CR>	Displays the new contents of location 1: *1 063077*
) 0A	Displays the contents of accumulator 0.
000060 ⤸	Places this value in accumulator 0.
) 0R	(Resume command) Writes the end-of-file message to the cartridge.

Using the ECLIPSE Front Panel Switches

Refer to Figure 7-6 to see the location of the switches on the computer console of C350, S140, or earlier machines.



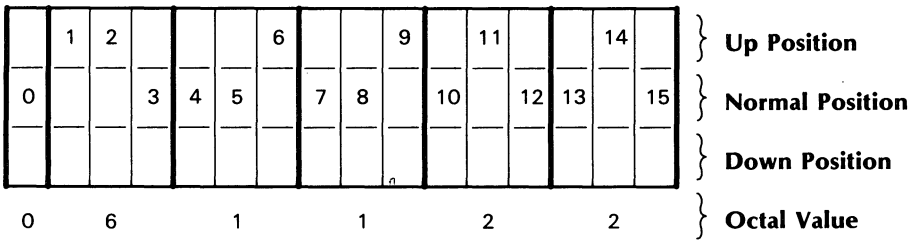
DG-00545

Figure 7-6. Sample ECLIPSE-Line Computer Console

We have outlined below the steps you must take to write an End-of-File message on this type of console.

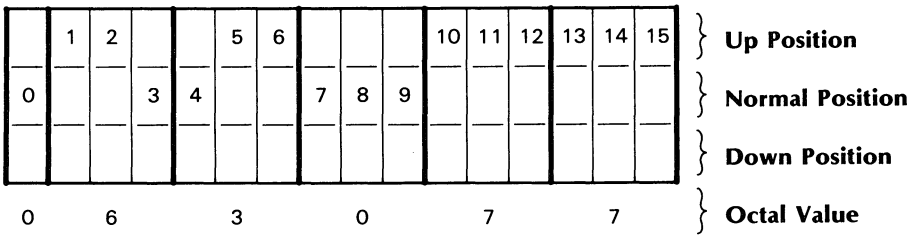
CAUTION: Be certain that the system is not up when you follow this procedure.

1. Push the RESET/STOP switch at the far left of the console up to the RESET position.
2. Turn the ADDRESS COMPARE dial to the MONITOR position. This enables you to examine your input to be sure that it is correct.
3. Check to be sure that all the switches are in the normal, resting position, as shown in Figure 7-6. The switches in this position indicate that you are at memory location 0.
4. Push up the examine (EXAM/EXAM NXT) switch. The current contents of memory location 0 will display in the DATA line. The illuminated numbers indicate which bits are on.
5. Set the numbered switches to read the octal value of 061122. This number is a driver instruction, which you will load into location 0. To enter the value correctly, push up switches 1, 2, 6, 9, 11, and 14, as shown in the following diagram.



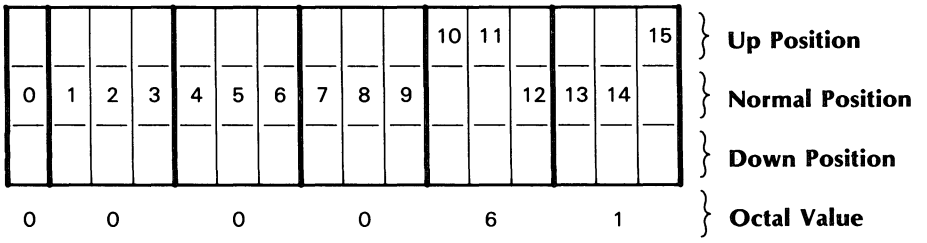
ID-01098

6. When you have set the switches correctly, deposit the values by flipping up the DEPOSIT/DEP NXT switch on the far right side of the control panel.
7. Return all the switches to their normal positions, as shown in Figure 7-6.
8. Examine the values you deposited in location 0 by pushing up the EXAM switch. Be sure the correct lights are illuminated on the DATA line of the console.
9. Set the address to memory location 1 by pushing up switch 15. (This gives you the octal value 1.)
10. Push up the EXAM switch to display the current contents of location 1.
11. Return switch 15 to its normal position.
12. Enter the octal value of 063077 -- a HALT instruction. To do this, push up the following switches: 1, 2, 5, 6, 10, 11, 12, 13, 14, and 15. We picture this concept below.



ID-01099

13. Deposit this value by pushing the DEPOSIT switch up.
14. Return the switches to their normal positions (see Figure 7-6).
15. Rotate the ADDRESS COMPARE dial to the STOP/STORE position to see the contents of the accumulator.
16. Examine the contents of accumulator 0 by pushing its switch down. (The switch is the leftmost of the four accumulator switches.)
17. Return this switch to its normal position, and enter the octal value of 00006X, where X is the tape unit number. For example, suppose we are at tape unit MTA1. We will push up the following switches: 10, 11, and 15. Below is our representation of this:



ID-01100

18. Deposit this value by flipping the 0 accumulator switch up.
19. Be sure your tape is ready on the tape unit you specified in the accumulator. Remember, if the tape has been rewound, *do not* write the end-of-file message to it — instead remove the tape and unlog the logged requests.
20. Write your end of file message to tape by pushing up the START/CONT switch to the START position. (This switch is located on the far right of the console, next to the DEPOSIT/DEP NXT switch.) You will see the tape jump as the message is written.

Restore Your Databases

If a disk failure occurs, then you must restore any databases that have been corrupted. We recommend using DLOAD to load the latest archival dump and the last incremental dump of the database(s) onto the system.

Now you are ready to continue with the recovery process. Use IRECOVER with your logged files.

INFOS II Failure

The following steps outline what to do in the event of an INFOS II failure.

1. Enter CONTROL @COMLOG STOP.
2. Terminate INFOS II users and disable consoles from logging on.
3. Restart INFOS II.
4. Bring up COMUNLOG.
5. Run IRECOVER with the log file of the day.

Hardware Failure

In the event of a hardware failure, follow the steps listed below.

1. Restore a valid backup of the disk pack(s) with the most recent version of the data.
2. Collect all necessary logfiles.
3. Bring the system and INFOS II up again.
4. Bring up COMUNLOG.
5. Run IRECOVER with the log file of the day.

Operational Considerations

We highly recommend that you observe the following cautions:

- Avoid switching logging on and off for a database while COMLOG is running.

If you were to put yourself in this situation, it might be impossible to recover from a database backup made before the log file was started. IRECOVER would not stop at the point where logging had been turned off. Remember that requests must be replayed in the order in which they were initially performed, or the recovery will not work correctly.

If you find it necessary to switch modes, you should make a backup. You *must* make the backup *after* you have changed the file mode back to request logging. You should not change the file mode between the backup and the next open of the file in logging mode, or IRECOVER might not function correctly.

- Don't move a database to be recovered into another directory on IRECOVER's searchlist. IRECOVER refers to databases by the full pathnames it finds on the log tape. If you move a database, IRECOVER won't find it.

Recovery Comparisons

The following sections compare recovery procedures for INFOS II files in Standard File Mode and those in Differential File Mode (without request logging). While this information points out the better security you have with DF mode, remember that SF mode provides greater system speed. We offer these comparisons for your information, to aid you in choosing a recovery option.

Standard File

After a system failure, follow these steps to recover your INFOS II file in Standard File Mode:

1. Run IVERIFY.
   - If IVERIFY accepts the INFOS II file, you might still have lost some requests. If so, you will have to re-enter them.
   - If IVERIFY does not accept the INFOS II file, restore it from your backup tapes and then re-enter all the changes made since you made the last backup tape.
2. Restart all programs that were running at the time the system failed.

Differential File (Failure Occurred Between Checkpoints)

If you were using Differential File Mode without logging and the system failed, but not during a checkpoint, follow these steps:

1. Re-enter changes made to the file since the last checkpoint operation.
2. Restart all programs that were running at the time the system failed.

The advantages of differential files over standard files in this case are:

- Minimal loss of data.
- Minimal recovery time.

Differential File (Failure Occurred During a Checkpoint)

If you use Differential File Mode and the system fails during a checkpoint, do the following:

1. Run the CHECKPOINT utility on the INFOS II file.
2. Restart all programs at the point where they left off at the time the system failed.

The advantages of differential files over standard files in this case are:

- No loss of data.
- Minimal recovery time.

With Differential File Mode, you will not have to wait for IVERIFY to restart your programs if the system fails. Standard File Mode and Modified Page Flush, however, would require that you run IVERIFY.

Although Differential File Mode doesn't require that you run IVERIFY to validate your file, you must run it periodically to ensure file integrity.

Remember that Differential File Mode will require you to restart the CHECKPOINT utility if the system fails during a checkpoint. The INFOS II system will refuse access to the INFOS II file if you need to restart the checkpoint.

End of Chapter

Chapter 8
The INFOS II Scientific Language Interface

AOS INFOS II has an interface for several of the scientific languages. In this chapter, we describe how to use the INFOS II interface to write programs in Data General's FORTRAN 5, FORTRAN 77, DG/L, and PL/I languages. In addition, we provide some information on using AOS INFOS II with Business BASIC. You can find all of the Business BASIC statements that you use with INFOS II in the Business BASIC manuals.

We do not attempt to teach the scientific languages in this manual; we just show you how to use the INFOS II system's capabilities when working within the languages. For more information on the details of each language, you should refer to the appropriate manual:

- *FORTRAN 5 Programmer's Guide (AOS).*
- *FORTRAN 5 Reference Manual.*
- *FORTRAN 77 Reference Manual.*
- *FORTRAN 77 Environment Manual (AOS).*
- *DG/L™ Language Reference Manual.*
- *DG/L™ Runtime User's Manual (AOS).*
- *PL/I Reference Manual.*
- *PL/I-INFOS® II Interface Reference Manual (AOS & AOS/VS).*
- *Business BASIC Commands, Statements, and Functions (AOS/VS, AOS, RDOS, DOS).*

Note that although we discuss the PL/I interface in this chapter, you will find more detailed information in the reference manual noted above.

The way in which you use AOS INFOS II with Business BASIC is substantially different from how you use it with the other languages we discuss in this chapter. For this reason, we present its information separately from that of the other languages. We discuss how to use INFOS II with Business BASIC at the end of the chapter.

For the FORTRAN 5, FORTRAN 77, DG/L, and PL/I languages, we discuss the following topics in this chapter:

- Using the interface
- Components of the interface
- The interface subroutines
- Specific considerations for each language

NOTE: We use the word *subroutine* throughout this chapter to refer both to FORTRAN subroutines and to PL/I or DG/L procedures.

If you have never used the INFOS II scientific language interface for the FORTRAN 5, FORTRAN 77, DG/L, or PL/I languages, we suggest that you read this entire chapter before attempting to do so. We have included numerous examples to help you. Once you become familiar with the interface, you will find the tables and subroutine descriptions useful for reference.

Table 8-1 lists the interface subroutines and the INFOS II commands to which they correspond.

Table 8-1. Corresponding INFOS II Commands and Interface Subroutines

INFOS II Command	Interface Subroutine	Comments
CLOSE	ICLOSE	
DEFINE SUBINDEX	ISDEFINE	
DELETE	IDELETE ILDELETE	Physical. Logical.
DELETE SUBINDEX	ISDELETE	
LINK SUBINDEX	ISLINK	
OPEN	IOPEN	
READ	IKEYREAD IRELREAD	Keyed access. Relative access. Both perform a Read and a Retrieve Key.
REINSTATE	ILDELETE	Use appropriate parameters to specify reinstate.
RELEASE LOCKS or POSITION	IRELEASE	Use appropriate parameters to specify locks, position, or both.
RETRIEVE HIGH KEY	IHIREAD	Performs both a Read and a Retrieve Key.
RETRIEVE KEY	IKEYREAD IRELREAD	Use appropriate parameters to specify key only.
RETRIEVE STATUS	None	Perform a Read operation and suppress everything.
RETRIEVE SUBINDEX DEFINITION	IRETDEF	
REWRITE	IREWRITE	
WRITE	IWRITE	
None	IIGET	Gets the last information retrieved.
None	IISET	Sets up default values.

Using the Interface

The INFOS II scientific language interface contains numerous subroutines and variables, but this does not mean it has to be confusing or difficult to use. In this section, we describe the general procedure for writing programs using the interface. We then introduce the subroutines and variables briefly, and describe them in detail in following sections. Bear in mind that this procedural description is general because we are dealing with several different programming languages at the same time.

We outline below some basic steps you can follow to write a program with the INFOS II interface.

1. Flowchart or outline your program steps, leaving a gap or comment everywhere that you want to use an INFOS function.
2. Identify the INFOS II subroutines you need to use by referring to Table 8-1, and subsequently reading the description for each subroutine you have chosen. (We describe the subroutines in detail, later in this chapter.)

3. Identify the variables that are appropriate for each subroutine you want to perform, by using the section of this chapter dealing with variable descriptions.
4. Verify that each variable you have chosen is allowable for the subroutine in which you intend to use it, by referring to Table 8-11 (shown in the "Interface Subroutines" section, later in this chapter).
5. Write your program, including each subroutine in the appropriate position. We suggest that you use the subroutine format we show in the examples, so you can easily recognize the function of each subroutine when you look at your program code.
6. Set up an error-handler that will stop program execution and identify any error encountered during the INFOS subroutines.
7. Write the declaration section of your program. Be certain to do the following:
   - Include the correct source language parameter file.
   - Declare an interface array of the correct length for your application.

   (We describe these procedures in the following sections.)
8. Compile your program.
9. Link your program, including the common language runtime environment library for INFOS II (CLREINF.LB) and the ICALL.OB subroutine.
10. Be sure to create your INFOS II database *before* you run your interface program. Use the ICREATE utility to do so.

Naturally, as you become more familiar with the INFOS II interface for your application language, you will be able to skip some of the above steps, and write the subroutines directly into your program as you go along. However, this procedure we have described should be helpful when you first use the interface.

Components of the Interface

The INFOS II language interface consists of two parts:

- A library of subroutines that perform INFOS II functions.
- A source language parameter file, which defines a group of program constants that you pass to the interface subroutines.

We will discuss the subroutines later in this chapter.

Source Language Parameter Files

At the beginning of every program you write using the interface, you must include the parameter file for your source language. You will find a copy of these parameter files in Appendix B. Each parameter file is named IIPAR.ext, where ext is FR (for FORTRAN 5), F77 (for FORTRAN 77), DG (for DG/L), or PL1 (for PL/I).

Look at any version of the source language parameter file, and you will notice that it is divided into two sections. The first provides an octal value for each interface variable (we describe these variables in detail in a following section). The second section defines all the possible parameter values for these variables. You should refer to this appendix when choosing your subroutine variables, so that you will select the appropriate parameter values for the variables you use.

As we discuss the interface variables, you might also notice that the parameters listed in Appendix B restrict you to four key levels in your INFOS file. If you need more levels than this, you should include in your program the Extended Source Level Parameter File for your specific language. The name of this file is IIPAR.EXTRA.ext, where ext is FR, F77, DG, or PL1. We have placed these parameters in separate files because most applications don't require them.

Please note that the files in Appendix B were current when this manual was printed; however, you should print out a copy of these files as they appear on your release tape to make sure that you are using the most current versions.

To use the interface, you must compile the source language parameter file into your program, and link the subroutine library CLREINF.LB with your program. Commonly, both files are located in the utilities directory (:UTIL). The "Specific Language Considerations" section of this chapter provides the details on how to do this for each language.

CAUTION: To ensure the integrity of the interface, you should never redefine any of the names that appear in the parameter files.

The ICALL Runtime Routine

In addition to the library and the parameter files, your program must contain a small subroutine called ICALL.OB in order for the INFOS II system to service it. This routine handles the interprocess communication between your process and the INFOS II process.

ICALL.OB is distributed along with the INFOS II system, but you determine where to place it on your disk. If you edit ICALL.OB into the system library, URT.LB, using the Library File Editor (LFE) utility, then you need not name it in the LINK command line for all your programs, since URT.LB is searched at the end of all LINK command lines. If you do not edit ICALL.OB into URT.LB, you should place it in :UTIL and include it on your LINK command line, along with the common language runtime environment library. For example, a FORTRAN 77 LINK command line might look like this:

```
F77LINK MYPROGRAM CLREINF.LB ICALL
```

If the Linker sends you the message *UNDEFINED EXTERNAL ?INFS* , it means that it did not find ICALL.OB; you should make sure that it is located in either URT.LB or :UTIL.

Interface Arrays

In the steps listed earlier, we mentioned that you need to declare at least one interface array when you write a program with the INFOS II scientific language interface. We discuss interface arrays in more detail here.

You must allocate space in your program for one interface array for every INFOS II file that the program will use. The interface uses this array to store interface variables and other processing information. After you declare this array, you only refer to it in your interface subroutines. You pass the array name as the first argument in each subroutine of your program.

Size of the Array

To determine the appropriate length for your interface array, first decide the maximum number of index levels (mlev) that you need for your INFOS II file. Then use that number in the following formula:

Interface Array Length = 32 + (12 * mlev)

For example, if your file needs only one level of indexing, you must declare an array length of 44, according to the formula:

32 + (12 * 1) = 44

For a two-level file, you need a 56-word array:

32 + (12 * 2) = 56

CAUTION: Be sure to specify the same number of maximum index levels here as you do in the IOPEN interface subroutine. This is important because IOPEN first fills out the interface array with zeros, using the above formula. If you have not allocated enough space when declaring your array length, IOPEN could overwrite some of your program variables.

The interface will not work if you try to allocate your interface array on the stack of each subroutine that must use it. We suggest that you declare your interface array(s) at the beginning of your main program as an integer array with subscripts from 1 to mlev (the length you calculated with the above formula). For example, suppose you are writing a program to build an INFOS II file that will have two index levels. Your declaration for the integer array called IARRAY will look like this:

FORTRAN 5 or FORTRAN 77:
```
INTEGER IARRAY (56)
```

PL/I:
```
DECLARE IARRAY(56)    FIXED BINARY;
```

DG/L:
```
INTEGER ARRAY IARRAY [56];
```

You will then pass IARRAY as an argument to your subroutines. Alternatively, you could reference it as a COMMON or EXTERNAL variable in the subroutines. (See the appropriate language manual for details on how to declare arrays as global.)

Interface Variables and Subroutines

The interface variables represent the INFOS II concepts that you read about in earlier chapters. You manipulate them in order to process INFOS II files. As we noted above, the interface variables all reside in your interface array, and you access them by including them as arguments in your subroutine calls.

Each subroutine contains arguments. The first argument is always the name of the interface array for the INFOS II file, and the last is the *error status variable,* an integer variable to which the interface returns the error status at that point in the program. Our examples in this chapter always refer to these two arguments as IARRAY and IER, respectively. When we use the terms generically, for instance to show the syntax of a subroutine call, we use lowercase: iarray and ier.

NOTE: In FORTRAN 77 subroutines, you must include an additional argument, ENDLIST, after your error status variable.

The Error Status Variable

The error status variable is mandatory. If you do not include it, the interface will attempt to write error status information into the last argument you supply in the INFOS subroutine, and an unfavorable condition will result. You might overwrite a program variable, thus corrupting it, or you could cause a *Write Protection Trap.* This means that the interface will attempt to write the error information where it is not allowed, and your program will terminate abnormally. Avoid these conditions by supplying an error status variable in every subroutine.

If the error return value is IONORM, it means that the subroutine has completed normally and your program can continue. If the variable is assigned any other value, then an error has occurred during processing that warrants your attention. Your program should include some kind of error-handler so that you can address these problems as they arise. You can include the error-handler as a subroutine that you call after each INFOS subroutine, or write one similar to those used in the sample programs listed in Appendix C. We list all of the INFOS II error messages in Appendix A, and include the actual interface errors near the end of this chapter, in Table 8-13.

Argument Pairs

In between the iarray and ier arguments are *argument pairs.* The first argument of an argument pair is an interface variable. The second argument is one of the following:

- A parameter (as listed in the source language parameter files).
- A constant.
- A location.

Which of these is the second argument depends on which subroutine it is included in and which variable it is paired with. Note that if the second argument is a character string constant, you must enclose it in quotation marks.

NOTE: If you use string constants with the language interface, we recommend that you use them with care. We especially caution you against using string literals in the IISET subroutine if you are calling it from another subroutine or procedure. If you do this your variables might not remain set to the literals that you specify. This is because different languages have different ways of dealing with literals and variables in subroutines and procedures.

In general, the interface subroutines will place the value of the second argument of the pair into the variable you include as the first argument. For example, if you have an IWRITE subroutine with an argument pair that consists of IIREC, ADDRESS, then IWRITE will place whatever data is stored in the variable ADDRESS into the data area IIREC. An exception to this rule is the subroutine IIGET, which does just the opposite. It takes the information stored in the first argument of the pair and places it into the variable that is the second argument. The purpose of this will become clearer in the next few sections, where we discuss the subroutines and interface variables in more detail.

It is important to note that using an interface variable in a subroutine changes its value only for the duration of that subroutine. If you want to change the value of a variable for longer than that duration, you must change it with the subroutine IISET. In addition, some interface variables can only be specified in the IISET subroutine, so if you want to specify their values for another subroutine, you must first set them with IISET.

The Interface Variables

There are nine classes of interface variables. We describe these classes briefly in Table 8-2. We discuss each class and the variables it includes in the next few sections.

Table 8-2. Classes of Interface Variables

Variable Class	Description
Data Area Variables	Indicate a data area in which to store data that is contained in a program variable, so that it can be used in a subroutine.
Length Variables	Indicate the maximum value for a key or record length, or how much of the allocated data area you want to use.
Mode Variables	Select various INFOS II processing options, such as locking and type of access. (The options are listed in the source language parameter files).
Multilevel Variables	Allow interface variables to be used for various levels in multilevel files.
Status Variables	Access information returned on a Read operation (IKEYREAD or IRELREAD), and values such as occurrence numbers and data record feedback.
Optimized Record Distribution Variables	Specify or find the merit factor of an entry. You can use these only if you specified optimized record distribution at file creation.
File Open Variables	Specify certain file open options, such as maximum number of record locks, exclusive use, or read-only open. You can use these variables only with the IOPEN subroutine.
Subindex Definition Variables	Define subindex parameters, such as maximum key length, partial record length, root node size, and allowing subindexes or duplicate keys. You can use these variables only with the ISDEFINE subroutine.
Link Subindex Variables	Specify interface variables to be used in reference to the destination key during a Link Subindex operation (ISLINK).

Data Area Variables

Data area variables specify a certain data area to which the INFOS II system passes data contained in the program variable you specify. The data area variable is the first argument of an argument pair in a subroutine; the program variable is the second argument of the pair. The interface passes the address indicated by the data area variable to the INFOS II system. The system uses this area and its associated length (see the following section on length variables) as byte sequences. The interface then inputs or outputs these sequences to or from an INFOS II file.

All of the data area variables are initially undefined; you cannot perform an IIGET on them until they contain data. If you perform a Read operation, for example, the values of the data area variables will change. The system will store the values it reads from your file into the program variables you specify as second elements in your argument pairs.

You should note that the interface subroutines will change the values only of program variables associated with the data area variables; they will not do so for any other interface variable. (The only exception to this is that the IIGET subroutine changes the values of *all* the variables you pass in argument pairs.)

Table 8-3 lists the data area variables.

Table 8-3. The Data Area Variables

Variable	Indicates
IIREC	Database Record Area.
IIKEY*[n]*	Key Data Area, where *n* indicates a key level between 2 and the maximum number allowed in the file. For the uppermost key level, the *n* value of 1 is assumed; you do not include it. The *n* value increases as you move down into deeper index levels.
IIPREC	Partial Record Data Area.

Example (DG/L)

In this example, we use the IKEYREAD subroutine to read a data record using keyed access.

```
IKEYREAD  (     IARRAY,
                IIKEY, CUSTOMER_NAME,
                IIREC, CUSTOMER_RECORD,
                IIPREC, PHONE_NUM,
                IIKLEN, 20,
                IER     );
```

With this subroutine, we instructed the INFOS II system to do a keyed read. The name of the level 1 key we requested is stored in the data area CUSTOMER_NAME. The record associated with this key is stored in CUSTOMER_RECORD. The partial record is stored in PHONE_NUM. (Note here that IIKLEN indicates that the key length is 20. We discuss the length variables in the following section.) If this subroutine is included in an interactive program and we want the record displayed on the screen, we next would issue a TYPE statement in the DG/L program, specifying the variable CUSTOMER_RECORD.

Important Language Considerations

DG/L

If you are coding in DG/L, you *must* use STRING type variables for data area variables. For example, a declaration and a subroutine could look like this:

```
STRING (80) RECORD;
IISET ( IARRAY,
        IIREC, RECORD,
        IER     );
```

FORTRAN

If you program in FORTRAN, we suggest using integer arrays for data area variables. For example, a FORTRAN 77 program segment might look like this:

```
      INTEGER REC_1(40)
      CALL IISET (    IARRAY,
     +                IIREC, REC_1,
     +                IER, ENDLIST    )
```

PL/I

In PL/I, we suggest using the CHARACTER (n) ALIGNED designator for data area variables. For example:

```
DECLARE RECORD_ONE CHARACTER (50) ALIGNED;
CALL IISET (    IARRAY,
                IIREC, RECORD_ONE,
                IER     );
```

For other PL/I possibilities, see the section dealing with specific language considerations, later in this chapter.

Length Variables

The length variables have the following two functions:

- Tell the INFOS II system the maximum value for a length.
- Indicate how much of a data area you want to use.

For all languages, length variables must be single-precision integers. Table 8-4 lists the length variables.

Table 8-4. The Length Variables

Variable	Indicates
IIRLEN	Data Record Length.
IIKLEN/*n*/	Key Length, where *n* indicates the key level. You need not specify *n* when it is level 1. The *n* value increases as you move down into deeper index levels.
IILRLEN	Limit Data Record Length.
IIPRLEN	Partial Record Length.
IIMRLEN	Maximum Data Record Length.
IIMKLEN	Maximum Key Length.

The INFOS II system uses the lengths you specify in the length variables with the data area, to define data and keys. For example, a FORTRAN 5 subroutine might look like this:

```
      CALL IISET  (   IARRAY,
     +                IIKEY, LEV_1_KEY,
     +                IIKLEN, 10,
     +                IIREC, RECORD_ONE,
     +                IIRLEN, 78,
     +                IER     )
```

In this case, the system resets the current values of IIKEY, IIKLEN, IIREC, and IIRLEN to the values specified in the second element of each argument pair. Therefore, the key length becomes 10 bytes and the record length becomes 78. In addition, the key data area is set as LEV_1_KEY and the data record area as RECORD_ONE.

The length variables are undefined when you open the file. You can set them either with the IISET subroutine or with a Read operation (IKEYREAD or IRELREAD). Note that any values you set with IISET remain set until a subsequent issuing of IISET, whereas any values you specify with a Read request will be set only for the duration of that subroutine.

You can retrieve the value of any of the length variables with the IIGET subroutine.

The system defines the variables IIPRLEN, IIMRLEN, and IIMKLEN when you open a file with IOPEN, using the values you assigned for these parameters at file creation. You can use the variable IIPRLEN only with the IIGET and IISET subroutines, and you can include IIMRLEN and IIMKLEN only with the IIGET subroutine. Therefore you can find out the set values of maximum record length and maximum key length, but you cannot change them once the file has been created.

If you want to read only part of a data record, you can limit the record length in your IKEYREAD or IRELREAD subroutine. Use IILRLEN and specify the number of bytes you want to read. Note that if you use the IILRLEN length variable, you must also use the IIAMODE access mode variable to indicate that you are limiting the record length on the read operation.

Mode Variables

Use the mode variables to select various INFOS II processing options, which are listed in your source language parameter file (see Appendix B). If you include a mode variable in an argument pair of your subroutine, you must choose one of the parameters listed in the parameter file as the second element of the argument pair. Any other value will result in an error. Table 8-5 lists the seven mode variables, and we explain them in more detail in the following sections.

Table 8-5. The Mode Variables

Variable	Type	Default Value
IIAMODE	Access Mode	Access key, data record, and partial record.
IIDMODE	Logical Deletion Mode	Reinstate.
IIKMODE/*n*/	Keyed Search Mode, where *n* indicates the key level.	Exact key.
IILMODE	Locking Mode	No locking.
IIPMODE	Current Position Mode	Set current position.
IIRMODE	Relative Processing Mode	Forward motion.
IISMODE	Nondefault Search Mode	Keyed.

IIAMODE Options

Use IIAMODE to specify the type of access you want for a read, write, or rewrite operation (IKEYREAD, IRELREAD, IWRITE, or IREWRITE). You can also set IIAMODE with the IISET subroutine so that its value will remain as you set it for longer than the duration of one subroutine.

Note that for any read request you make, the system also performs a retrieve key operation. If you want to perform a retrieve key without a read, you need to use the key only option that we mention below.

You have the following access mode options with IIAMODE:

Full Access	Gains access to the key, the entire data record, and the partial record. This is the default.
Suppress Database	Gains access to the key and partial record only.
Suppress Partial Record	Gains access to the key and data record only.
Invert	Performs an inverted operation. You must include the correct data record feedback. We discuss inversion in greater detail in the "Status Variables" section.
Key Only	Suppresses both the data record and the partial record. Use this option with IKEYREAD or IRELREAD if you just want to retrieve the key rather than read the record.

Limit Record Length	Gains access only to part of the data record. You must also use the length variable IILRLEN to specify the number of bytes you want accessed.
Limit Record Length/ Suppress Partial Record	Gains access only to part of the data record, and none of the partial record. You must use the length variable IILRLEN to specify the number of bytes you want returned.

If you don't specify one of these options, you gain access to the key, the data record, and the partial record. You can use IIAMODE in the following subroutines:

IISET	IWRITE
IKEYREAD	IREWRITE
IRELREAD	IRETDEF
IHIREAD	

Be sure to refer to your source language parameter file to find the correct way to code these options.

IIAMODE Example (FORTRAN 77)

In this subroutine, the INFOS II system gains access to the key whose 4-byte name is stored in the data area PART_NUM, and reads the first 20 bytes of its record into the data area PART_RECORD. The IIAMODE variable instructs the system to limit the data record length to 20 bytes and to suppress the partial record so that it is not read.

```
      CALL IKEYREAD ( IARRAY,
     +                IIKEY, PART_NUM,
     +                IIKLEN, 4,
     +                IIREC, PART_RECORD,
     +                IILRLEN, 20,
     +                IIAMODE, LIM SUPP PR,
     +                IER, ENDLIST    )
```

IIDMODE Options

Use IIDMODE to specify options on a logical delete. These options follow:

Logical Delete	Performs a standard logical delete. Flags the record and partial record as logically deleted.
Partial Record Logical Delete	Logically deletes the partial record only. The data record remains undeleted.
Data Record Logical Delete	Logically deletes the data record, not the partial record.
Reinstate	Reinstates the logically deleted data record and partial record.
Reinstate Partial Record	Reinstates the logically deleted partial record, but leaves the data record flagged as logically deleted.
Reinstate Data Record	Reinstates the logically deleted data record, but leaves the partial record logically deleted.

If you don't specify one of these options, the system will not perform a logical delete. Remember to refer to the parameter file for your source language so you will code the IIDMODE options correctly.

You can use the IIDMODE variable with the subroutines ILDELETE and IISET only.

IIDMODE Example (PL/I)

In this example, the system logically deletes the data record that is associated with the key stored in the data area NAME. This key is 10 bytes long. The key's partial record is not logically deleted.

```
CALL ILDEL  (   IARRAY,
                IIKEY, NAME,
                IIKLEN, 10,
                IIDMODE, INF_LD_DB,
                IER     );
```

IIKMODE Options

IIKMODE allows you to specify a keyed search using approximate, generic, exact, or duplicate keys. Use IIKMODEn in a multilevel request, where n specifies a key level other than the first. Your options follow:

Exact	Search for the exact key specified. The key cannot be a duplicate. This is the default search mode.
Approximate	Use the specified key in an approximate key search.
Generic	Use the specified key in a generic key search.
Duplicate	The specified key might be a duplicate. You can use this parameter only if you allowed duplicate keys when you defined the index or subindex.

IIKMODE is a valid option for the following subroutines:

IISET	IDELETE
IIGET	ILDELETE
IKEYREAD	ISDELETE
IRELREAD	ISDEFINE
IHIREAD	ISLINK
IWRITE	IRETDEF
IREWRITE	

IIKMODE Example (DG/L)

In this example, we write a key into the second key level of the INFOS file and its record into the database. We specify that the key might be a duplicate.

```
IWRITE  (       IARRAY,
                IIKEY, "NAME",
                IIKLEN, 4,
                IIKEY2, CO_NAME,
                IIKLEN2, 20,
                IIKMODE2, DUP_KEY,
                IINLEV, 2,
                IIREC, RECORD,
                IIRLEN, 124,
                IER     );
```

Note that we used the variable IIKMODE2, using the "2" to specify that it is the key level 2 at which we might have a duplicate. Had we forgotten to include the 2, then IIKMODE would have referred to the key at the first key level (NAME); if CO_NAME was a duplicate, we would receive an error message when we ran the program.

Refer to Appendix B to find the correct way to code the interface parameters in your language.

IILMODE Options

Use IILMODE to specify locking and unlocking for data records, partial records, or both. Your options follow:

Lock	Locks both the partial record and data record.
Lock Partial Record	Locks the partial record only. The data record remains unlocked.
Lock Data Record	Locks the data record only. The partial record remains unlocked.
Unlock	Unlocks both the partial and data records.
Unlock Partial Record	Unlocks the partial record and leaves the data record locked.
Unlock Data Record	Unlocks the data record and leaves the partial record locked.
No Locking	Neither the data record nor the partial record is locked. This is the default mode.
Release Locks	Releases locks, when included in the IRELEASE subroutine. This option is not valid in any other subroutine.

You can use IILMODE with the following interface subroutines:

IISET	IRELEASE
IKEYREAD	IHIREAD
IRELREAD	IRETDEF
IWRITE	IREWRITE

IILMODE Example (DG/L)

In this example, we lock both the data record and partial record, and rewrite them, using the data stored in the data areas DIF_RECORD and DIF_PART_REC.

```
IREWRITE  (     IARRAY,
                IIKEY, NUMBER,
                IIKLEN, 3,
                IIREC, DIF_RECORD,
                IIRLEN, 56,
                IIPREC, DIF_PART_REC,
                IISMODE, KEYED,
                IILMODE, LCK,
                IER     );
```

Be sure to refer to your source language parameter file for the correct way to code the IILMODE options in your application language.

IIPMODE Options

IIPMODE allows you to specify a current position mode on the key you access for the current subroutine. It has three options.

Set Position	Sets current position at the key accessed for the current operation. This is the default mode.
No Change	Maintains current position where it was before the current operation.
Release Position	Releases current position. You can only specify this option with the IRELEASE subroutine.

You can use IIPMODE in the following subroutines:

IISET	IDELETE
IIGET	ILDELETE
IKEYREAD	ISDELETE
IRELREAD	ISDEFINE
IHIREAD	ISLINK
IWRITE	IRETDEF
IREWRITE	IRELEASE

IIPMODE Example (FORTRAN 5)

In this example, we read down and forward, placing the 10-byte key we accessed into the data area NAME, and its data record into the data area ADDRESS_PHONE. We suppress the partial record with an access mode variable, and use IIPMODE with the parameter NO CHANGE so that the current position will remain where it was before we called this subroutine.

```
      CALL IRELREAD ( IARRAY,
     +                IIRMODE, DOWN FOR,
     +                IIKEY, NAME,
     +                IIKLEN, 10,
     +                IIREC, ADDRESS_PHONE,
     +                IIPMODE, NO CHANGE,
     +                IIAMODE, SUPP PR,
     +                IER     )
```

Refer to your source language parameter file for the correct way to code the IIPMODE options in your language.

IIRMODE Options

You must use IIRMODE if you do not use keyed access in a subroutine. You must choose a direction of relative motion within an index whenever you specify relative or keyed relative motion. Your options for relative motion are the following:

Forward
Backward
Down
Up
Down and forward
Up and forward
Up and backward
Static

The default direction of movement is forward. See the source language parameter file for your language to determine the correct way to code the direction of motion you want.

IIRMODE Example (PL/I)

In this example, we read backward in a NAME subindex, placing the record we access into the data area RECORD. We do not read the partial record because it is suppressed. Our current position must be in the key level 2 subindex before we call this subroutine, because we are only reading backwards, and not changing levels. We also set the current position on the key we access for this subroutine.

```
CALL IRELR  (   IARRAY,
                IIRMODE, INF_BACK,
                IIPMODE, INF_SET_POS,
                IIKEY2, NAME,
                IIKLEN2, 20,
                IINLEV, 2,
                IIREC, RECORD,
                IIAMODE, INF_SUPP_PR,
                IER     );
```

IISMODE Options

Use IISMODE to specify a search mode for a subroutine. Your choices are keyed, relative, or keyed and relative. The default is keyed. You can use the IISMODE variable with the following subroutines:

IKEYREAD	ISDEFINE
IWRITE	ISDELETE
IREWRITE	ISLINK
IDELETE	IHIREAD
ILDELETE	IRETDEF

IISMODE Example (FORTRAN 77)

In this example, we use relative and keyed positioning. First we position down and forward from the current position. Then we use keyed access to locate the key PIECE_NUM. We rewrite the record, using the information stored in PIECE_REC, and set the current position on the key we have accessed.

```
      CALL IREWRITE ( IARRAY,
     +                IISMODE, REL KEY,
     +                IIRMODE, DOWN FOR,
     +                IIKEY2, PIECE_NUM,
     +                IIKLEN2, 7,
     +                IIREC, PIECE_REC,
     +                IIPMODE, SET POS,
     +                IER, ENDLIST    )
```

Multilevel Variable

The only multilevel variable we discuss here is IINLEV. You use IINLEV to limit the number of levels in any multilevel processing request. Set IINLEV to the number of levels you want to traverse. IINLEV is initially set to 1; you can leave it at that setting if you do not need multilevel processing.

Some of the variables we have already discussed can also be considered multilevel variables, such as IIKEYn and IIKMODEn, where n is greater than 1. Since we deal with these variables in other sections, we will not discuss them here.

You can use IINLEV with the following subroutines:

IISET	IDELETE
IIGET	ILDELETE
IKEYREAD	ISDELETE
IRELREAD	ISDEFINE
IHIREAD	ISLINK
IWRITE	IRELEASE
IREWRITE	IRETDEF

IINLEV Example (PL/I)

In this example, we indicate that we want to traverse two index levels with IINLEV. The system performs a keyed search to the key in the second level that we have specified in CO_NAME. It then physically deletes the key and its record.

```
CALL IDELE  (   IARRAY,
                IIKEY, "COMPANY",
                IIKLEN, 7,
                IIKEY2, CO_NAME,
                IIKLEN2, 25,
                IINLEV, 2,
                IER     );
```

Status Variables

The status variables gain access to various types of information after a Read operation. They are undefined until after your first Read. You should note that every Read request you do with the scientific language interface also performs a Retrieve Key operation.

Table 8-6 lists the status variables.

Table 8-6. Status Variables

Variable	Indicates
IILEV	Current Subindex Level.
IIFLEV	Maximum Index Level. The interface returns this value when you open a file.
IIOCCH*/n/* IIOCCL*/n/*	Duplicate Key Occurrence Number (High). Duplicate Key Occurrence Number (Low). In both variables, *n* indicates the key level. At the top level (1), do not specify the *n* value. The *n* value increases as you move down into deeper index levels.
IIFDBKH IIFDBKL	Data Record Feedback (High). Data Record Feedback (Low).
IIDUP	Duplicate Key Flag.
IIDBDEL	Data Record Logical Delete Flag.
IIPRDEL	Partial Record Logical Delete Flag.
IIOVFLO	Record Length Limit Exceeded Flag.
IISIP	Subindex Present Flag.

You can use the occurrence number variables and the data record feedback variables with IISET. You need to do this only if you use duplicate keys or perform inverted write and rewrite operations. The remaining status variables can only appear in the IIGET subroutine; that is, you can find out their values, but you cannot change them.

In all languages, the status variables return a single-precision integer. Since occurrence numbers and data record feedback values are double-word values, they require two variables each (IIOCCH and IIOCCL for occurrence numbers; IIFDBKH and IIFDBKL for data record feedback). The IILEV and IIFLEV values indicate subindex levels from 1 to 32, but are always single-word integers.

The variables IIDUP, IIDBDEL, IIPRDEL, IIOVFLO, and IISIP return either a 0 or a 1 as a status flag. The flag indicates whether the given condition is true (1) or false (0). In FORTRAN subroutines, you can declare these variables as type LOGICAL.

Status Variables Example (DG/L)

In this example, we perform an IIGET on the key at the current position. We find out its data record feedback, its index level, if it is a duplicate key (although we do not ask for its occurrence number, and if it has a subindex defined below it.

```
IIGET ( IARRAY,
        IIFDBKH, HIGH_FEEDBACK_NUM,
        IIFDBKL, LOW_FEEDBACK_NUM,
        IILEV, INDEX_LEVEL,
        IIDUP, DUP_KEYS,
        IISIP, SUBIND,
        IER     );
```

Optimized Record Distribution Variables

You can only use the optimized record distribution variables if you selected optimized record distribution at file creation time. (See Chapter 10 for a complete description of optimized record distribution.) There are two variables in this group. We list them in Table 8-7.

Table 8-7. Optimized Record Distribution Variables

Variable	Indicates
IIRMF	Record Merit Factor.
IIVMF	Volume Merit Factor.

You can use IIRMF to specify the merit factor of a record when you use the IWRITE or IREWRITE subroutines. You can also set the record merit factor by using IIRMF with the IISET subroutine. You must specify a merit factor between 1 and 255. The default is 1.

Use IIVMF with the IIGET subroutine only. Issue the call for IIGET after you perform a Read operation, because the Read returns the merit factor of the volume in which it found the index entry.

You can also use the subindex definition variable IISPRLEN as an optimized record distribution variable. You can use it to define the merit factor of a subindex root node when you define the subindex. This involves special manipulation of IISPRLEN, which we discuss in the section on subindex definition variables.

IIRMF Example (FORTRAN 5)

In this example, we write a 45-byte record for a key in the second key level. We specify a record merit factor of 12.

```
      CALL IWRITE  (  IARRAY,
     +                IIKEY, "DATE",
     +                IIKLEN, 4,
     +                IIKEY2, MM_DD_YY,
     +                IIKLEN2, 8,
     +                IIREC, DAILY_RECORD,
     +                IIRLEN, 45,
     +                IIRMF, 12,
     +                IINLEV, 2,
     +                IER     )
```

File Open Variables

You can only use the file open variables with the IOPEN subroutine. They specify three options on the open (see Table 8-8).

Table 8-8. File Open Variables

Variable	Indicates	Default Value
IINLOX	Maximum Number of Record Locks. You can specify 0 to 32.	0
IIEXCLU	Exclusive use of the database, the index file, or both.	None
IIRDO	Read-only open. Specify either read-only or read-write access of the file.	Read-write

Refer to Appendix B to find the appropriate way to code these variables in your application language.

When the IOPEN subroutine has successfully completed, the interface returns the value of the status variable IIFLEV — the maximum number of index levels allowed in the index. You can then retrieve this value with the IIGET subroutine.

IIRDO Example (FORTRAN 77)

In this example, we open the INFOS II file TEST_FILE, which has three index levels. We specify that we will perform only read operations during this opening of the file.

```
      CALL IOPEN  (   IARRAY,
     +                TEST_FILE, 3,
     +                IIRDO, READONLY,
     +                IER, ENDLIST    )
```

Subindex Definition Variables

Use the subindex definition variables in an ISDEFINE subroutine if you want to specify values other than the defaults for certain parameters. Table 8-9 lists the four variables in this group.

Table 8-9. Subindex Definition Variables

Variable	Indicates	Default Value
IISALLOW	Allow subindexes, duplicate keys, or both.	Neither allowed
IISMKLEN	Maximum key length in subindex.	256
IISPRLEN	Partial record length in subindex.	0
IIRNS	Root node size of subindex.	2042

Be sure to refer to Appendix B to find the correct parameter value for IISALLOW for your language.

Subindex Definition Variables Example (PL/I)

In this example, we define a subindex under the key NUM, and allow it to have both duplicate keys and subindexing. The keys in this subindex must be 4 bytes long, and can have partial records up to 8 bytes long.

```
CALL ISDEF  (   IARRAY,
                IIKEY, "NUM",
                IIKLEN, 3,
                IISALLOW, INF_ALLOW_BOTH,
                IISPRLEN, 8,
                IISMKLEN, 4,
                IER     );
```

IISPRLEN

You can also use IISPRLEN as an optimized record distribution variable, to specify a merit factor for the root node of the subindex you are defining. (Note that you can only do this if you specified optimized record distribution upon creating the INFOS II file.) To do this, you must use IISPRLEN in a special way. The high byte of IISPRLEN will contain the subindex root node merit factor and the low byte will contain the partial record length. To specify both of these values, set IISPRLEN to the value determined by the following equation:

IISPRLEN = (Merit Factor * 256) + Partial Record Length

For example, if you want a merit factor of 10 and a partial record length of 8 bytes, you will set IISPRLEN to 2568, according to the equation:

IISPRLEN = (10 * 256) + 8 = 2568

Link Subindex Variables

Use the link subindex variables with the ISLINK subroutine only. These variables correspond to other interface variables we have already discussed. You use them to specify data areas, lengths, key levels, and modes for the destination key of your subindex link. For the source key, you use the variables that we described in earlier sections. We list the link subindex variables in Table 8-10, and note which other interface variables they correspond to.

Table 8-10. Link Subindex Variables

Variable	Indicates	Corresponds to
IIDKEY*[n]*	Destination key data area, where *n* is the key level.	IIKEY*[n]*
IIDKLEN*[n]*	Destination key length, where *n* is the key level.	IIKLEN*[n]*
IIDKMODE*[n]*	Destination keyed search mode, where *n* is the key level.	IIKMODE*[n]*
IIDNLEV	Destination number of key levels.	IINLEV
IIDPMODE	Destination current positioning mode.	IIPMODE
IIDRMODE	Destination relative positioning mode.	IIRMODE
IIDSMODE	Destination search mode.	IISMODE

These variables function exactly like their corresponding variables listed in the rightmost column of Table 8-10, except that the interface does not remember their values after it finishes the link subindex operation.

Link Subindex Variables Example (DG/L)

In this example, we link the subindexes NAME and NUMBER, setting current position on the key NUMBER after the operation. Both NAME and NUMBER are located under the selector key "CUSTOMER".

```
ISLINK   (      IARRAY,
                IIKEY, "CUSTOMER",
                IIKLEN, 8,
                IIKEY2, "NAME",
                IIKLEN2, 4,
                IINLEV, 2,
                IISMODE, KEYED,
                IIDKEY, "CUSTOMER",
                IIDKLEN, 8,
                IIDKEY2, "NUMBER",
                IIDKLEN2, 6,
                IIDNLEV, 2,
                IIDSMODE, KEYED,
                IIDPMODE, SET_POS,
                IER     );
```

The Interface Subroutines

In the following sections we describe each interface subroutine and explain how to use it with the various languages. The number of arguments you use with a subroutine depends on your purpose for using the subroutine. Note that for many of the interface variables, you will need default values, which means you won't have to include the variables in the subroutine. Of course it won't matter if you do specify a default value in an argument pair. For example, set current position is a default option, but you can specify it with IIPMODE in subroutines for which IIPMODE is valid.

We noted our conventions for notations earlier; we repeat them here for your convenience:

Notation	Meaning
iarray	Interface array name.
ier	Error status variable.
arg-pair	Argument pair; i.e., an interface variable followed by a comma and a value or variable.

For all other notation conventions, see the Preface of this manual.

Also note that Appendix C contains sample programs using the interface in FORTRAN 5, FORTRAN 77, PL/I, and DG/L. Refer to them for additional help in writing your subroutine calls.

Table 8-11 shows the interface variables that you can use for each interface subroutine.

Table 8-11. Interface Variables by Subroutine

Variable \ Subroutine	IOPEN	ICLOSE	IISET	IIGET	IKEYREAD	IRELREAD	IWRITE	IREWRITE	IDELETE	ILDELETE	ISDEFINE	ISLINK	IRELEASE	ISDELETE	IHIREAD	IRETDEF
Data Area Variables																
IIREC			x		x	x	x	x							x	
IIKEY*[n]*			x		x	x	x	x	x	x	x	x		x	x	x
IIPREC			x		x	x	x	x							x	x
Length Variables																
IIRLEN			x	x			x	x								
IIKLEN*[n]*			x	x	x	x	x	x	x	x	x	x		x	x	x
IILRLEN			x		x	x									x	x
IIPRLEN			x	x												
IIMRLEN				x												
IIMKLEN				x												
Mode Variables																
IIAMODE			x		x	x	x	x		x						
IIDMODE			x							x						
IIKMODE*[n]*			x	x	x	x	x	x	x	x	x	x		x	x	x
IILMODE			x		x	x	x	x	x	x			x			
IIPMODE			x	x	x	x	x	x	x	x	x	x		x	x	x
IIRMODE			x	x	x	x	x	x	x	x	x	x		x	x	x
IISMODE					x		x	x	x	x	x	x		x	x	x
Status Variables																
IILEV				x												
IIFLEV				x												
IIOCCH*[n]*			x	x												
IIOCCL*[n]*			x	x												
IIFDBKH			x	x												
IIFDBKL			x	x												
IIDUP				x												
IIDBDEL				x												
IIPRDEL				x												
IIOVFLO				x												
IISIP				x												

(continues)

Table 8-11. Interface Variables by Subroutine

Variable \ Subroutine	IOPEN	ICLOSE	IISET	IIGET	IKEYREAD	IRELREAD	IWRITE	IREWRITE	IDELETE	ILDELETE	ISDEFINE	ISLINK	IRELEASE	ISDELETE	IHIREAD	IRETDEF
Optimized Record Distribution Variables																
IIRMF			x				x	x								
IIVMF				x												
Multilevel Variable																
IINLEV			x	x	x	x	x	x	x	x	x	x	x	x	x	x
File Open Variables																
IINLOX	x															
IIEXCLU	x															
IIRDO	x															
Subindex Definition Variables																
IISALLOW											x					
IISMKLEN											x					
IISPRLEN											x					
IIRNS											x					
Link Subindex Variables																
IIDKEY*[n]*												x				
IIDKLEN*[n]*												x				
IIDKMODE*[n]*												x				
IIDNLEV												x				
IIDPMODE												x				
IIDRMODE												x				
IIDSMODE												x				

(concluded)

IOPEN

Opens an INFOS II file.

Syntax

FORTRAN 5: CALL IOPEN (iarray,filename,mlev *[,arg-pair...]* ,ier)

FORTRAN 77: CALL IOPEN (iarray,filename,mlev *[,arg-pair...]* ,ier, ENDLIST)

DG/L: IOPEN (iarray,filename,mlev *[,arg-pair...]* ,ier);

PL/I: CALL IOPEN (iarray,filename,mlev *[,arg-pair...]* ,ier);

Description

Use this subroutine to do the following:

- Open the existing INFOS II file, filename.
- Initialize the interface array, iarray.
- Position down (relatively), so that your current position is before the first key of the main (level 0) index.

IOPEN must be the first interface subroutine that refers to your interface array. It associates filename with iarray so that in your subsequent interface calls you only need to mention iarray, and not the filename.

The interface interprets mlev as the key depth and array size that you will not exceed. If this key level exceeds either the file depth or the size you determined for your interface array, the system returns the appropriate error message.

Interface Variables

You can use all of the file open interface variables with the IOPEN subroutine. They include:

IINLOX
IIEXCLU
IIRDO

IINLOX allows you to specify the maximum number of outstanding record locks that you want to have at any one time. The default value is 0; the maximum is 32.

IIEXCLU allows you to specify that you want exclusive use of either the database, the index file, or both. See Appendix B for applicable values for these variables.

Use IIRDO if you want to set or reset the read-only option from the language interface. Specify the appropriate argument pair as listed in your source language parameter file. The default is read-write access. If you perform a read-only OPEN on a file, and the AOS or INFOS II system fails, you do not have to use IVERIFY on that file to run it.

Examples

FORTRAN 5

In this example, we open the file INVENTORY, which has two index levels. We use the IIRDO variable to specify that we will perform only read requests.

```
      CALL IOPEN  (   IARRAY,
     +                INVENTORY, 2,
     +                IIRDO, READONLY,
     +                IER     )
```

PL/I

In this example, we open the file INFOS_FILE, which has three index levels, and request exclusive use of both the index and the database.

```
CALL IOPEN  (   IARRAY,
                INFOS_FILE, 3,
                IIEXCLU, INF_EX_ALL,
                IER     );
```

DG/L

In this example, we specify a maximum of four outstanding locks on the CUSTOMERS file. CUSTOMERS has two index levels.

```
IOPEN  (        IARRAY,
                CUSTOMERS, 2,
                IINLOX, 4,
                IER     );
```

ICLOSE

Closes an INFOS II file.

Syntax

FORTRAN 5: `CALL ICLOSE (iarray,ier)`

FORTRAN 77: `CALL ICLOSE (iarray, ier, ENDLIST)`

DG/L: `ICLOSE (iarray,ier);`

PL/I: `CALL ICLOSE (iarray,ier);`

Description

This subroutine closes the INFOS II file associated with iarray. After a successful call to ICLOSE, the only interface subroutine that can successfully refer to iarray is IOPEN.

You should always close your INFOS II file when you are finished processing it. The INFOS II system will close the file when your program terminates (normally or abnormally), even if you don't use the ICLOSE subroutine. But, it is considered poor programming practice to cause this abnormal closing of the file. See the sample programs in Appendix C for examples of using ICLOSE with an error-handler.

Interface Variables

Interface variables are not applicable with the ICLOSE subroutine.

Examples

FORTRAN 77

```
      CALL ICLOSE ( IARRAY, IER, ENDLIST )
```

DG/L

```
      ICLOSE ( IARRAY, IER );
```

IISET

Sets an interface variable.

Syntax

FORTRAN 5: CALL IISET (iarray *[,arg-pair...]* ,ier)

FORTRAN 77: CALL IISET (iarray *[,arg-pair...]* ,ier, ENDLIST)

DG/L: IISET (iarray *[,arg-pair...]* ,ier);

PL/I: CALL IISET (iarray *[,arg-pair...]* ,ier);

Description

Use this subroutine to change the current value of any of the interface variables listed below. Remember that changing the value of these variables with any subroutine other than IISET changes the value only for the duration of that subroutine. You must change the values with IISET if you want them to remain any longer than that. (The only exception to this is that the Read and Retrieve Key subroutines set the lengths of IIRLEN and IIKLEN to the lengths of the accessed record and key.) Once you change the value of a variable with IISET, its value remains as you have set it until you again issue IISET, or close the file.

Note that you cannot use this subroutine until you have initialized iarray in an IOPEN call.

Interface Variables

You can use the following interface variables with IISET. The *n* on certain variables indicates the key level.

Data Area Variables:

IIREC
IIKEY*[n]*
IIPREC

Length Variables:

IIRLEN*[n]*
IIKLEN*[n]*
IILRLEN
IIPRLEN

Mode Variables:

IIAMODE
IIDMODE
IIKMODE*[n]*
IILMODE
IIPMODE
IIRMODE

Status Variables:

IIOCCH*[n]*
IIOCCL*[n]*

Optimized Record Distribution Variable:

IIRMF

Multilevel Variable:

IINLEV

IISET (continued)

We recommend that you do not use string literals in the IISET subroutine in procedures or subroutines below the main program, for any of the scientific languages. Some compilers store these constants in read-only memory, and if INFOS II returns data to these areas it could cause a validity trap. We suggest that you use global variables in IISET or call this subroutine only from the main program.

Example (FORTRAN 77)

In this example, we set the current position on the key whose name is stored in the data area CO_NAME. This key is located in the subindex under the key NAM. Its record is located in RECORD and is 55 bytes long. The partial record is stored in PHONE and is 8 bytes long. We have set our deletion mode to reinstate. This indicates that if we subsequently call an ILDELETE subroutine, the key that we specify will be reinstated, not deleted.

```
      CALL IISET  (   IARRAY,
     +                IIKEY, "NAM",
     +                IIKLEN, 3,
     +                IIKEY2, CO_NAME,
     +                IKLEN2, 20,
     +                IIREC, RECORD,
     +                IIRLEN, 55,
     +                IIPREC, PHONE,
     +                IIPRLEN, 8,
     +                IIDMODE, REINST,
     +                IIPMODE, SET POS,
     +                IER, ENDLIST    )
```

IIGET

Retrieves interface variable(s).

Syntax

FORTRAN 5: CALL IIGET (iarray *[,arg-pair...]* ,ier)

FORTRAN 77: CALL IIGET (iarray *[,arg-pair...]* ,ier, ENDLIST)

DG/L: IIGET (iarray *[,arg-pair...]* ,ier);

PL/I: CALL IIGET (iarray, *[,arg-pair...]* ,ier);

Description

Use IIGET to retrieve the current value of an interface variable. The list below indicates which variables you can use with the IIGET subroutine. Use IIGET primarily to retrieve information that the INFOS II system returns after a database access. For instance, you might use the IIGET subroutine to find out the data record feedback of a key in order to perform inversion. Or you might need to find out a key's occurrence number in an index that has duplicate keys, so you can perform an operation on the correct key. Note that you cannot issue IIGET after you have used ICLOSE on that interface array.

Interface Variables

You can use the following interface variables in an IIGET subroutine, where *n* indicates the key level.

Length Variables:

IIRLEN*[n]*
IIKLEN*[n]*
IIPRLEN
IIMRLEN
IIMKLEN

Mode Variables:

IIKMODE*[n]*
IIPMODE
IIRMODE

Status Variables:

IILEV
IIFLEV
IIOCCH*[n]*
IIOCCL*[n]*
IIFDBKH
IIFDBKL
IIDUP
IIDBDEL
IIPRDEL
IIOVFLO
IISIP

Optimized Record Distribution Variable:

IIVMF

Multilevel Variable:

IINLEV

IIGET (continued)

Example (FORTRAN 5)

In the following example, we use the IIGET subroutine to retrieve information on the key we have accessed for a prior subroutine call. We use the appropriate variables to retrieve the length of the key, the record, and the partial record. We also ask for data record feedback information, the key's index level, and whether or not the key is a duplicate. Finally, we set the current position on the key.

```
      CALL IIGET  (   IARRAY,
     +                IIKLEN, KEY_LENGTH,
     +                IIRLEN, REC_LENGTH,
     +                IIPRLEN, P_REC_LENG,
     +                IIPMODE, SET POS,
     +                IIFDBKH, DB_FDBK_HIGH,
     +                IIFDBKL, DB_FDBK_LOW,
     +                IIDUP, DUP_KEY_STATUS,
     +                IILEV, INDEX_LEVEL,
     +                IER     )
```

IKEYREAD

Uses keyed access to read and retrieve a record.

Syntax

FORTRAN 5: CALL IKEYREAD (iarray *[,arg-pair...]* ,ier)

FORTRAN 77: CALL IKEYREAD (iarray *[,arg-pair...]* ,ier ,ENDLIST)

DG/L: IKEYREAD (iarray *[,arg-pair...]* ,ier);

PL/I: CALL IKEYR (iarray *[,arg-pair...]* ,ier);

Description

The IKEYREAD subroutine performs first a Read and then a Retrieve Key operation. If you do not specify any argument pairs, IKEYREAD performs an exact key search through the current number of key levels specified in IINLEV (the default is one level). If no error occurs, IKEYREAD does the following:

- Reads the data record into the database area specified in IIREC.
- Reads the partial record into your partial record area (IIPREC).
- Sets IIRLEN and IIKLEN to the lengths of the accessed record and key.
- Sets any applicable status variables.
- Sets the current position on this key.

If the system does not find the key you specified, it sets ier to IOKPE (Keyed Positioning Error). If there is any other error, then it sets ier to the appropriate error code. As long as you have set up an error-handler, you will be able to trace the error.

Interface Variables

You can use IIKEYREAD with any of the interface variables we list below, where *n* indicates the key level.

Data Area Variables:

IIREC
IIKEY*[n]*
IIPREC

Length Variables:

IIKLEN*[n]*
IILRLEN

Mode Variables:

IIAMODE
IIKMODE*[n]*
IILMODE
IIPMODE
IIRMODE
IISMODE

Multilevel Variable:

IINLEV

If you have not defined IIREC or IIPREC prior to calling this subroutine, then IKEYREAD automatically suppresses the database or the partial record, whichever is appropriate.

IKEYREAD (continued)

To specify an alternate source for the key or an alternate destination for the data record or partial record, you can set the appropriate Data Area interface variable — IIKEY, IIREC, or IIPREC.

You can also set IIKLEN to indicate an alternate key length or IILRLEN to limit the length of the data record that the system retrieves. Note, however, that if you do the latter, you must also set IIAMODE to your language's appropriate parameter to indicate that you are limiting the record length. If IIKMODEn at your lowest key level calls for an approximate or generic search, then IKEYREAD will read the lowest level key it finds into IIKEYn and set IIKLENn (where n, in each case, is the current number of levels specified in IINLEV). However, you can specify an approximate or generic search at any number of levels.

To specify a combined keyed and relative search, you must set IISMODE to the appropriate parameter for relative and keyed, and IIRMODE to the direction of relative motion you want.

Use IINLEV to specify a number of key levels to search, if you don't want the default.

If you want to perform an INFOS II Retrieve Key without a Read, you must set IIAMODE to the KEY ONLY parameter. Refer to your source language parameter file to find the appropriate parameter.

Example (DG/L)

In the following example, we perform a Read using keyed and relative access. We first read down and forward from our current position. Then we use keyed access to locate the key in the data area NAME. We use the interface variables IILRLEN and IIAMODE to limit the length of the record we read to 40 bytes. Finally, we set the current position on the accessed key.

```
IKEYREAD  (     IARRAY,
                IISMODE, RELKEY,
                IIRMODE, DOWN_FOR,
                IINLEV, 2,
                IIKEY2, NAME,
                IIKLEN2, 15,
                IIREC, NAME_RECORD,
                IIAMODE, LIM_RLEN,
                IILRLEN, 40,
                IIPMODE, SET_POS,
                IER     );
```

IRELREAD

Uses relative access to read and retrieve a record.

Syntax

FORTRAN 5: CALL IRELREAD (iarray */,arg-pair.../* ,ier)

FORTRAN 77: CALL IRELREAD (iarray */,arg-pair.../* ,ier ,ENDLIST)

DG/L: IRELREAD (iarray */,arg-pair.../* ,ier);

PL/I: CALL IRELR (iarray */,arg-pair.../* ,ier);

Description

The IRELREAD subroutine uses relative access to first read and then retrieve a key. If you supply no argument pairs, IRELREAD will use the current position and perform relative access according to the current value of IIAMODE. If an error occurs, it will set ier to the appropriate error code. If no error occurs, IRELREAD will do the following:

- Read the data record into the area specified in IIREC.
- Read the partial record into your partial record area (IIPREC).
- Read the key into your key data area (IIKEY).
- Set IIRLEN and IIKLEN to the lengths of the accessed record and key.
- Set any applicable status variables.
- Set the current position on the accessed key.

If you have not previously defined IIREC, IIPREC, or IIKEY, then this subroutine will also suppress the database or partial record automatically. This automatic suppression can prevent you from receiving an otherwise confusing error message.

Interface Variables

You can use IRELREAD with any of the variables we list below, where *n* indicates the key level.

Data Area Variables:

IIREC
IIKEY*/n/*
IIPREC

Length Variables:

IIKLEN*/n/*
IILRLEN

Mode Variables:

IIAMODE
IIKMODE*/n/*
IILMODE
IIPMODE
IIRMODE

Multilevel Variable:

IINLEV

IRELREAD (continued)

To specify an alternate destination for the returned key, data record, or partial record, set the appropriate Data Area interface variable(s) — IIKEY, IIREC, and IIPREC.

To limit the length of the retrieved data record, you can set IILRLEN. If you do, be sure to also set IIAMODE to the appropriate parameter for your language to indicate that you are limiting record length.

To specify that you want the system to return a key other than the top level key as a result of relative positioning, set IINLEV to the appropriate number of index levels.

If you want to retrieve the key without reading the record, set IIAMODE to the appropriate parameter to indicate KEY ONLY in your application language.

Example (PL/I)

In the following example, we use relative access to read a key. We read forward, setting the current position.

```
CALL IRELR  (   IARRAY,
                IIKEY, KEY_NAME,
                IIKLEN, 15,
                IIREC, RECORD,
                IIRMODE, INF_FORWARD,
                IIPMODE, INF_SET_POS,
                IER     );
```

IHIREAD

Retrieves the highest key in an index and reads its record.

Syntax

FORTRAN 5: CALL IHIREAD (iarray *[,arg-pair...]* ,ier)

FORTRAN 77: CALL IHIREAD (iarray *[,arg-pair...]* ,ier ,ENDLIST)

DG/L: IHIREAD (iarray *[,arg-pair...]* ,ier);

PL/I: CALL IHIREAD (iarray *[,arg-pair...]* ,ier);

Description

Use the IHIREAD subroutine to gain access to the highest key in a subindex. If you specify any relative motion, IHIREAD will first do this positioning, and then retrieve the highest key in the subindex in which you are positioned. If you use keyed access, IHIREAD will locate the key you specify and then retrieve the highest key in that subindex. Unlike the other subroutines that perform Read operations, IHIREAD *first* does the Retrieve Key, and then the Read. The IKEYREAD and IRELREAD subroutines perform the Read, and then the Retrieve Key.

Interface Variables

You can use the following interface variables with IHIREAD. The *n* on certain variables indicates the key level.

Data Area Variables:

IIREC
IIKEY*[n]*
IIPREC

Length Variables:

IIKLEN*[n]*
IILRLEN

Mode Variables:

IIAMODE
IIKMODE*[n]*
IILMODE
IIPMODE
IIRMODE
IISMODE

Multilevel Variable:

IINLEV

IHIREAD returns the key in IIKEY, its length in IIKLEN, its record in IIREC, and its record length in IIRLEN. If you then want to read these values, you must perform an IIGET operation.

IHIREAD, like the other Read subroutines, performs a Retrieve Key as well as a Read. If you want to perform just the Retrieve Key, you must set IIAMODE to KEY ONLY.

It is important to note that this subroutine, like IISET, sets the values of the variables IIRLEN and IIKLENn to the lengths of the accessed record and key. All of the subroutines that perform read and retrieve key operations set these values.

IHIREAD (continued)

Example (FORTRAN 5)

In the following example, we use IHIREAD to position down and forward to a level 2 subindex, and retrieve the highest key and its record. We limit the length of the record that will be returned to 20 bytes.

```
      CALL IHIREAD  ( IARRAY,
     +                IIKEY2, LEVEL_2_KEY,
     +                IIKLEN2, KEY_LENGTH,
     +                IIREC, RECORD,
     +                IILRLEN, 20,
     +                IIAMODE, LIM RLEN,
     +                IINLEV, 2,
     +                IISMODE, RELATIVE,
     +                IIRMODE, DOWN FOR,
     +                IER     )
```

IWRITE

Writes a record and its key.

Syntax

FORTRAN 5: CALL IWRITE (iarray *[,arg-pair...]* ,ier)

FORTRAN 77: CALL IWRITE (iarray *[,arg-pair...]* ,ier, ENDLIST)

DG/L: IWRITE (iarray *[,arg-pair...]* ,ier);

PL/I: CALL IWRITE (iarray *[,arg-pair...]* ,ier);

Description

Use the IWRITE subroutine to write keys, records, and partial records. If you don't use any argument pairs with this subroutine, it will write the key previously set by IISET (length specified in IIKLENn) into the index at level n, where n is the number you specified in IINLEV.

If a key by that name already exists in the index, IWRITE will check to see if you allowed duplicate keys when you created the index and if you set IIKMODE at the correct key level to indicate the possibility of a duplicate key. If you did, it will write the duplicate key. Otherwise, IWRITE will set ier to IOKAE (Key Already Exists).

If there are no errors and you have not used IIAMODE to suppress the data record or partial record, this subroutine will do the following:

- Write the key in the index.
- Write a new database record.
- Write a partial record.

Interface Variables

You can use the following interface variables with the IWRITE subroutine, where *n* indicates the key level.

Data Area Variables:

IIREC
IIKEY*[n]*
IIPREC

Length Variables:

IIRLEN
IIKLEN*[n]*

Mode Variables:

IIAMODE
IIKMODE*[n]*
IILMODE
IIPMODE
IIRMODE
IISMODE

Optimized Record Distribution Variable:

IIRMF

Multilevel Variable:

IINLEV

IWRITE (continued)

Note that you cannot specify partial record length (IIPRLEN); the partial record length is automatically the length you specified when you defined the subindex. In addition, you cannot limit the record length (IILRLEN). There is no need to do so because if the record you write is smaller than the maximum record length, IWRITE will pad it with blanks.

You can specify an alternate source for the key, data record, or partial record by setting the appropriate Data Area interface variables and IINLEV (if the key is multilevel).

If you are gaining access to the file for the first time, or if you want to specify record and key lengths other than the current defaults, set IIRLEN and IIKLEN.

Examples

FORTRAN 77

The following FORTRAN 77 IWRITE subroutine uses the default values to write a key, record, and partial record.

```
      CALL IWRITE  (  IARRAY,
     +                IER, ENDLIST    )
```

The key and records written with this subroutine are those currently located in the data areas IIKEY, IIREC, and IIPREC. Their lengths are currently located in the length variables IIKLEN, IIRLEN, and IIPRLEN.

PL/I

The following PL/I subroutine writes the key located in the data area NAME into the level 2 subindex located under the key CO_NAME. Note that we specify the exact level 1 key name, CO_NAME, by enclosing it in quotes. We write the record located in the data area CO_RECORD into the database, and we specify suppress partial record because we don't want to write a partial record. In addition, we specify that the key could be a duplicate.

```
CALL IWRIT  (   IARRAY,
                IIKEY, "CO_NAME",
                IIKLEN, 7,
                IIKEY2, NAME,
                IIKLEN2, 12,
                IINLEV, 2,
                IIKMODE2, DUP KEY,
                IIREC, CO_RECORD,
                IIRLEN, 125,
                IIAMODE, INF_SUPP_PR,
                IER     );
```

DG/L

The following DG/L subroutines perform an inverted write. We assume that we have already gained access to the existing key that points to the data record we want to use, and stored its data record feedback in HIGH_FDBK and LOW_FDBK. Now we must use IISET to set the data record feedback. We can then invert the correct data record to the new key. The key to which we are performing the inverted write is stored in the data area PART_NAMES.

```
IISET ( IARRAY,
        IIFDBKH, HIGH_FDBK,
        IIFDBKL, LOW_FDBK,
        IER      );

IWRITE (IARRAY,
        IIAMODE, INVERT,
        IIKEY, "PARTS",
        IIKLEN, 5,
        IIKEY2, PART_NAMES,
        IIKLEN2, 15,
        IINLEV, 2,
        IER      );
```

IREWRITE

Rewrites a data record.

Syntax

FORTRAN 5: CALL IREWRITE (iarray *[,arg-pair...]* ,ier)

FORTRAN 77: CALL IREWRITE (iarray *[,arg-pair...]* ,ier, ENDLIST)

DG/L: IREWRITE (iarray *[,arg-pair...]* ,ier);

PL/I: CALL IREWRITE (iarray *[,arg-pair...]* ,ier);

Description

The IREWRITE subroutine allows you to rewrite an existing record or partial record, or write a new data record or partial record if the key you specified does not currently have one.

If you specify only a key (IIKEY), the IREWRITE subroutine will perform an exact key search through the number of levels contained in IINLEV. If it does not find that key, then it will set ier to IOKPE (Keyed Positioning Error). Unless there are errors, IREWRITE will update the data record and the partial record, as long as you have not suppressed either or both with IIAMODE.

Interface Variables

You can use the following interface variables with the IREWRITE subroutine. The *n* on certain variables indicates the key level.

Data Area Variables:

IIREC
IIKEY*[n]*
IIPREC

Length Variables:

IIRLEN
IIKLEN*[n]*

Mode Variables:

IIAMODE
IIKMODE*[n]*
IILMODE
IIPMODE
IIRMODE
IISMODE

Optimized Record Distribution Variable:

IIRMF

Multilevel Variable:

IINLEV

To specify an alternate source for the key, data record, or partial record, set the appropriate Data Area interface variable(s).

If this is the first time you have gained access to the file, or if you just want to specify key and record lengths that are different from the current defaults, set IIRLEN and IIKLEN accordingly.

To specify a different number of search levels other than the default, set IINLEV to the number of levels you want.

If you are using Optimized Record Distribution, you can specify a merit factor for the record you're rewriting by setting IIRMF as described earlier in this chapter.

Examples

FORTRAN 77

The following FORTRAN 77 subroutine rewrites the partial record of the key located in the data area LEV_1_KEY. It does not rewrite the data record because we have suppressed it.

```
      CALL IREWRITE ( IARRAY,
     +                IIKEY, LEV_1_KEY,
     +                IIKLEN, 10,
     +                IIPREC, NEW_PART_REC,
     +                IIAMODE, SUPP DB,
     +                IER, ENDLIST    )
```

PL/I

The following PL/I subroutine uses static relative positioning to rewrite the data record and the partial record of the key on which we are currently positioned.

```
CALL IREWR  (   IARRAY,
                IISMODE, INF_RELATIVE,
                IIRMODE, INF_STATIC,
                IIREC, NEW_RECORD,
                IIRLEN, 255,
                IIPREC, NEW_NUMBER,
                IER     );
```

DG/L

The following DG/L subroutine uses the default values contained in IIKEY, IIREC, IIKLEN, and IIRLEN to rewrite a data record.

```
IREWRITE  ( IARRAY, IER );
```

IDELETE

Physically removes a record and its key.

Syntax

FORTRAN 5: CALL IDELETE (iarray *[,arg-pair...]* ,ier)

FORTRAN 77: CALL IDELETE (iarray *[,arg-pair...]* ,ier ,ENDLIST)

DG/L: IDELETE (iarray *[,arg-pair...]* ,ier);

PL/I: CALL IDELE (iarray *[,arg-pair...]* ,ier);

Description

Use the IDELETE subroutine to delete a key and its data record and partial record. You must use keyed or keyed and relative access with the IDELETE subroutine. IDELETE searches through the number of levels you specify in IINLEV for the key you indicate in IIKEYn, where n is the key level. If it cannot find the key, it will set ier to IOKPE (Keyed Positioning Error).

If there are no errors, this subroutine will physically remove the key, its partial record, and its associated data record. However, if the key points to a data record that is pointed to by another key, then IDELETE will delete the key and not the record. Likewise, if the key you specify is linked to more than one other key or subindex, IDELETE will not delete it, but will break the link according to the key path you specified. (See Chapter 3 of this manual for more details on physical deletion.)

Interface Variables

You can use the following interface variables with the IDELETE subroutine:

Data Area Variable:

IIKEY*[n]*

Length Variable:

IIKLEN*[n]*

Mode Variables:

IIKMODE*[n]*
IIPMODE
IIRMODE
IISMODE

Multilevel Variable:

IINLEV

Note that you cannot use the mode variable IIDMODE with IDELETE. If you want to indicate a logical delete rather than a physical one you must use the ILDELETE subroutine.

If you set the current position (IIPMODE) on the key you delete, or if your current position was already set on the deleted key, then your resulting current position will be on the key that preceeds the one that was deleted. If the key you deleted was the first in a subindex, and you set your position on it, then your resulting position will be in front of the key that is now the first in that subindex.

CAUTION: If you use IDELETE to delete a single index file of a multi-index database, you will corrupt the INFOS II database. In addition, you must be sure to use the IDELETE utility *only* to delete INFOS II files.

Example (FORTRAN 5)

In this example, we delete a key using relative and keyed positioning. We want to delete a level 2 key, so we specify IINLEV as 2 and include the 2 on IIKMODE, which we also use to indicate that the key could be a duplicate.

```
      CALL IDELETE  ( IARRAY,
     +                IISMODE, RELKEY,
     +                IIRMODE, DOWN,
     +                IIKEY2, SCHOOL_NAME,
     +                IIKLEN2, 30,
     +                IIKMODE2, DUP KEY,
     +                IINLEV, 2,
     +                IER     )
```

ILDELETE

Logically deletes a record and its key.

Syntax

FORTRAN 5: CALL ILDELETE (iarray *[,arg-pair...]* ,ier)

FORTRAN 77: CALL ILDELETE (iarray *[,arg-pair...]* ,ier ,ENDLIST)

DG/L: ILDELETE (iarray *[,arg-pair...]* ,ier);

PL/I: CALL ILDEL (iarray *[,arg-pair...]* ,ier);

Description

Use ILDELETE to logically delete a data record or partial record or both, or to reinstate a logically deleted key and its records. You must use keyed or keyed and relative motion. ILDELETE searches through the number of levels you specify in IINLEV for IIKEYn, where n is the key level. If it does not find the key, it will return a Keyed Positioning Error (IOKPE) to ier.

If there are no errors, ILDELETE will logically delete the key and data record. A logical deletion does not remove any of the file contents, nor does it prevent access to the key or data record. It simply flags the entry, which notifies all users that it has been logically deleted. You can also reinstate the key and record with this subroutine if you use the correct parameter with IIDMODE.

See Chapter 3 for more details on logical deletion.

Interface Variables

You can use the following interface variables with the ILDELETE subroutine. The n on certain variables indicates the key level.

Data Area Variable:

IIKEY*[n]*

Length Variable:

IIKLEN*[n]*

Mode Variables:

IIDMODE
IIKMODE*[n]*
IIPMODE
IIRMODE
IISMODE

Multilevel Variable:

IINLEV

You can specify an alternate source for the key by setting the IIKEY Data Area variable, along with IINLEV if the key is multilevel.

Use the IIDMODE variable to choose between logical deletion and reinstatement, and to specify whether you want the operation to include the data record, the partial record, or both.

Example (PL/I)

In this example, we reinstate the key whose name is stored in the data area NAME_KEY. In order for this subroutine to work, we must have previously used the ILDELETE subroutine to logically delete this key.

```
CALL ILDEL  (   IARRAY,
                IIKEY, NAME_KEY,
                IIKLEN, 20,
                IIDMODE, INF_REINST,
                IER     );
```

ISDEFINE

Defines a subindex using the specified parameters.

Syntax

FORTRAN 5: CALL ISDEFINE (iarray *[,arg-pair...]* ,ier)

FORTRAN 77: CALL ISDEFINE (iarray, *[,arg-pair...]* ,ier ,ENDLIST)

DG/L: ISDEFINE (iarray *[,arg-pair...]* ,ier);

PL/I: CALL ISDEF (iarray *[,arg-pair...]* ,ier);

Description

Use the ISDEFINE subroutine to specify parameters for a subindex that you want to define under a key. You must use either keyed or keyed and relative positioning with ISDEFINE. The subroutine searches the number of levels you specify in IINLEV for the key you indicate in IIKEYn, where n is the key level. If it does not find the key, it will set ier to IOKPE (Keyed Positioning Error). If the key already has a subindex, it will set ier to IOSAE (Subindex Already Exists). For other errors it sets ier accordingly. If there are no errors, ISDEFINE will define a subindex under the key you specified. It will also set the current position on the accessed key.

If you don't use the subindex definition variables to specify alternative parameters, ISDEFINE will define the subindex according to the defaults.

Interface Variables

You can use the following interface variables with ISDEFINE, where *n* is the key level.

Data Area Variables:

IIKEY*[n]*

Length Variable:

IIKLEN*[n]*

Mode Variables:

IIKMODE*[n]*
IIPMODE
IIRMODE
IISMODE

Multilevel Variable:

IINLEV

Subindex Definition Variables:

IISALLOW
IISMKLEN
IISPRLEN
IIRNS

Here are the Subindex Definition Variables in more detail:

Variable	Indicates	Default Value
IISALLOW	If subindexes and/or duplicate keys are allowed in the subindex.	Neither allowed.
IISMKLEN	Maximum key length in the subindex.	255
IISPRLEN	Partial record length in the subindex.	0
IIRNS	Root node size of the subindex.	2042

You can find a full explanation of the defaults and options for subindex definition in Chapter 3 of this manual. Refer to your source language parameter file to select a value for IISALLOW, unless you want the default.

If you are using Optimized Record Distribution, you can specify a merit factor for the root node of the subindex you are defining. To so so, you must use IISPRLEN in a special way. That is, the high byte of IISPRLEN will contain the root node merit factor of the subindex, and the low byte will contain the partial record length. Therefore, to specify both of these, you should set IISPRLEN to a value determined as follows:

IISPRLEN = (Merit Factor * 256) + Partial Record Length

Examples

FORTRAN 77

In the following FORTRAN 77 ISDEFINE subroutine, we define a subindex under the key "NAM". The keys in this subindex will be a maximum of 24 bytes long, and they can be duplicates. They can also have partial records up to 20 bytes long. We will accept the default root node size.

```
      CALL ISDEFINE ( IARRAY,
     +                IIKEY, "NAM",
     +                IIKLEN, 3,
     +                IISALLOW, ALLOW DUP,
     +                IISMKLEN, 24,
     +                IISPRLEN, 20,
     +                IER, ENDLIST    )
```

DG/L

The following DG/L subroutine shows how to define a subindex with the default values. We define the subindex under the level 1 key CO_NUMBERS.

```
ISDEFINE  (     IARRAY,
                IIKEY, "CO_NUMBERS",
                IIKLEN, 10,
                IER     );
```

ISDELETE

Deletes a subindex.

Syntax

FORTRAN 5: CALL ISDELETE (iarray *[,arg-pair...]* ,ier)

FORTRAN 77: CALL ISDELETE (iarray *[,arg-pair...]* ,ier ,ENDLIST)

DG/L: ISDELETE (iarray *[,arg-pair...]* ,ier);

PL/I: CALL ISDEL (iarray *[,arg-pair...]* ,ier);

Description

The ISDELETE subroutine locates the key you indicate and deletes the subindex beneath it. As with the other Delete operations, you cannot use relative positioning alone with ISDELETE; you must use keyed or keyed and relative access. ISDELETE searches the number of index levels you indicate in IINLEV. If it cannot find the key you specify, it will set ier to IOKPE; if the key has no subindex, ISDELETE will set ier to IOSNP (Subindex Not Present).

If there are no errors, then ISDELETE will act in one of two ways:

- If the key's subindex is not linked to any other subindexes, ISDELETE will physically delete the key's subindex.
- If the subindex contains links to other subindexes, ISDELETE will unlink the subindex from the key you specified.

Interface Variables

You can use the following interface variables with the ISDELETE subroutine, where *n* is the key level.

Data Area Variable:

IIKEY*[n]*

Length Variable:

IIKLEN*[n]*

Mode Variables:

IIKMODE*[n]*
IIPMODE
IIRMODE
IISMODE

Multilevel Variable:

IINLEV

Example (PL/I)

In the following example, we use keyed access with the ISDELETE subroutine to delete a subindex under the level 2 key TEMP_NOS. We use IIPMODE to indicate that we want the current position to remain where it was prior to this operation.

```
CALL ISDEL  (   IARRAY,
                IIKEY, "EMPLOYEES",
                IIKLEN, 9,
                IIKEY2, TEMP_NOS,
                IIKLEN2, 6,
                IIPMODE, INF_NO_CHANGE,
                IER     );
```

ISLINK

Links two subindexes.

Syntax

FORTRAN 5: CALL ISLINK (iarray *[,arg-pair...]* ,ier)

FORTRAN 77: CALL ISLINK (iarray *[,arg-pair...]* ,ier, ENDLIST)

DG/L: ISLINK (iarray *[,arg-pair...]* ,ier);

PL/I: CALL ISLIN (iarray *[,arg-pair...]* ,ier);

Description

Use the ISLINK subroutine to link the subindex associated with the source key to the destination key. The link subindex operation requires two key paths. Use the Data Area, Length, Multilevel, and Mode variables that we list below to specify the source key. Use the Link Subindex variables to specify the destination key.

If you don't supply any argument pairs to specify a path for the source key, the interface will determine the source key by performing an exact key search using the current values of IIKEY and IINLEV.

If there are no errors, this subroutine will link the destination key to the subindex of the source key. If it cannot find either the source or destination key, it will set ier to IOKPE (Keyed Positioning Error). If the source key does not have a subindex, then ISLINK will set ier to IOSNP (Subindex Not Present).

Interface Variables

You can use the following interface variables with the ISLINK subroutine:

Data Area Variable:

IIKEY*[n]*

Length Variable:

IIKLEN*[n]*

Mode Variables:

IIKMODE*[n]*
IIPMODE
IIRMODE
IISMODE

Multilevel Variable:

IINLEV

Link Subindex Variables:

IIDKEY*[n]*
IIDKLEN*[n]*
IIDKMODE*[n]*
IIDNLEV
IIDPMODE
IIDRMODE
IIDSMODE

ISLINK (continued)

Here are the Link Subindex variables in more detail:

Variable	Indicates
IIDKEY*/n/*	Destination key data area, where *n* indicates the key level.
IIDKLEN*/n/*	Destination key length, where *n* indicates the key level.
IIDKMODE*/n/*	Destination keyed search mode, where *n* indicates the key level.
IIDNLEV	Destination number of key levels.
IIDPMODE	Destination current positioning mode.
IIDRMODE	Destination relative processing mode.
IIDSMODE	Destination search mode.

These variables function exactly like IIKEY, IIKLEN, IIKMODE, IINLEV, IIPMODE, IIRMODE, and IISMODE, except that the interface does not retain their values after it finishes the Link Subindex operation.

Example (DG/L)

In this example, we link the subindex associated with the key located in the data area SOURCE_KEY with the subindex of the key located in DEST_KEY. These are both level 1 subindexes.

```
ISLINK ( IARRAY,
         IIKEY, SOURCE_KEY,
         IIKLEN, 24,
         IIDKEY, DEST_KEY,
         IIDKLEN, 12,
         IER    );
```

IRELEASE

Releases locks and/or position.

Syntax

FORTRAN 5: CALL IRELEASE (iarray *[,IILMODE,REL LCK] [,IIPMODE,REL POS]* ,ier)

FORTRAN 77: CALL IRELEASE (iarray *[,IILMODE,REL LCK] [,IIPMODE,REL POS]* ,ier ,ENDLIST)

DG/L: IRELEASE (iarray *[,IILMODE,REL_LCK][,IIPMODE,REL_POS]* ,ier);

PL/I: CALL IRELE (iarray *[,II LMODE,INF_REL_LCK] [,IIPMODE,INF_REL_POS]* ,ier);

Description

Use the IRELEASE subroutine for either or both of the following operations:

- To release all locks on data records and partial records.
- To release the current position and reset it to a point above the main index.

Interface Variables

You can use the following interface variables with the IRELEASE subroutine:

Mode Variables:

IILMODE
IIPMODE

Multilevel Variable:

IINLEV

If you want to release locks, include II LMODE and its appropriate parameter, as listed in your source language parameter file and noted in the syntax for each language above. If you want to release position, use IIPMODE with the correct parameter. Don't forget to specify a number of index levels in IINLEV if you want locks or position released on a key level other than the first.

Example (FORTRAN 5)

In the following example, we indicate that we want to release all locks on the data records and partial records in two key levels of the INFOS II file that we are currently processing. We do not release the current position.

```
      CALL IRELEASE ( IARRAY,
     +                IINLEV, 2,
     +                IILMODE, REL LCK,
     +                IER     )
```

IRETDEF

Retrieves subindex definition information.

Syntax

FORTRAN 5: CALL IRETDEF (iarray, packet *[,arg-pair...]* ,ier)

FORTRAN 77: CALL IRETDEF (iarray, packet *[,arg-pair...]* ,ier ,ENDLIST)

DG/L: IRETDEF (iarray, packet *[,arg-pair...]* ,ier);

PL/I: CALL IRETD (iarray, packet *[,arg-pair...]* ,ier);

Description

Use the IRETDEF subroutine to retrieve information about a subindex and its parameters. Unlike the other subroutines, IRETDEF returns a raw INFOS II Subindex Definition Packet (SDP). (For a description of this packet, see Chapter 9.

The packet we include above in the syntax descriptions must be a data area that you declare early in your program. This data area must be eight words in length so that it is the appropriate size to receive an INFOS II SDP.

You can read the Subindex Definition Packet once IRETDEF has returned it to the packet data area.

Interface Variables

You can use the same interface variables with IRETDEF as you do with the IKEYREAD subroutine. We list them below, where *n* indicates the key level.

Data Area Variables:

IIREC
IIKEY*[n]*
IIPREC

Length Variables:

IIKLEN*[n]*
IILRLEN

Mode Variables:

IIAMODE
IIKMODE*[n]*
IILMODE
IIPMODE
IIRMODE
IISMODE

Multilevel Variable:

IINLEV

Example (PL/I)

In the following example, we show our declaration of the data area THE_PKT and our IRETDEF subroutine. We want to receive the following information about the "NAMES" subindex: the maximum key length, the partial record length, the root node size, if duplicate keys are allowed, and if further subindexing is allowed.

```
%REPLACE $SDP_LEN BY 8;
DECLARE THE_PKT($SDP_LEN)  FIXED BINARY(15);
.
.
.
CALL IRETD  (   IARRAY, THE_PKT,
                IIKEY, "NAMES",
                IIKLEN, 5,
                IISMKLEN, MAX_KEY_LEN,
                IISPRLEN, PAR_REC_LEN,
                IIRNS, RT_NODE,
                IISALLOW, WHAT_ALLOW,
                IER     );
```

Specific Language Considerations

In order to use the AOS INFOS II scientific language interface, you must have Read (R) access to two files, no matter which language you are coding in. The files are:

CLREINF.LB The common language runtime environment library for INFOS II. This is a library of all the INFOS II subroutines for the scientific languages.

IIPAR.ext The source language parameter file for your particular language. The .ext extension is one of the following:

Extension	Language
.FR	FORTRAN 5
.F77	FORTRAN 77
.DG	DG/L
.PL1	PL/I

In addition, remember that if you haven't merged ICALL.OB into the system library, you must include it on your LINK command line. We include it on the LINK command lines in the sections below.

Finally, be sure you create your INFOS II database *before* you run your program, otherwise you will receive the following error message:

```
7212 (IODDE) DATABASE FILE DOES NOT EXIST
```

The following sections contain more detailed information about using each language with the interface. If you need more information on compiling, linking/binding, or running a program on AOS, see the appropriate reference manual for your language.

FORTRAN 5 Considerations

To use the INFOS II interface with a FORTRAN 5 program, you must have Read access to three files: IIPAR.FR (the source language parameter file), CLREINF.LB (the common language runtime environment library), and F5INF.LB (the library of interface subroutines for FORTRAN 5). Normally, you will find these files in the directory :UTIL.

Therefore, to define the interface variables, you must include the following line somewhere near the beginning of your source program:

```
INCLUDE "IIPAR.FR"
```

You need not include an IMPLICIT INTEGER (A-Z) statement in your program; the FORTRAN 5 parameter file explicitly declares all parameters as integers.

After you compile your program, link it with the following CLI command:

```
F5LD programname CLREINF.LB ICALL
```

This includes the interface subroutines in your program. Note that if you build a CLI macro to link your program, make sure that CLREINF.LB precedes F5ISA.LB in your LINK command line. For more details on AOS-related FORTRAN 5 considerations (compiling, linking, etc.), see the *FORTRAN 5 Programmer's Guide (AOS)*.

FORTRAN 77 Considerations

To use the INFOS II interface with your FORTRAN 77 program, you must have Read access to the source language parameter file for FORTRAN 77 and the library containing the interface subroutines. These files, IIPAR.F77 and CLREINF.LB, are normally located in the directory :UTIL.

You must include the following line at the beginning of your program to define the interface variables:

```
INCLUDE"IIPAR.F77"
```

You need not include an IMPLICIT INTEGER (A-Z) statement in your program. The FORTRAN 77 parameter file explicitly declares all parameters as integers.

After you compile your program, link it with the subroutine library, using the following command line:

```
F77LINK programname CLREINF.LB
```

Remember that if you haven't merged ICALL.OB into the system library, URT.LB, then you must include it on the LINK line after CLREINF.LB. This command line follows:

```
F77LINK programname CLREINF.LB ICALL
```

DG/L Considerations

To use the INFOS II interface with your DG/L program, you must have Read access to IIPAR.DG, the source language parameter file, and CLREINF.LB, the library of interface subroutines. These files are normally located in the directory :UTIL.

In order to define the interface variables, you must include the following command line at the beginning of your source program:

```
INCLUDE IIPAR.DG;
```

When you declare your variables, you must also declare each interface subroutine that you use. Declare interface subroutines as external procedures. For example, if you want to use the subroutines IOPEN, IISET, IRELREAD, and ICLOSE, you declare the following:

```
EXTERNAL PROCEDURE IOPEN, IISET, IRELREAD, ICLOSE;
```

Then, after compiling your program, link it with the following CLI command line to define the interface subroutines:

```
XEQ LINK programname CLREINF.LB [DGLIB] ICALL
```

Finally, note that neither the INFOS II system nor the interface takes full advantage of DG/L's string data types. This affects the interface in two places: filenames and data areas. String constants will work as you expect them to; however, you must be careful with any string variable values that the interface will return to you on a Read procedure.

Note the following important considerations as you code your programs:

- You must place a null <0> at the end of the filename string you use in IOPEN. For example, your IOPEN subroutine might read:

  ```
  IOPEN (IARRAY , "MY_FILE<0>" , 2, IER);
  ```

- When you input a data area to the system, you must have previously specified the length. For example, IIREC must have a previously defined IIRLEN. You cannot expect the interface to know about the current length of strings.
- Remember that you must declare the values of data areas as STRING types.

- The interface will not modify the current length of strings when it outputs data to a data area. (This operates like a Read from a file.) For example, if you wish to use RECORD as a string after an IKEYREAD call, you must perform the following:

```
IIGET (iarray,IIRLEN,LEN,ier);

SETCURRENT (RECORD,LEN);
```

- Do not use SUBSTR function calls in argument pairs.

PL/I Considerations

To use the INFOS II interface with your PL/I program, you must have Read access to IIPAR.PL1, the source language parameter file, and CLREINF.LB, the library of interface subroutines. Normally you will find these files in the directory :UTIL.

To define the interface variables, you must include the following line at the beginning of your source program:

```
%INCLUDE "IIPAR.PL1";
```

This file declares the interface variables and all external entries that the interface needs.

After compiling your program, you must link to all external procedures with the following CLI command line:

```
PL1LINK programname CLREINF.LB ICALL
```

Also notice that neither the interface nor the INFOS II system takes full advantage of PL/I's many data types. In general, the interface can easily handle any word-aligned data type, as described in the *PL/I Reference Manual.* Note, however, that the interface expects the following conditions:

- All numbers, including the error status variable, ier, should be in Fixed Binary format. For example:

```
DECLARE (IER, IPT, I)    FIXED BINARY (15);
```

- The interface variables are BIT (16) ALIGNED values, as defined in IIPAR.PL1.
- Data areas should be CHARACTER (n) ALIGNED, although you can use any word-aligned array or structure. For example:

```
DECLARE MY_KEY    CHAR (15) ALIGNED;
```

- You cannot use CHARACTER VARYING type variables.
- You cannot use SUBSTR functions in argument pairs.
- Make sure that the filename you use in the IOPEN call is word-aligned. Terminate the filename with a null character <0> in your IOPEN call. For example:

```
CALL IOPEN ( IARRAY , "INFOS_FILENAME<0>"C, 2, IER );
```

CAUTION: Avoid the direct passing of literals in the IISET subroutine in procedures below the main procedure. Because of the way in which PL/I maintains addresses on the stack, your variables might not remain set to the literals you specify. It is valid to use global variables in IISET or to use literals when you call IISET from the main program.

Error Codes

There are three types of error messages that you might receive while using the INFOS II interface:

- AOS system error codes.
- INFOS II system error codes.
- INFOS II interface error codes.

The system error message file, ERMES, contains all of these error codes. You can gain access to them with the CLI MESSAGE command. Note, however, that the code number returned to FORTRAN 5 and DG/L programs is equal to that of the other two languages, plus 3. Therefore, you might want to write a macro or program that will take the decimal error code as input, subtract 3 from it, and then perform the call to the CLI MESSAGE command.

Some common INFOS II system (runtime) error codes are listed in the source language parameter files that are distributed with the interface. We also include the parameter files in Appendix B.

Finally, we define the error codes for the language interface itself in Table 8-12. If the interface detects one of these errors, it stops processing the current interface subroutine and ignores any following argument pairs. Note, again, that you should include the ier argument in all of your interface subroutines. If the interface returns any value other than IONORM, it means that some condition exists which requires your attention.

Table 8-12. Language Interface Error Codes

Code		Meaning
F5 DG/L	F77 PL/I	
3910	3907	The first argument in the interface subroutine is not an initialized interface array. Either you have not previously performed a successful IOPEN, or your processing has overwritten the interface array.
3911	3908	The first element of an argument pair is not an interface variable.
3912	3909	The value you have specified for the second element of an argument pair is too large. Most likely, this means that the value of IINLEV exceeds the maximum number of levels you selected in the IOPEN call.
3913	3910	The second element of an argument pair has tried to set bits it is not allowed to. Make sure that you use values from your source language parameter file.
3914	3911	The number of levels specified in IOPEN is not between 1 and the maximum number of levels specified when you created the file.
3915	3912	The first element of an argument pair does not apply to this interface subroutine.

Using AOS INFOS II with Business BASIC

The Business BASIC INFOS II interface is substantially different from the interfaces of other Data General products we discussed earlier in this chapter. The Business BASIC interface consists of a set of statements that behave like regular Business BASIC commands, although their formats are different.

You can use the Business BASIC INFOS II statements both as commands and as program statements. For each statement, there are applicable arguments which function similar to switches in that they alter the statement in various ways. Some arguments are required for certain statements, while others are optional or not applicable to certain statements.

The INFOS II statements are designed to allow you easy access to INFOS II files. Wherever possible, statements will take default values for arguments, freeing you from having to enter excessive information with every statement. You can also redefine default values with one of the INFOS II statements, DBSET.

You should note that you cannot create an INFOS II file with AOS Business BASIC. You must use the AOS INFOS II utility ICREATE. Once you have created your index and database, Business BASIC allows you to manipulate your INFOS II file structure with all the functionality described in this manual.

Argument Pairs

As we mentioned above, every Business BASIC INFOS II statement has applicable arguments. You supply these arguments in pairs. Most pairs are optional and many have default values that the system supplies if you don't include arguments. There are five types of argument pairs; they do the following operations:

- Select INFOS II options.
- Set numeric values.
- Return numeric values.
- Specify string variables.
- Handle errors.

Your *Business BASIC Commands, Statements, and Functions* manual describes all of the arguments.

Selecting INFOS II Options

Argument pairs that select INFOS II options have a statement *keyword* to the right of the equals sign. For example:

```
ACCESS=REL
```

means that you want to use relative access. REL is a keyword. Likewise, the following examples indicate forward (FWD) motion and database suppression.

```
MOTION=FWD        SDBASE=YES
```

The *Business BASIC Commands, Statements, and Functions* manual describes the argument and keyword combinations you can use.

Setting Numeric Values

For this group of argument pairs, the right-hand value must be a numeric constant or numeric variable. You can subscript numeric variables. The INFOS II parameter on the left-hand side of the argument pair is set to the value you specify on the right-hand side at the time the statement is executed.

For example, if you want to specify an occurrence number of 3, you use the following argument pair:

```
OCCUR=3
```

To set the number of keyed access levels you want to use on the next INFOS II access, you use the LEVELS parameter. In the next example, we set LEVELS to the numeric value stored in the variable SOME_NUMBER.

```
LEVELS=SOME_NUMBER
```

Likewise, if we want to perform an inverted operation, we can set data record feedback information to a value contained in a subscripted variable:

```
FEEDBACK=FD[FNA(I)+2]
```

Specifying String Variables

Argument pairs that specify string variables must have a nonsubscripted string variable as the right-hand argument. The string is associated to the INFOS channel string, and the string variable, *not its contents,* is transmitted to INFOS. For example, look at the following program fragment.

```
0100 LET KVAR$="DECREPID COMPUTERS"
0110 DBSET F$,KEY=KVAR$,REC=RVAR$
0120 LET KVAR$="DATA GENERAL"
0130 DBREAD F$,ACCESS=KEY
0140 PRINT RVAR$
```

At the time the interface executes the DBSET statement (line 0110), the contents of KVAR$ are not significant. KEY is set to the string KVAR$, not to its contents. Subsequently, the content of the string KVAR$ changes; the variable now reads DATA GENERAL instead of DECREPID COMPUTERS. Therefore, when statement 0130 executes, the key is DATA GENERAL.

Returning Numeric Values

For argument pairs that will return numeric values, on the right-hand side use a variable into which INFOS II will return the requested value. You can subscript the variable. For example, if you want to know the subindex level where the last INFOS II access occurred, you can use the following argument pair with the DBGET statement:

```
SILEVEL=INDEX_LEV
```

The interface will return the subindex level to the variable INDEX_LEV.

Handling Errors

With any of the Business BASIC INFOS II statements, you can use the error-handling argument pair: ERR=linenumber, where linenumber is any valid statement number. Whenever one of the INFOS II statements detects a user error, the INFOS II system checks for the following conditions. If it finds one of these conditions to be true, it stops checking and issues the appropriate error message.

1. If the user issues the INFOS II statement as a command, INFOS II will send the appropriate error message to the user's screen.
2. If the user's statement contains the error-handling argument pair, INFOS will perform a GO-TO to the line number specified in the argument pair. If the line number does not exist, then INFOS will generate a LINE NUMBER error.
3. If the user's program has an ON ERR condition in effect, it will perform the THEN clause of that statement.
4. If none of the above conditions is true, INFOS II will direct the error message to the user's screen.

The Business BASIC INFOS II Statements

The *Business BASIC Commands, Statements, and Functions* manual contains explanations of all the INFOS II statements you can use with Business BASIC. However, we list them in Table 8-13 with a brief description, for your reference.

Table 8-13. The Business BASIC INFOS II Statements

BBASIC Statement	Function	Corresponding INFOS II Command
DBCLOSE	Close an open INFOS II file.	CLOSE
DBDELETE	Delete a key and/or record.	DELETE
DBGET	Get values returned by INFOS II.	None
DBOPEN INFOS	Open an INFOS II index.	OPEN
DBREAD	Read from an INFOS II file.	READ
DBREINSTATE	Reinstate a logically deleted record.	REINSTATE
DBRELEASE	Release locks and/or position.	RELEASE LOCKS/ POSITION
DBRETRIEVE HIGH KEY	Retrieve the highest key in the current subindex.	RETRIEVE HIGH KEY
DBRETRIEVE KEY	Retrieve a key.	RETRIEVE KEY
DBRETRIEVE SIDEF	Retrieve a subindex definition.	RETRIEVE SUBINDEX DEFINITION
DBRETRIEVE STATUS	Return the INFOS II status.	RETRIEVE STATUS
DBREWRITE	Rewrite an INFOS II database record.	REWRITE
DBSET	Reset the default value of an INFOS II parameter.	None
DBSUBINDEX DEFINE	Define a subindex.	DEFINE SUBINDEX
DBSUBINDEX DELETE	Delete a subindex.	DELETE SUBINDEX
DBSUBINDEX LINK	Link a subindex.	LINK SUBINDEX
DBSUBINDEX LINKINIT	Initialize a link subindex string.	None
DBSUBINDEX LINKSET	Set parameters in a link subindex string.	None
DBWRITE	Write to an INFOS II file.	WRITE

Important Considerations and Errors

In this section we note some important things for you to keep in mind while using the Business BASIC interface, and some errors you might receive.

Channel String

It is important to note that you must use only the Business BASIC INFOS II statements to gain access to and modify the channel string and the link subindex string. If you attempt to modify either of these strings with a LET statement you could inadvertantly modify an index or database.

INFOS II Channels

If you issue the Business BASIC CLOSE statement without any arguments or the NEW statment, all INFOS II Channels will be closed.

Business BASIC currently allows up to 16 open INFOS II channels. If you attempt to exceed this limit, you will receive the error *NO MORE CHANNELS AVAILABLE*.

Locks

If you want to lock records, you must set the NLOCKS=number argument pair of the DBOPEN INFOS statement to a value greater than zero. This argument pair specifies how many outstanding locks you can have at any one time. If you do not set this value, and you attempt to lock a record while writing it, you will receive the *LOCK LIMIT EXCEEDED* error message and the record will be created with no locking capacity.

If you use the DBSET statement with the LOCK=YES argument pair, you must specify either GLOBAL=YES (for data record lock) or LOCAL=YES (for partial record lock). Do not specify both.

DBDELETE

If you want to use the DBDELETE statement with the DELETE=LOG argument pair (for a logical delete), you must specify GLOBAL=YES (data record logical delete), LOCAL=YES (partial record logical delete), or both. Note that there is no error message to inform you of your syntax error if you fail to do this.

DBRETRIEVE KEY

You must use the REC=string-variable argument pair with a DBSET or DBRETRIEVE statement prior to issuing the DBRETRIEVE KEY statement. Otherwise, you will receive an error message for invalid statement or command syntax.

Parameter Range Error

This error occurs when you attempt to set an INFOS II parameter to a value that is not in the range allowed by INFOS II. For example, INFOS II allows you to specify up to 255 levels of subindexes, although it only supports 32 (levels 0 to 31). If you indicate a value greater than 254, you will receive the Parameter Range Error. If you specify a value of 32 to 254, Business BASIC passes the value to INFOS II and you will receive an INFOS error.

INFOS II Error

Whenever an INFOS II error occurs, SYS(7) returns a value of 83, and SYS(31) contains the actual error code. You can use this in conjunction with the AERMS$ function to retrieve the INFOS II error message text. Consult your Business BASIC manual.

End of Chapter

Chapter 9
The INFOS II Assembly Language Interface

This chapter describes how you communicate with the INFOS II system at the assembly language level. It assumes that you are familiar with Chapters 1 through 7 of this manual. You will also find the following manuals useful with this chapter:

- *Programmer's Reference Manual, ECLIPSE®-Line Computers.*
- *Advanced Operating System (AOS) Programmer's Manual.*
- *AOS Macroassembler Reference Manual.*
- *AOS & AOS/VS CLI User's Manual.*
- *AOS Binder User's Manual.*

INFOS II System Macros

You communicate with the INFOS II system with the aid of the macros contained in the file PARAU.SR, not with MASM macros. You can include the PARAU.SR file on the CLI command line when you invoke the AOS Macroassembler (MASM), as in the following example:

```
)XEQ MASM PARAU.SR programname ⤸
```

Or you can use a MASM.PS file built with PARAU.SR, as in this example:

```
)XEQ MASM programname ⤸
```

In general, you'll issue an INFOS II system macro call for some request, such as a READ, and specify the arguments for that call in a software structure called a *packet*. A packet is a formalized structure in which you specify arguments for any given INFOS II macro call. We will discuss packets in greater detail later in this chapter. PARAU.SR contains a special-purpose macro, ?PACKET, which generates the appropriate packet for each processing macro call that you issue.

You can use all the AOS CLI options and switches you need to invoke MASM when you assemble a program containing INFOS II macro calls. We recommend that you check your source program listing carefully, since the Macroassembler's error detection facilities cannot detect illegal INFOS II numeric parameter values, though they will detect such errors as misspelled INFOS II parameter symbols.

Linking the Object File

Link your object file into a relocatable binary file with the AOS linker. When you invoke it, you must allocate a stack and include the object filename, ICALL.OB, in the CLI command line. The stack and ICALL.OB allow the INFOS II system and AOS to communicate. The INFOS II system requires 110 words on the stack; therefore, you should make the stack large enough for both the system's needs and your own. Invoke the linker with the following command line:

```
XEQ LINK/Z=110 programfilename ICALL.OB
```

Or, you can make ICALL.OB a permanent part of the system's runtime library, URT.LB, and invoke the linker with the following:

```
XEQ LINK/Z=110 programfilename
```

In this case, ICALL.OB will be extracted from URT.LB.

You can use all the AOS CLI options and switches when linking an object program to process an INFOS II file.

Packets and Tables

Your program and the INFOS II system communicate with each other at the assembly language level through software structures called *packets* and *tables*. A packet is a group of logically related control parameters. From it, you specify those features that you want to use in a particular system call. For example, to execute an INFOS II READ command, your program must supply a Processing Packet which tells the system both how to find a record and where to place it in memory.

The INFOS II system defines six different types of packets and two types of tables. We summarize the packets in Table 9-1.

Table 9-1. INFOS II Request Packets

Packet	Use
File Definition Packet (FDP)	Specify parameter values for index and database files when you open them for creation or update processing.
Volume Definition Packet (VDP)	Specify parameter values for index and database volumes.
Processing Packet (PP)	Specify processing function options.
Key Descriptor Packet (KDP)	Specify parameter values for a key to be used in a processing function.
Subindex Definition Packet (SDP)	Specify parameter values for a subindex.
Link Subindex Packet (LSP)	Specify the source and destination keys that will be linked to the same subindex.

The INFOS II tables consist of *volume* tables and *key* tables. A volume table is a chain of one or more Volume Definition Packets (VDPs) in the address space contiguous to the end of a File Definition Packet (FDP). Similarly, a key table is a linkage of one or more Key Descriptor Packets (KDPs) contiguous to a Processing Packet (PP).

Packet Configurations

The following diagrams illustrate the combinations of packets you can use to make different requests. In these diagrams, we represent packets and pointers in this way:

Symbol	Meaning
⟶	Word Pointer
▭	Packet

FDP/VDP Packet Structure

Use the File Definition Packet (FDP) and the Volume Definition Packet (VDP) when you create an INFOS II file. The FDP defines the parameters for the file, while the VDP defines the parameters for the specific volume. The INFOS II file is an AOS directory and the INFOS II volumes are AOS files. Use only the FDP if you want to open an existing INFOS II file.

Figure 9-1 depicts the FDP/VDP packet structure.

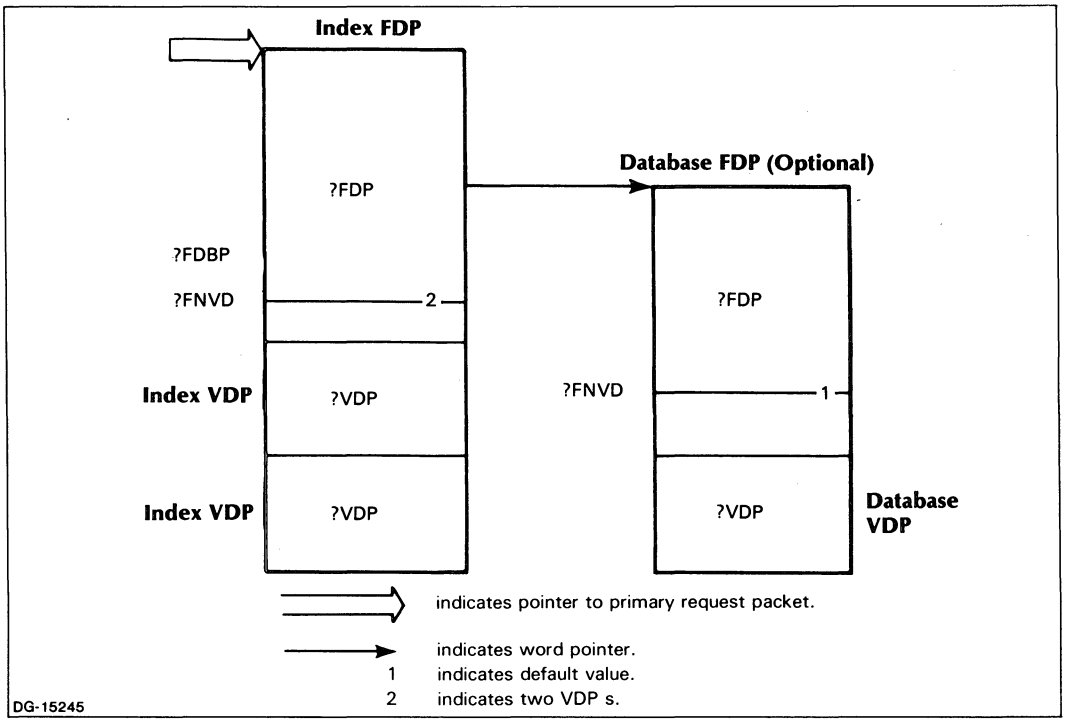


Figure 9-1. FDP/VDP Packet Structure

When you fill these packets with default parameter values (zeros), they will create a standard INFOS II file. If you want to specify certain parameters other than the defaults, enter the appropriate values.

PP/KDP Packet Structure

Use the Processing Packet (PP) and the Key Descriptor Packet (KDP) to specify parameters for the INFOS II processing requests. At least one KDP *must* follow the PP, whether you select keyed access, relative access, or a combination of both. The KDPs follow the PP contiguously in memory.

Figure 9-2 illustrates PP/KDP packet structure.

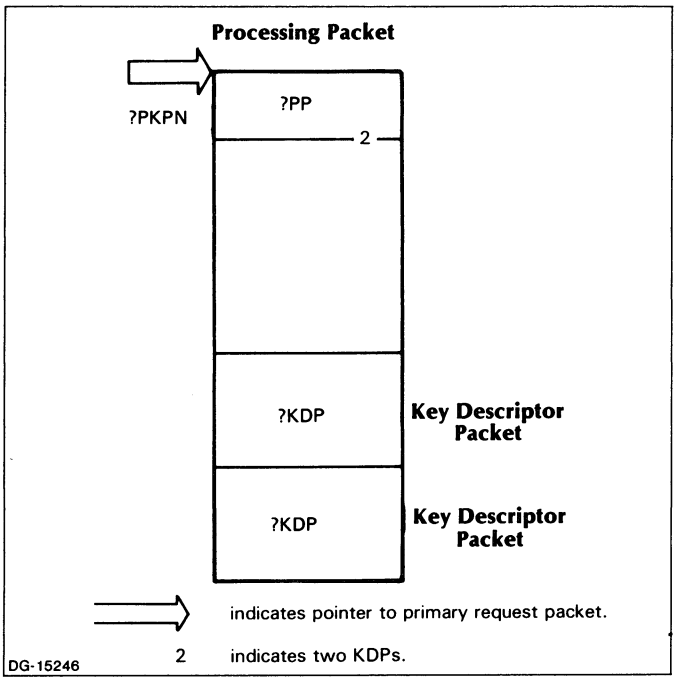


Figure 9-2. PP/KDP Packet Structure

PP/SDP Packet Structure

The Subindex Definition Packet (SDP) is an extension to the above structure. The DEFINE SUBINDEX and RETRIEVE SUBINDEX DEFINITION commands use this packet structure. When you want to define a subindex, you use the SDP to specify parameter values. When you retrieve subindex information, the INFOS II system returns to you the parameters you have specified in the SDP.

This structure appears in Figure 9-3.

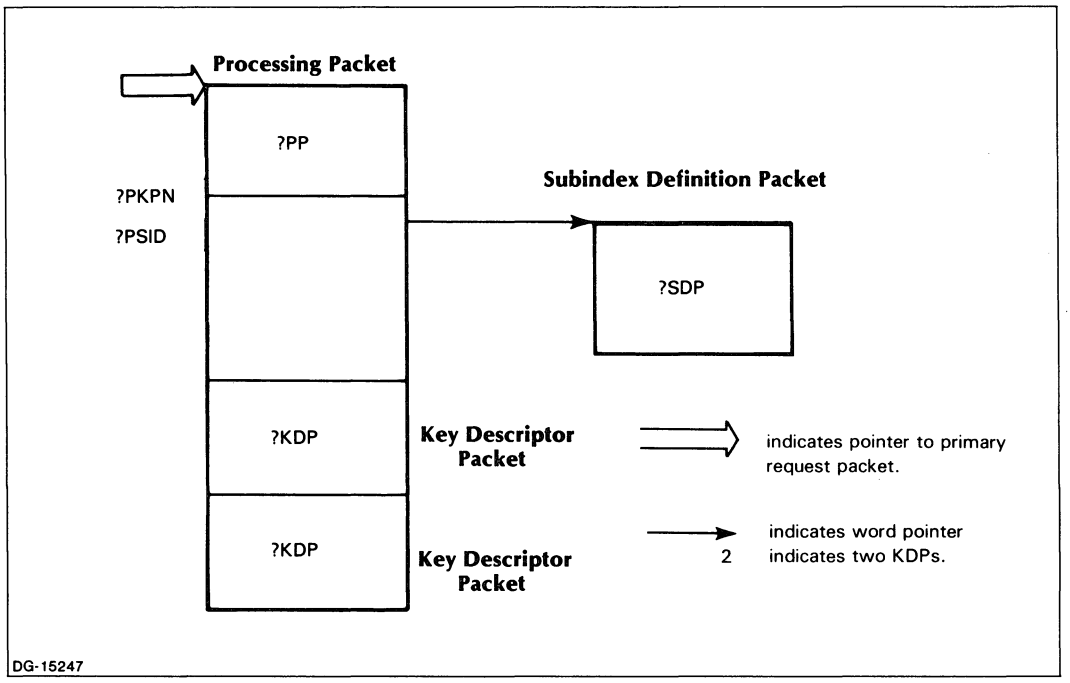


Figure 9-3. PP/SDP Packet Structure

LSP Packet Structure

The LINK SUBINDEX command uses the Link Subindex Packet (LSP). This packet contains the information required to gain access to the source and destination keys.

The LINK SUBINDEX command also uses two sets of KDPs. The source KDPs form the source key descriptor table (KDT); they follow the LSP contiguously in memory. The destination KDPs form the destination KDT. (Remember from Chapter 3 that the source key is the key that contains the subindex to be shared, and the destination key is the key that is to be linked to that subindex.)

Figure 9-4 shows the structure of the LSP and its two sets of KDPs.

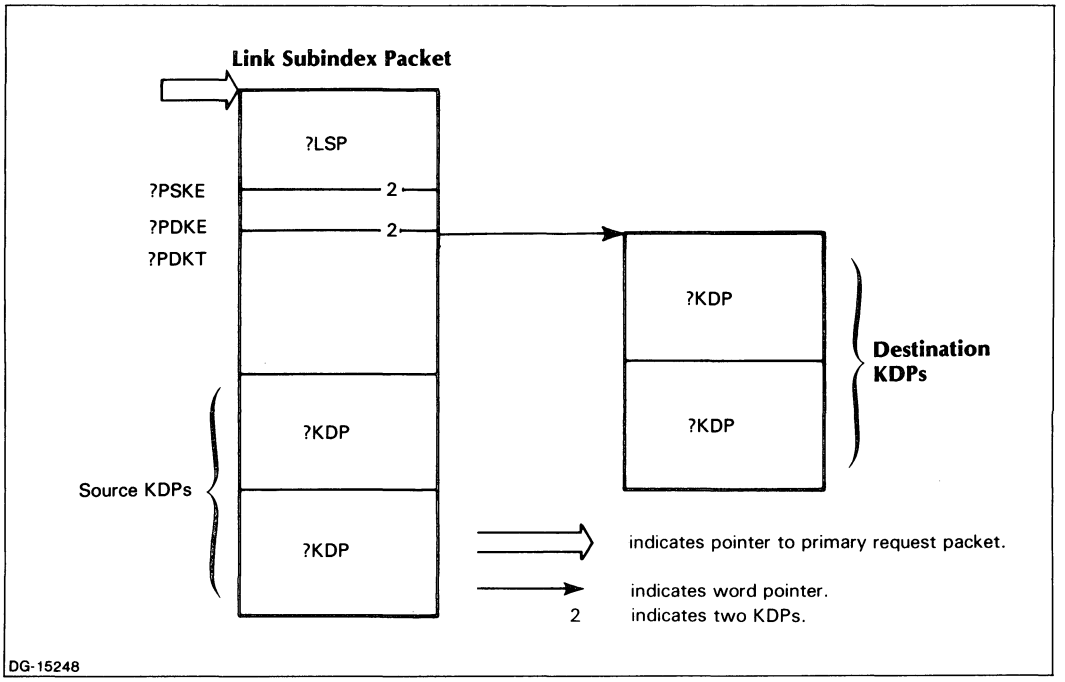


Figure 9-4. LSP Packet Structure

Packet Descriptions

The following sections and illustrations detail each packet's format and parameter locations. Within each packet, at offset ?TYPE, is the packet type word. This determines the packet's length and internal structure.

When you generate any INFOS II packet, the ?PACKET macro automatically makes the contents of reserved areas zero. Do not alter the values of any areas that we have marked as reserved. We include reserved areas as shaded sections in the packet diagrams.

Refer to PARAU.SR to find the various masks that are applicable to the INFOS II packets.

File Definition Packet

Table 9-2 contains parameters of the File Definition Packet (FDP), their standard values, range of values, and, where applicable, the relationships between them. Table 9-3 details the file definition flags (?FFLG), and Table 9-4 describes the index flags (?FIFL). We illustrate the packet in Figure 9-5.

The following are packet identification flags:

?FDP File Definition Packet (revision and type).

?FDPL File Definition Packet Length.

Table 9-2. File Definition Packet Parameters

Offset	Parameter	Definition
0	?TYPE	Packet Type — When you invoke the ?PACKET macro to generate an FDP, you must specify ?FDP. Thus you make the value of this field ?FDP.
1	?FCHN	Channel Number — The INFOS II system automatically returns a channel number to this field at the successful completion of an OPEN request. This number links the issuing process to the file. Consequently, all subsequent processing calls issued by the opening process must specify this channel number. A file can be opened multiple times; each open call results in a unique channel number. NOTE: The system returns a channel number in an index FDP only, never a database FDP.
2	?FNAM	Filename Byte Pointer — You must specify a byte pointer to an index filename, whether you are opening the file for creation, inversion, or update processing. When creating a nonstandard file, you can specify a byte pointer to a unique database filename in the database FDP, or you can use the default database filename (indexfilename.DB). Note that if you choose to specify a filename for the database in the FDP, you must use its simple filename only, *not* a pathname. You can specify either the filename *or* the pathname for an index file, however.
3	?FPAG	Page Size — The default page size for both index and database files is 2048 bytes, when you create a file. The alternative for both files is 4096 bytes. These values apply to an index FDP when you invert a file. For database files, the smaller page size allows you to write data records up to 2040 bytes in length, while the larger page size can accommodate 4088-byte records. When you open an existing file, the INFOS II system returns the page size value to your program.
4	?FFLG	File Definition Flags — We describe the file definition flags in Table 9-3.

(continues)

Table 9-2. File Definition Packet Parameters

Offset	Parameter	Definition
5	?FNVD	Number of Volumes — This field applies in both an index and database FDP when you create a new INFOS II file. It also applies in an index FDP when you invert a file. The default number of volumes for an index and database is one each. When you create or invert a file, you can specify up to 32 volumes for either or both files. NOTE: If you default this field in either FDP, you need not supply a volume table. However, if you specify one or more volumes for either file, then you must include one Volume Definition Packet (VDP) for each volume you specify, in the respective table. When you open an existing file, the INFOS II system returns the number of volumes to your program.
6	?FNIL	Maximum Number of Index Levels — This field applies in a DBAM index FDP only when you create or invert a file. The default is one level of subindexes below the main (level 0) subindex. You can, however, specify up to 32 levels. If the access method is ISAM, the INFOS II system ignores the contents of this field. When you open an existing file, the system returns the number of levels to you.
7	?FVER	Parameter Version Number — The ?PACKET macro automatically fills this field with ?PVER when you create or invert a file. ?PVER indicates the system version number in which the file is created.
8	?FDRN	Reserved.
9	?FIRV	Reserved.
10	?FNLR	Maximum Number of Locks — This is a runtime parameter, applicable in an index FDP each time you open a file. It has no meaning in a database FDP. The default number of locks is 0, but you can specify up to 32.
11	?FDBP	Database FDP Word Pointer or Database Name Byte Pointer — This field applies in an index FDP only; it has no meaning in a database FDP. When you create a standard file, this field's contents must be zero. When you create a nonstandard file, its contents must be a word pointer to the database FDP or zero if you do not generate a database FDP. When you invert a file, this field's contents must be a byte pointer to the name of the database you are inverting. The value of ?FFDP, a File Definition Flag (see Table 9-3), determines the contents of this field.

(continued)

Table 9-2. File Definition Packet Parameters

Offset	Parameter	Definition
12	?FRNS	Index Root Node Size — This field applies in an index FDP only when you create or invert a file. It applies only to the main index (level 0). The default root node size is 6 bytes less than the page size. If you specify an alternative root node size, it must be at least large enough to contain three maximum length index entries, but cannot be greater than the default (6 bytes less than page size). We describe index entry length and root nodes in Chapter 10.
13	?FMKL	Maximum Key Length — This field applies in an index FDP only when you create or invert a file, and it applies only to the main index (level 0). The standard key length is 255 bytes, but you can specify from 1 to 255 bytes.
14	?FRMF/ ?FPRL	Root Node Merit Factor/Partial Record Length — In an index FDP this field applies to the main (level 0) index only. You can specify the field when you create or invert a file. The first 8 bits of the field are the root node merit factor. The default is 0, but you can specify a root node merit factor from 1 to 32. See Chapter 10 for more information on the root node merit factor. The second 8 bits of the field are the partial record length. You can specify a partial record length from 0 to 255 bytes. The default is 0. See Chapter 2 for more information on partial records. You can specify the entire field with either parameter name. Be careful when altering information in one section of the field not to alter information in the other section unintentionally.
15	?FIFL	Index Flag Word — We describe the index flags and their default and optional values in Table 9-4.

(concluded)

Table 9-3. File Definition Flags (?FFLG)

Bit Position	Flag	Description	Default	Option
0	——	Reserved.		
1	?FSCP	Reserved.		
2	?FNCP	Reserved.		
3	?FRCV	Reserved.		
4	?FSPM	Space Management — This flag is meaningful at file creation only. You can set it in either the index FDP, the database FDP, or both.	0 (No)	1 (Yes)
5	?FEIO	Exclusive Index Open — This is a runtime flag that you can set each time you open an INFOS II file. It is applicable only in an index FDP; it has no meaning in a database FDP.	0 (No)	1 (Yes)
6	?FEDO	Exclusive Database Open — This is a runtime flag that you can set each time you open an INFOS II file. It is applicable only in an index FDP and has no meaning in a database FDP.	0 (No)	1 (Yes)
7 - 8	?FAM1 ?FAM2	Access Method Field — This 2-bit field applies only in an index FDP when you're creating or inverting a file. The right bit of the field can take the value for ISAM (?FISM) or DBAM (?FDBM). The default access method is ISAM. So, if you want to create a DBAM file (or a DBAM index when inverting) you must specify ?FDBM.	0 (ISAM)	1 (DBAM)
9	?FRDO	Read Only Open — You can set this bit mask to specify a READONLY OPEN. It will prevent any modification operation (such as WRITE) from succeeding on the returned channel number. If you attempt such an operation, the system will return the error message: *(IOACE) ACCESS DENIED - ACL ERROR ON READONLY OPEN.*	0 (No)	1 (Yes)
10	?FINV	Invert File — This flag applies only when you create a new index for an existing database. ?FFDP can be either 0 or 1, and ?FDBP must be either a byte pointer to databasename or a word pointer to the database FDP.	0 (No)	1 (Yes)

(continues)

Table 9-3. File Definition Flags (?FFLG)

Bit Position	Flag	Description	Default	Option
11	?FESR	Reserved.		
12	?FFCR	Create New File Mode — This flag applies in an index FDP only; it has no meaning in a database FDP. The default processing mode is update, so if you want to create or invert a file you must specify ?FFCR.	0 (Update)	1 (Create)
13	?FCDR/ ?FCKY	Data Record Compression/Index Key Compression — These flags are meaningful only at file creation. ?FCDR applies to the database FDP only. ?FCKY applies only to the index FDP. You must set them if you want compression.	0 (No)	1 (Yes)
14	?FORD	Optimized Record Distribution — This flag is meaningful at file creation only. You must set it if you want optimized record distribution.	0 (No)	1 (Yes)
15	?FFDP	Database FDP Present Flag — This flag applies only in an index FDP; it has no meaning in a database FDP. The flag must be 0 for a default database file, 1 for a nonstandard database file. When the flag is 0, it means one of two things: · The pointer in the field ?FDBP is a byte pointer to a databasename which is the case when you are inverting); or · You left the pointer as null. The database filename is the standard: indexfilename.DB. When the flag is set (1), it means that the pointer in the field ?FDBP is a word pointer to a database FDP. The value of this field determines the contents of the FDP parameter ?FDBP.	0 (default)	1 (non-default)

(concluded)

Table 9-4. The Index Flags (?FIFL)

Bit Position	Flag	Description	Default	Option
0	——	Reserved.		
1	?SNSI	No Subindexing — This flag applies in an index FDP when you create or invert a file. If your access method is ISAM, the INFOS II system will set this flag to 1, which prevents subindexing. Do not set this flag if your access method is DBAM, or else you will not be able to create subindexes, regardless of the number you specified in ?FNIL.	0 Allow Subindexes	1 No Subindexes
2	——	Reserved.		
3	——	Reserved.		
4	?SDUP	Duplicate Keys Allowed — This flag applies in an index FDP only when you create or invert a file. It applies only to the main index (level 0). The default is no duplicates, so if you intend to use duplicate keys in the main index, you must set ?SDUP to 1.	0 No Duplicates	1 Allow Duplicates
5	?SUID	Start of User ID Field — This field occupies the bit positions defined in the mask ?SUIM. It applies in an index FDP only when you are using the IRECOVER utility to recover a database after a system failure. IRECOVER output reports whatever value you put in this field. See Chapter 9 for more details.	N/A	N/A

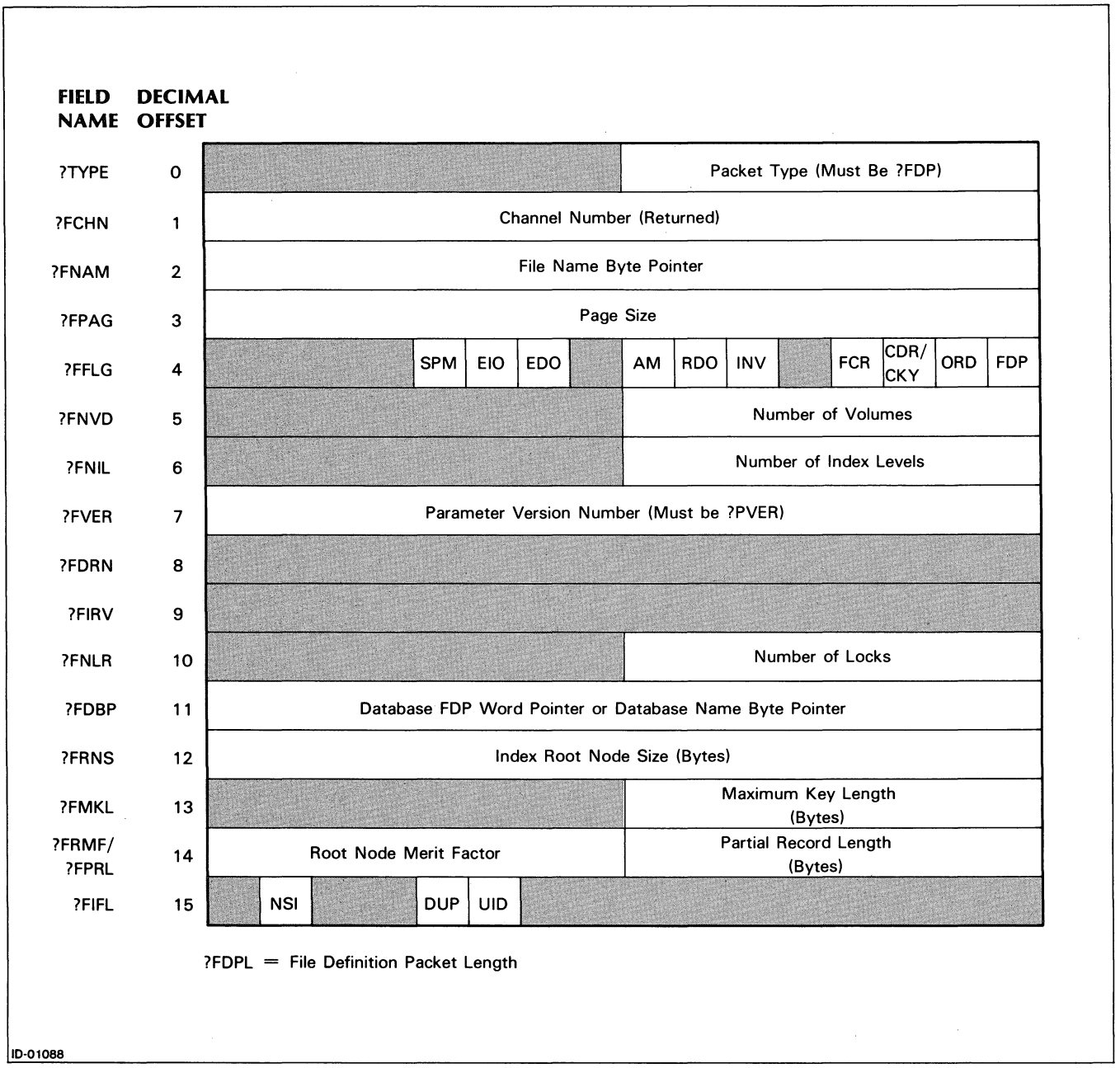


Figure 9-5. File Definition Packet (FDP)

Volume Definition Packet

Table 9-5 contains the parameters of the Volume Definition Packet (VDP), their standard values, and, where applicable, the relationships between them. We illustrate the packet in Figure 9-6. The following are packet identification flags:

?VDP Volume Definition Packet (revision and type).

?VDPL Volume Definition Packet Length.

Table 9-5. Volume Definition Packet Parameters

Offset	Parameter	Definition
0	?TYPE	Packet Type — When you invoke the ?PACKET macro to generate a VDP, you must specify ?VDP. The value of this field then becomes ?VDP.
1	?VVNP	Volume Name Byte Pointer — You can specify a byte pointer to the name of the volume defined in this VDP. We recommend, however, that you use the default name, which you will get with a null pointer. The default volume name is always VOLnn, where nn is a decimal number between 01 and 32, which indicates this volume's ordered position in its volume table.
2 3	?VVSH and ?VVSL	Volume Size in Blocks: High Order/Low Order — Standard volume size depends on file page size. For the smaller page size, standard volume size is more than 2 million 512-byte blocks. For the larger page size, standard volume size is greater than 4 million 512-byte blocks. To specify a volume size less than or equal to 65,535 blocks, fill ?VVSH with zeros and specify the number of blocks you want in ?VVSL. NOTE: If you specified space management for the file, the INFOS II system will use the first two pages on each volume of the file for an available space table. If you don't specify space management, the system will reserve the first two pages on each volume for its own use.
4	?VVMF	Volume Merit Factor — The volume merit factor can be any number from 1 to 32. You must use a number that is equal to or less than your previously created volumes. The volume merit factor only applies to files you are using with optimized record distribution. See Chapter 10 for more information on the volume merit factor.
5	?VVEL	Volume Element Size — Default element size is 32 blocks. To ask for a different size, specify a multiple of 4 if the file uses the smaller page size, or a multiple of 8 if it uses the larger page size. In either case, the element size cannot exceed the size of a Logical Disk (LD), which is established through the AOS Disk Formatter utility. If you do not specify a multiple of 4 or 8, the INFOS II system rounds your number down to the next lower appropriate multiple. NOTE: To allocate a contiguous volume, make volume and element size equal. The largest contiguous file AOS allows is 65,535 blocks.
6	——	Reserved.
7	——	Reserved.

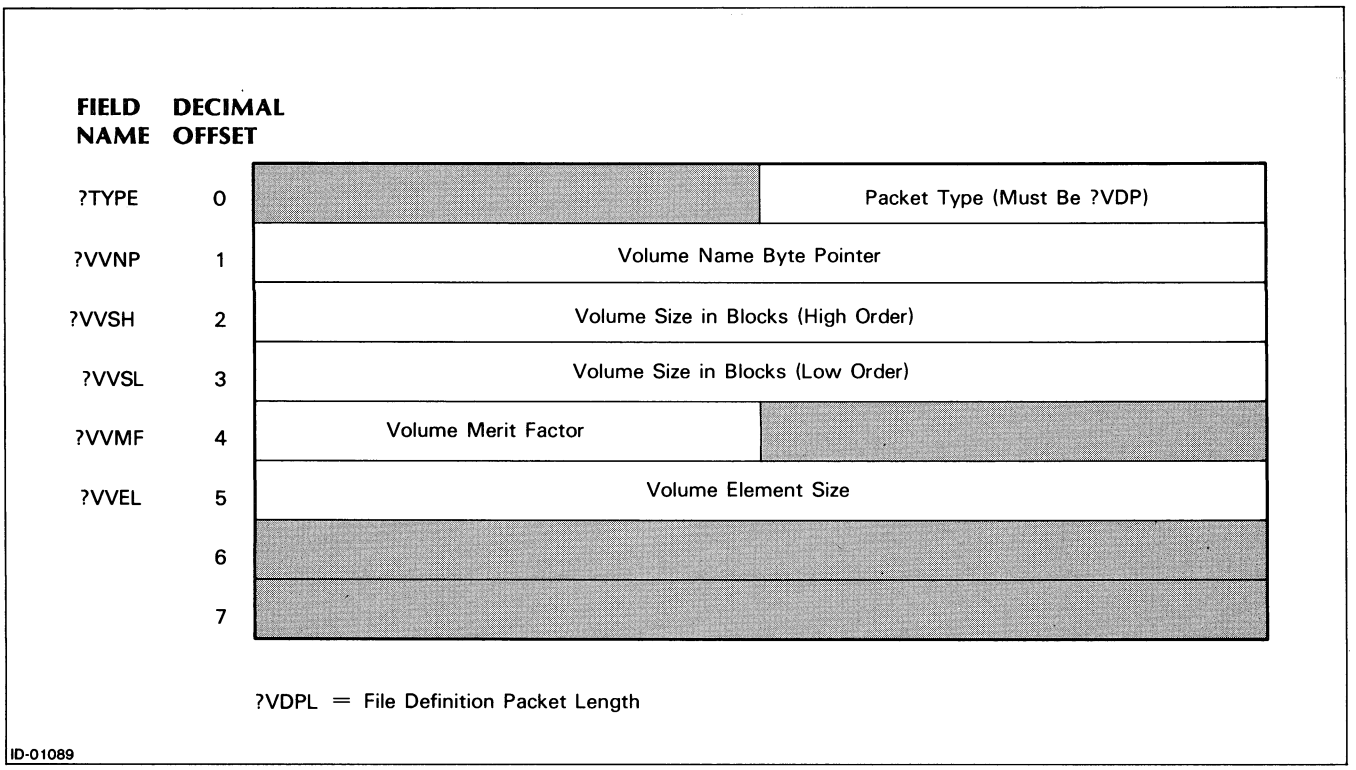


Figure 9-6. Volume Definition Packet (VDP)

Processing Packet

Tables 9-6, 9-7, 9-8, and 9-9 contain the parameters of the Processing Packet (PP), their standard and optional values, and, where applicable, the relationships between them. We illustrate the packet in Figures 9-7 and 9-8. Table 9-10 shows which command control word flags you can, must, or cannot use for a processing request.

The following are packet identification flags:

?PP Processing Packet (revision and type).

?PPL Processing Packet Length.

Table 9-6. Processing Packet Parameters

Offset	Parameter	Definition
0	?TYPE	Packet Type — When you invoke the ?PACKET macro to generate a processing packet, you must specify ?PP. The value of this field will then be ?PP.
1	?PCHN	Channel Number — Specify the channel number for which you issued the current request. The system assigned this number when you opened the file.
2	?PKPN	Number of KDPs in Key Table — Specify the number of Key Descriptor Packets in the key table for the current request. You must always specify at least 1, and you can have up to 32.
3	?PDAT	Byte Pointer to Data Area — Normally, on input or output, a data record is moved to or from the area pointed to in this field. If your request does not normally move data to or from your address space (for example, DEFINE SUBINDEX), set the value of this field to zero. If you suppress database access, the INFOS II system does not look at this field.
4	?PLEN	Record Length — On a WRITE or REWRITE, specify the record length in bytes. If you suppress database access on a WRITE or REWRITE, the INFOS II system does not look at this field. On an inverted WRITE or REWRITE, the system expects you to modify the contents of the data record to which you are linking, unless you suppress database access. Buffer Area Length — On a READ, the INFOS II system returns the length (in bytes) of the record moved to the data area. You can specify in ?PLEN the number of bytes you want to read and set the flag ?CSPL in the command control word (?PCCW). In this case, the INFOS II system moves only the specified number of bytes into your data area. If the record is longer than you specify, INFOS II sets the ?PRLO flag in the status word (?PSTU).

(continues)

Table 9-6. Processing Packet Parameters

Offset	Parameter	Definition
5	?PPRA	Byte Pointer to Partial Record Area — The system processes partial records only on READ, WRITE, and REWRITE operations. When the key you access for the current operation is in a subindex that allows partial records, specify the address of the area to which you want the partial record moved. Otherwise, suppress partial record processing by setting the flag ?CSPR in the command control word (see Table 9-7).
6	?PCCW	Processing Command Control Word — We describe the command control word flags and their standard and optional values in Table 9-7.
7	?PECW	Extended Processing Control Word — The extended control word flags and their standard and optional values appear in Table 9-8.
8 9	?PLKH ?PLKL	Data Record Feedback: High Order/Low Order — On every READ, WRITE, REWRITE, and RETRIEVE KEY request the INFOS II system returns the logical address of the data record associated with the accessed key. The feedback is a 32-bit internal format, binary integer. Use the feedback to link a data record to a new or existing key with an inverted WRITE or REWRITE command. Once a record is written, its feedback will remain the same, even if you subsequently rewrite the record.
10	?PSXL	Subindex Level — Every time you access a key, the system returns its subindex level number to this field. The subindex level is part of the index entry information that the system returns to you on the completion of certain commands.
11	?PSTU	Returned Status — We detail the returned status flags in Table 9-9.
12	?PMKL	Key Length — The INFOS II system returns the length of the key accessed for the current request to this field.
13	?PRMF/ ?PPRL	Record Merit Factor/Partial Record Length — Specify the merit factor of a record in the first 8 bytes of this field. In the second 8 bytes, the INFOS II system returns the partial record length of the key accessed for any operation. You can use either parameter name to specify the entire field.
14	?PSID	Word Pointer to Subindex Definition Packet — On a DEFINE SUBINDEX or RETRIEVE SUBINDEX DEFINITION operation, specify the address of the subindex definition packet in this field. For all other operations, this field should be 0.
15	?PXPP	Reserved.

(concluded)

Table 9-7. Processing Packet Command Control Word Flags (?PCCW)

Bit Position	Flag	Description	Default	Option
0	?CKEY	Keyed Access — You must specify keyed access for a WRITE request and for a physical DELETE operation. You can also specify it for any other request, except RELEASE LOCKS/POSITION.	0 (No)	1 (Yes)
1	?CREL	Relative Access — You may specify relative access for any operation except WRITE and physical DELETE. Note, however, that you can combine keyed and relative access when doing a WRITE or physical DELETE. When you use relative access, you must specify the direction of motion in the motion control field.	0 (No)	1 (Yes)
2 3 4	?CMC1 ?CMC2 ?CMC3	Motion Control — If you set ?CREL, you must specify one of the eight directions of motion shown in Figure 9-8. NOTE: Except for RELEASE LOCKS/POSITION and CLOSE, you must set either ?CKEY or ?CREL for each command that requires a processing packet. If you set both, then the motion control field should contain ?CDWN, ?CUP, or ?CSTA.	None	(See Figure 9-8)
5	?CSCP	Set Current Position — Except for CLOSE and RELEASE LOCKS/POSITION, you can set current position for each command that requires a processing packet. If you do so, the system will set the current position on the key accessed for the current request at the successful completion of that request. If you don't set the current position, your current position will be the same as it was before the current operation. NOTE: The RELEASE POSITION command requires a PP, but you don't specify anything except the channel number. The system resets the current position above the index.	0 (No)	1 (Yes)

(continues)

Table 9-7. Processing Packet Command Control Word Flags (?PCCW)

Bit Position	Flag	Description	Default	Option
6 7	?CLCK ?CUNL	Lock, Unlock — You can set either one of these flags on any READ, WRITE, REWRITE, or DELETE operation. If either lock or unlock is set, you must also set either or both of the flags ?CXPR and ?CXDB, to specify a lock or unlock of a partial or data record.	0 (No)	1 (Yes)
8	?CINV	Invert — You can set this flag on a WRITE request to link an existing data record to a new key or on a REWRITE request to link an existing data record to an existing key. Remember to suppress database access if you don't intend to update the record.	0 (No)	1 (Yes)
9	?CSPR	Suppress Partial Record — You can set this flag to suppress partial record processing on your current request if the request is a READ, WRITE, REWRITE, or RETRIEVE STATUS.	0 (No)	1 (Yes)
10	?CSPL	Limit Record Length — You can set this flag on a READ operation to limit the amount of data moved to your data area. (See the earlier description of ?PLEN.)	0 (No)	1 (Yes)
11	?CSDB	Suppress Database — If the current request is a READ, WRITE, REWRITE, or RETRIEVE STATUS, you can set this flag to suppress the movement of data to or from your data area. Suppressing database access on a READ request prevents the system from moving the record associated with the accessed key to your data area. Suppressing database on a WRITE request prevents the INFOS II system from writing a data record for the key you are writing to the index file. Suppressing database access on a REWRITE request indicates that you are not updating the data record associated with the accessed key. In this case, you would be updating the partial record.	0 (No)	1 (Yes)

(continued)

Table 9-7. Processing Packet Command Control Word Flags (?PCCW)

Bit Position	Flag	Description	Default	Option
12	?CLOG	Logical Delete Specifier — This flag applies only to a DELETE request. When you set this flag, the system marks the partial or data records as logically deleted; it does not physically delete them. You must also set ?CXPR or ?CXDB to indicate the logical deletion of a partial record or data record.	0 (No)	1 (Yes)
13	?CXPR	Partial Record Specifier — Use this flag in conjunction with the lock and unlock flags (?CLCK and ?CUNL) and with the logical delete flag (?CLOG) as described above.	0 (No)	1 (Yes)
14	?CXDB	Data Record Specifier — Use this flag in conjunction with the lock and unlock flags (?CLCK and ?CUNL) and with the logical delete flag (?CLOG) as described above.	0 (No)	1 (Yes)
15	?CXSI	Reserved.		

(concluded)

Table 9-8. Extended Control Word Flags (?PECW)

Bit Offset	Flag	Description	Default	Option
0 - 2	——	Reserved.		
3	?EBGT	Reserved.		
4	?EENT	Reserved.		
5	——	Reserved.		
6	?ERPS	Release Current Position — Use this flag in conjunction with the RELEASE LOCKS/POSITION command only. When set, it specifies that the system will reset the current position above the index.	0 (No)	1 (Yes)
7	?ERLK	Release Locks — Use this flag in conjunction with the RELEASE LOCKS/POSITION command only. When set, it specifies that the system will release (unlock) all locks.	0 (No)	1 (Yes)
8 - 14	——	Reserved.		
15	?EIRBK	Reserved.		

Table 9-9. Processing Packet Returned Status Flags (?PSTU)

Bit Offset	Flag	Description	Default	Option
0 - 5	——	Reserved.		
6	?PSLK	Reserved.		
7	?PPRO	Reserved.		
8	?PPNR	Current Position Nonrecoverable — This flag indicates that the INFOS II system could not recover the current position because you deleted the key on which you were positioned.	0	1
9	?PRLO	Record Length Overflowed Buffer Area — This flag indicates whether or not the actual data record is longer than the length of the buffer you supplied for it. When the system returns from the current READ, WRITE, or REWRITE operation with this bit set, it means that the record did overflow the buffer area. See the description of the ?PLEN field in Table 9-6.	0 No Buffer Overflow	1 Buffer Overflow
10	?PHSI	Index Entry has Subindex Defined — This flag, when set, means that the key to which you gained access for the current request is linked to a subindex.	0 (No link)	1 (Link)
11	?PPLK	Partial Record Locked — This flag, when set, indicates that the partial record you tried to gain access to is locked.	0 (No lock)	1 (Lock)
12	?PDRL	Data Record Locked — This flag, when set, means that the data record you tried to gain access to is locked.	0 (No lock)	1 (Lock)
13	?PDUP	Duplicate Key Encountered — This flag is set when the key to which you gained access is a duplicate. The system returns the duplicate key's occurrence number in the last or only KDP in the key table.	0 (No)	1 (Yes)
14	?PLLD	Local (Partial) Record Logically Deleted — If this flag is set when the system returns from a READ, REWRITE, or DELETE operation, it means the partial record for the accessed key will be marked as logically deleted. Logical deletion does not prevent access to a partial record.	0 (No)	1 (Yes)
15	?PGLD	Global (Database) Record Logically Deleted — If this flag is set when the system returns from a READ, REWRITE or DELETE operation, it means the data record for the accessed key will be marked as logically deleted. Logical deletion does not prevent access to a data record.	0 (No)	1 (Yes)

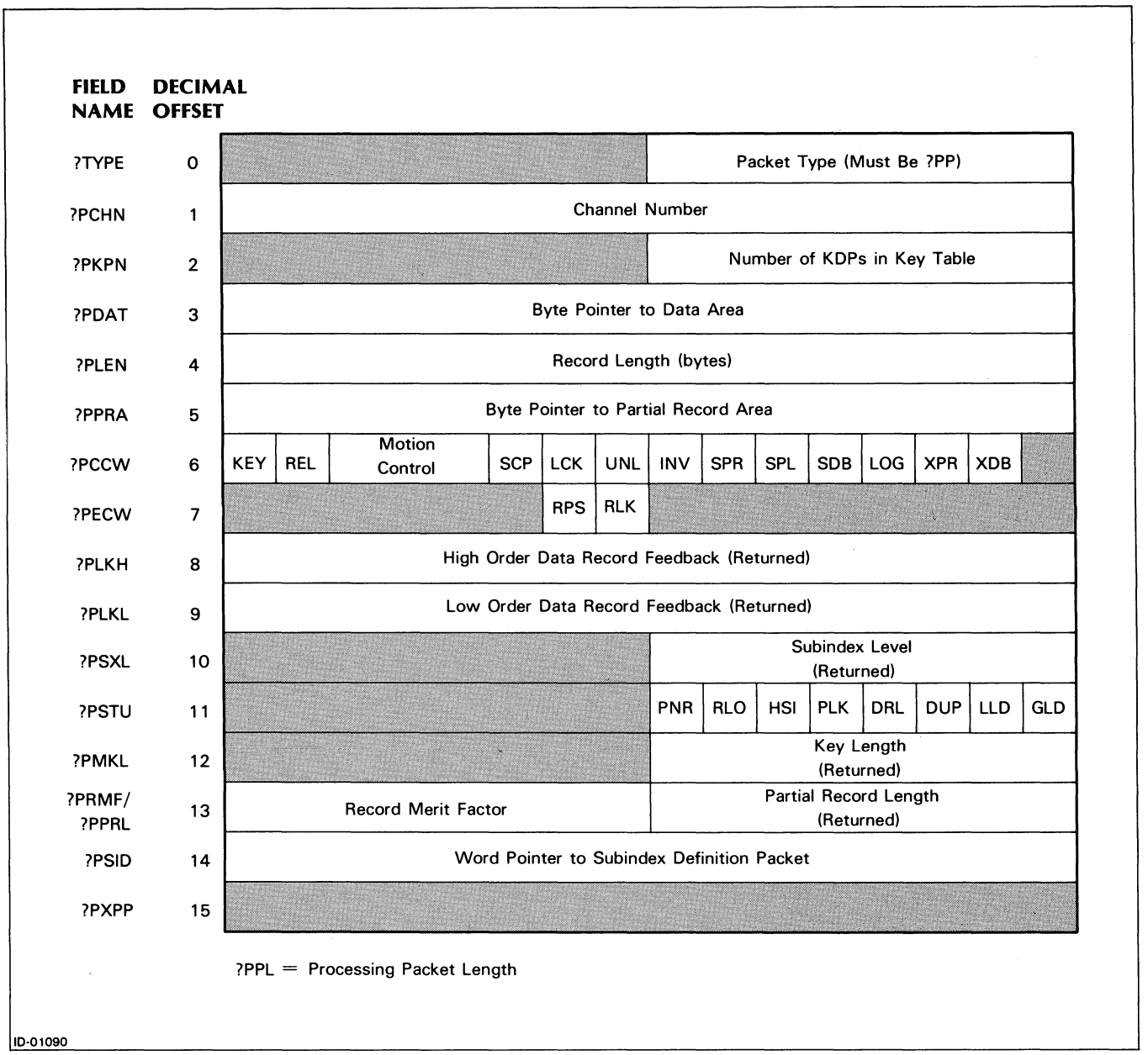


Figure 9-7. Processing Packet (PP)

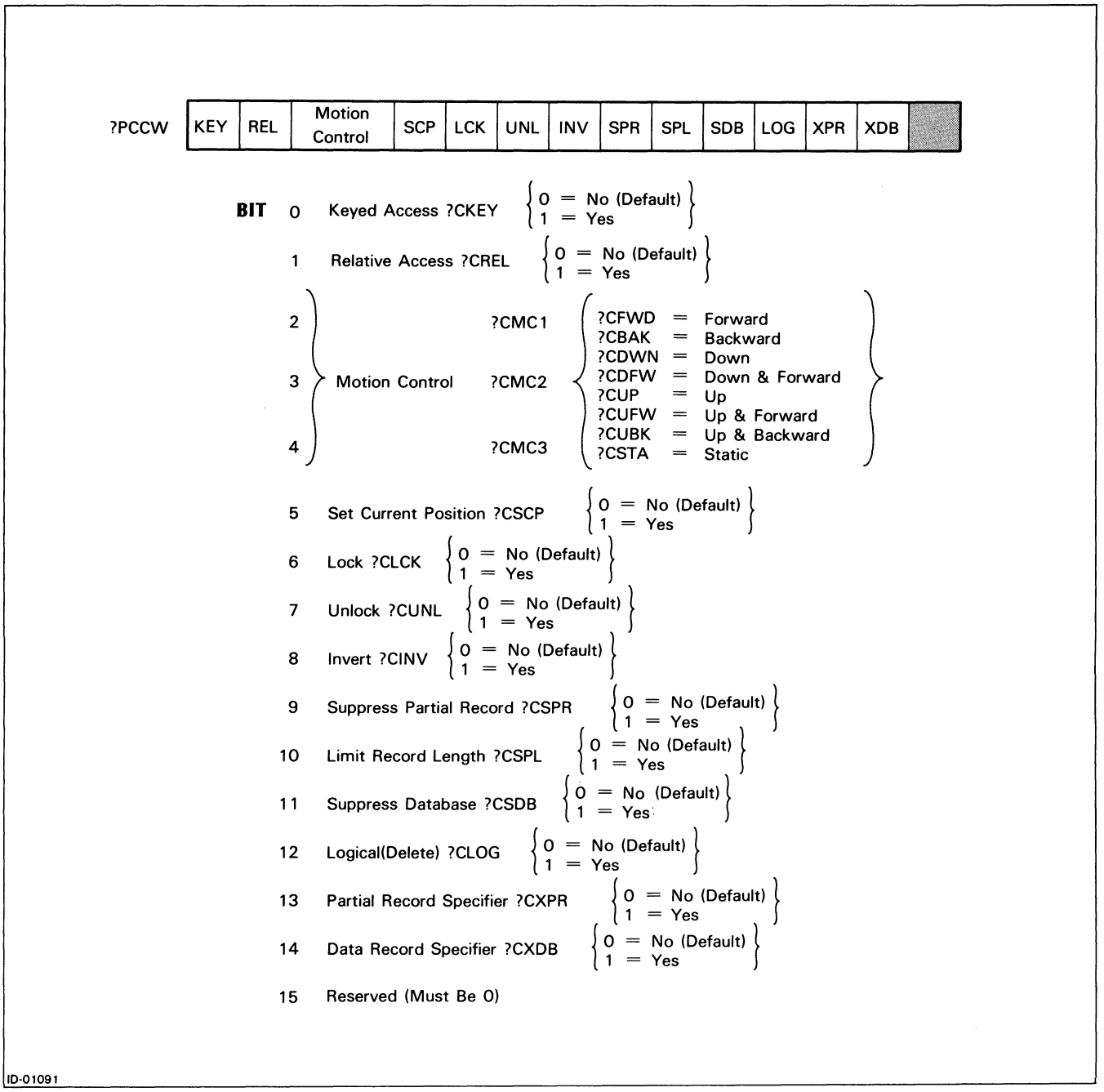


Figure 9-8. Processing Packet Command Control Word (?PCCW)

Table 9-10. Command Control Words Vs. Processing Commands

Processing Commands / Command Control Words	Read	Write	Rewrite	Physical Delete	Logical Delete	Reinstate	Release Locks/Position	Define subindex	Delete subindex	Link subindex	Retrieve Key	Retrieve High Key	Retrieve Status	Retrieve Subindex Def.	Close
?CKEY	1	X	1	X	1	1	NA	1	1	1	1	1	1	1	NA
?CREL	1	Y	1	Y	1	1	NA	1	1	1	1	1	1	1	NA
Motion Control	Y	Y	Y	Y	Y	Y	NA	Y	Y	Y	Y	Y	Y	Y	NA
?CSCP	Y	Y	Y	Y	Y	Y	NA	Y	Y	Y	Y	Y	Y	Y	NA
?CLCK	Y	Y	Y	NA	Y	Y	NA	NA	NA	NA	NA	NA	NA	NA	NA
?CUNL	Y	Y	Y	NA	Y	Y	NA	NA	NA	NA	NA	NA	NA	NA	NA
?CINV	NA	Y	Y	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
?CSPR	Y	Y	Y	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
?CSDB	Y	Y	Y	NA	NA	NA	NA	NA	NA	NA	NA	NA	Y	NA	NA
?CLOG	NA	NA	NA	NA	X	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
?CXPR	Y2	Y2	Y2	NA	Y2	Y2	NA	NA	NA	NA	NA	NA	NA	NA	NA
?CXDB	Y2	Y2	Y2	NA	Y2	Y2	NA	NA	NA	NA	NA	NA	NA	NA	NA

Key:

Y = May be specified.
X = Must be specified.
NA = Not Applicable.
1 = Specify at least one of these.
2 = Specify at least 1 whenever you use lock, unlock, or logical delete.

DG-26351

Key Descriptor Packet

Tables 9-11 and 9-12 contain the parameters of the Key Descriptor Packet (KDP), their standard values, range of values, and, where applicable, the relationships among them. We illustrate the packet in Figure 9-9.

The following are packet identification flags:

?KDP Key Descriptor Packet (revision and type).

?KDPL Key Descriptor Packet Length.

Table 9-11. Key Descriptor Packet Parameters

Offset	Parameter	Definition
0	?TYPE	Packet Type — When you invoke the ?PACKET macro to generate a Key Descriptor Packet, you must specify ?KDP. The value of this field is then ?KDP.
1	?KTYP	Key Type Flags — The key type flags and their default and optional values appear in Table 9-12.
2	?KYLN	Key Length — Specify in bytes the length of the key pointed to in this KDP. Key length must be in the range of 1 to 255 bytes, but it cannot exceed the maximum allowable number of bytes in the subindex that the Processing Packet points to.
3	?KKYP	Key Byte Pointer — Specify the byte address of the key described in the KDP.
4 5	?KDKH ?KDKL	Duplicate Key Occurrence Number: High Order/Low Order — Normally the value of these two fields should be 0. The only time you must specify an occurrence number is when you want a particular occurrence of a duplicate key. To do so, set ?KDUP (see Table 9-12) and specify the unique occurrence number of the desired key in ?KDKL or in ?KDKH and ?KDKL. When the key accessed for any request is a duplicate, the system returns that key's occurrence number to ?KDKH and ?KDKL. When it's not a duplicate, the system fills these fields with zeros.
6 - 7	——	Reserved.

Table 9-12. Key Type Flags (?KTYP)

Offset	Flag	Description	Default	Option
0	?KDUP	Duplicate Key — Use this flag with a WRITE request to determine if the key you are writing is the first occurrence of the key or a duplicate. Set ?KDUP in the last or only KDP in the key table and make the value of ?KDKH and ?KDKL zero. If the key duplicates an existing key, the flag ?PDUP in ?PSTU of the PP will be set when the system returns. You can also use ?KDUP to gain access to a particular occurrence of a duplicate key, other than the primary key, on any processing request (other than a WRITE). Set ?KDUP and specify the occurrence number of the key you want in ?KDKL or in ?KDKH and ?KDKL.	0 (No)	1 (Yes)
1	?KGEN	Generic Key — If you set ?KGEN on any processing request (except WRITE) you specify that the key pointed to in this KDP is a generic search key. You cannot specify a generic key on a WRITE operation.	0 (No)	1 (Yes)
2	?KAPX	Approximate Key — If you set ?KAPX on any processing request (except a WRITE) you specify that the key pointed to in this KDP is an approximate search key. You cannot specify an approximate key on a WRITE.	0 (No)	1 (Yes)
3	?KKLD	Reserved.		
4	?KBOF	Reserved.		
5 - 15	——	Reserved.		

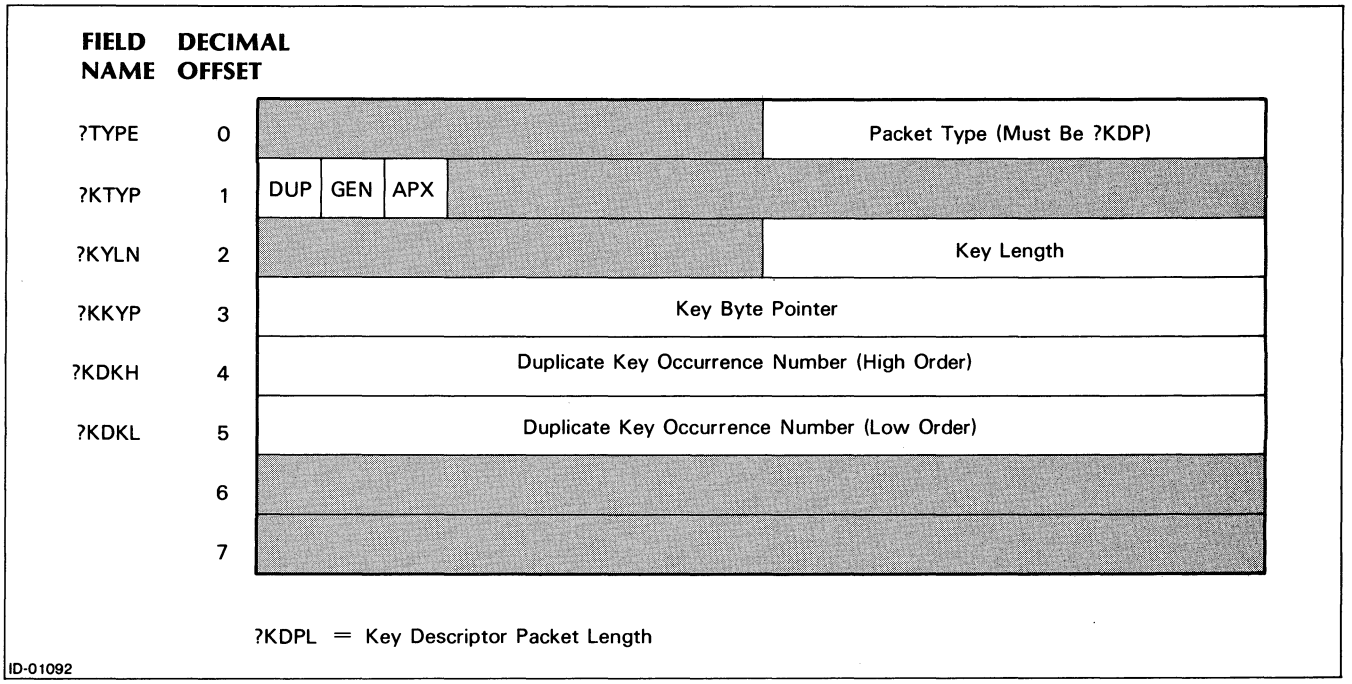


Figure 9-9. Key Descriptor Packet (KDP)

Subindex Definition Packet

Tables 9-13 and 9-14 contain the parameters of the Subindex Definition Packet (SDP), their standard values, range of values, and, where applicable, the relationships between them. We illustrate the packet in Figure 9-10.

The following are packet identification flags:

?SDP Subindex Definition Packet (revision and type).

?SDPL Subindex Definition Packet Length.

Table 9-13. Subindex Definition Packet Parameters

Offset	Parameter	Definition
0	?TYPE	Packet Type — When you invoke the ?PACKET macro to generate a subindex definition packet, you must specify ?SDP. This makes the value of this field ?SDP.
1	?SRNS	Root Node Size — The standard root node size is 6 bytes less than page size of the index file. If you specify a root node size (in bytes), it must be at least large enough to contain three maximum-length index entries, and cannot be greater than 6 bytes less than page size. We describe root nodes and index entries in Chapter 10.
2	?SMKL	Maximum Key Length — The standard maximum key length is 255 bytes. But you can specify from 1 to 255 bytes for this subindex. Specifying a key length prevents you from subsequently writing keys longer than the length you specified.
3	?SRMF/ ?SPRL	Root Node Merit Factor/Partial Record Length — Each of these is 1 byte long. You use the root node merit factor with optimized record distribution. (See Chapter 10 for more information.) The standard byte length for partial records is 0, but you can specify anywhere from 0 to 255 bytes for this subindex. Zero length means the index entries in this subindex will not contain a partial record field.
4	——	Reserved.
5	?SIFL	Subindex Definition Flags — We describe the subindex definition flags and their standard and optional values in Table 9-14.
6	?SRNH	Reserved.
7	?SRNL	Reserved.

Table 9-14. Subindex Definition Flags (?SIFL)

Bit Offset	Flag	Description	Default	Option
0	——	Reserved.		
1	?SNSI	Subindexing Not Allowed — Normally, you can define subindexes under the keys in this subindex. Setting ?SNSI prevents the subsequent definition of subindexes under these keys.	0 Allow Subindexes	1 No Subindexes
2	——	Reserved.		
3	——	Reserved.		
4	?SDUP	Duplicate Keys — The default is that no duplicate keys are allowed in the subindex being defined. If you intend to write duplicate keys to this subindex, set ?SDUP.	0 No Duplicates	1 Allow Duplicates
5-15	——	Reserved.		

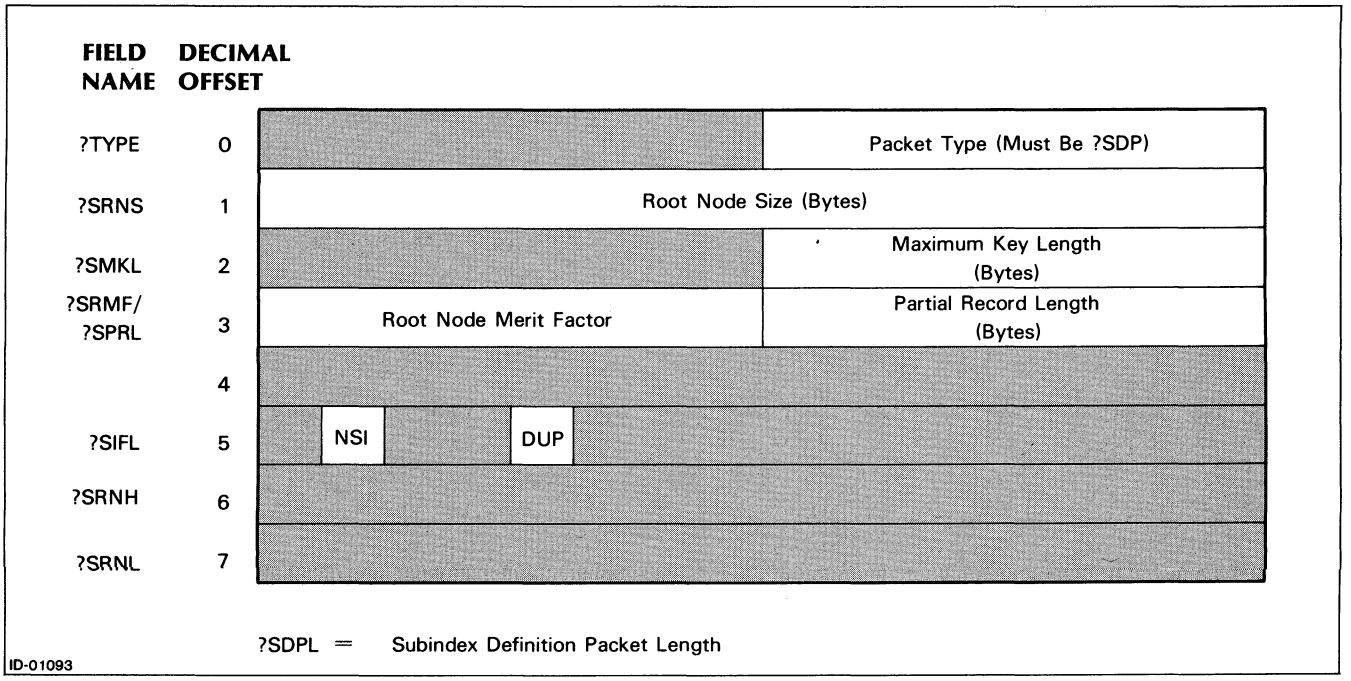


Figure 9-10. Subindex Definition Packet (SDP)

Link Subindex Processing Packet

Tables 9-15 and 9-16 contain the parameters of the Link Subindex Packet (LSP), their standard values, range of values, and where applicable, the relationships between them. We illustrate the packet in Figures 9-11 and 9-12.

The following are packet identification flags:

?LSP Link Subindex Packet (revision and type).

?LSPL Link Subindex Packet Length.

Table 9-15. Link Subindex Packet Parameters

Offset	Parameter	Definition
0	?TYPE	Packet Type — When you invoke the ?PACKET macro to generate a link subindex processing packet you must specify ?LSP. This makes the value of this field ?LSP.
1	?PCHN	Channel Number — Specify the channel number the system assigned when you opened the file and for which you issued the LINK SUBINDEX request.
2	?PSKE	Number of Source Key Descriptor Packets — Specify the number of KDPs in the source key table. You must always specify at least one and you can have as many as 32 for each.
3	?PDKE	Number of Destination Key Descriptor Packets — Specify the number of KDPs in the destination key table. You must always specify at least one and you can have as many as 32 each.
4	?PDKT	Destination Key Table Word Pointer — Specify the word address of the destination key table.
5	——	Reserved.
6 7	?PSCC ?PDCC	Source/Destination Command Control Word — Table 9-16 shows the source and destination command control words and their standard and optional values.
8-15	——	Reserved.

Table 9-16. Source/Destination Command Control Word Flags

Bit Position	Flag	Description	Default	Option
0	?CKEY	Keyed Access — Set ?CKEY to use the keyed access technique to access either or both of the source or destination keys.	0 (No)	1 (Yes)
1	?CREL	Relative Access — Set ?CREL to use the relative access technique to access either or both of the source or destination keys. If you do, you must specify which direction of motion you want in the appropriate motion control field.	0 (No)	1 (Yes)
2 3 4	?CMC1 ?CMC2 ?CMC3	Motion Control — If you set ?CREL for either the source or destination key, or both, you must specify one of the eight directions of motion shown in Figure 9-8. For both the source and destination keys, you must set either ?CKEY or ?CREL. You can set both for either the source key, the destination key, or both. If you do this, the only valid entries in the respective motion control fields will be ?CUP, ?CDWN, and ?CSTA.	None	(See Figure 9-8)
5	?CSCP	Set Current Position — You can set the current position on the source key or on the destination key. If you set it on the source key, that is where the current position will be at the completion of the operation. Similarly, if you set it on the destination key, the current position will be there at the completion of the operation. If you Set Current Position on both the source and destination keys, it will be on the source key while that is the accessed key and then on the destination key while that is the accessed key. Hence, the current position will be on the destination key at the completion of the operation, because that is the last key accessed.	0 (No)	1 (Yes)
6 - 15	——	Reserved.		

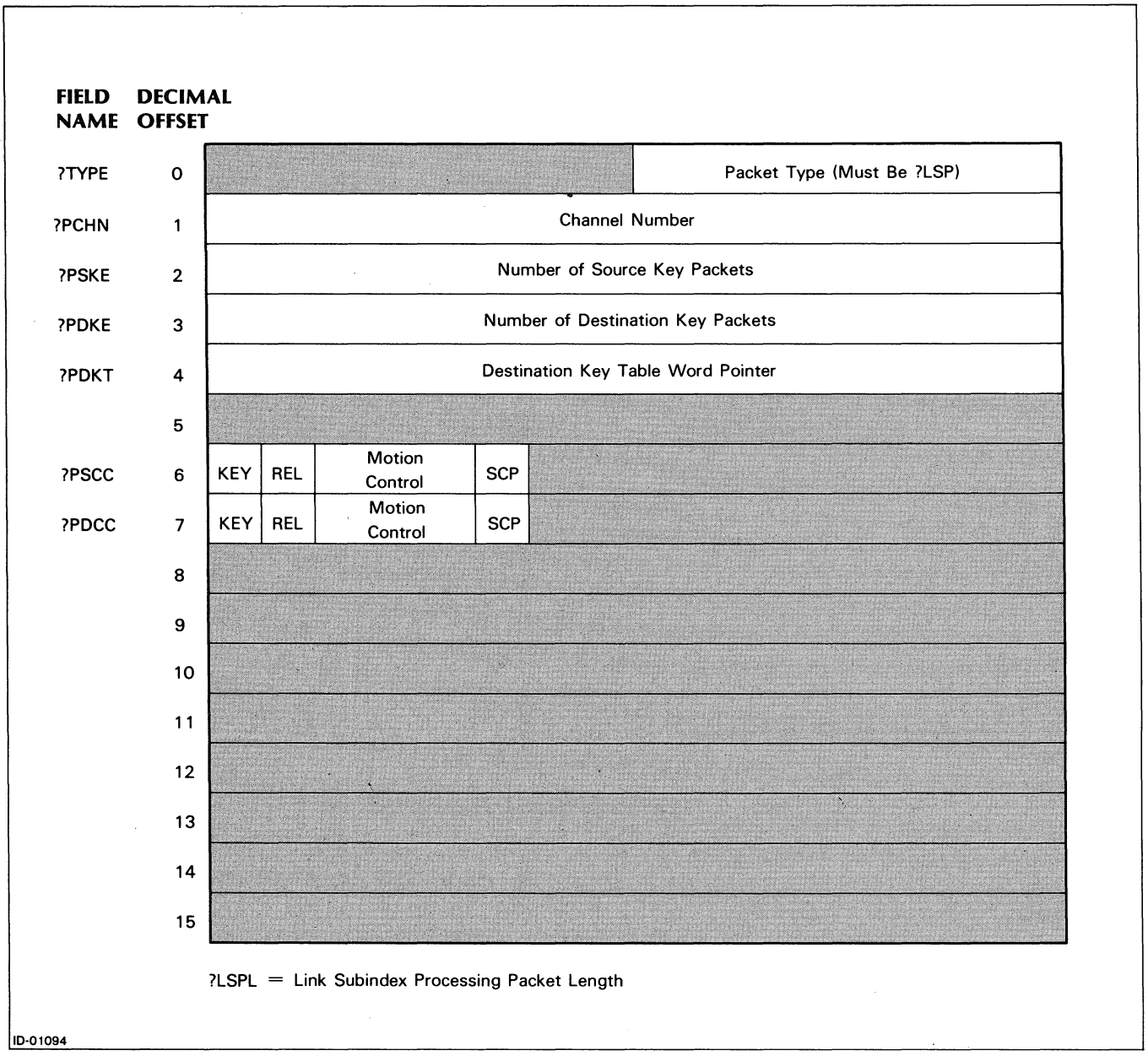


Figure 9-11. Link Subindex Packet (LSP)

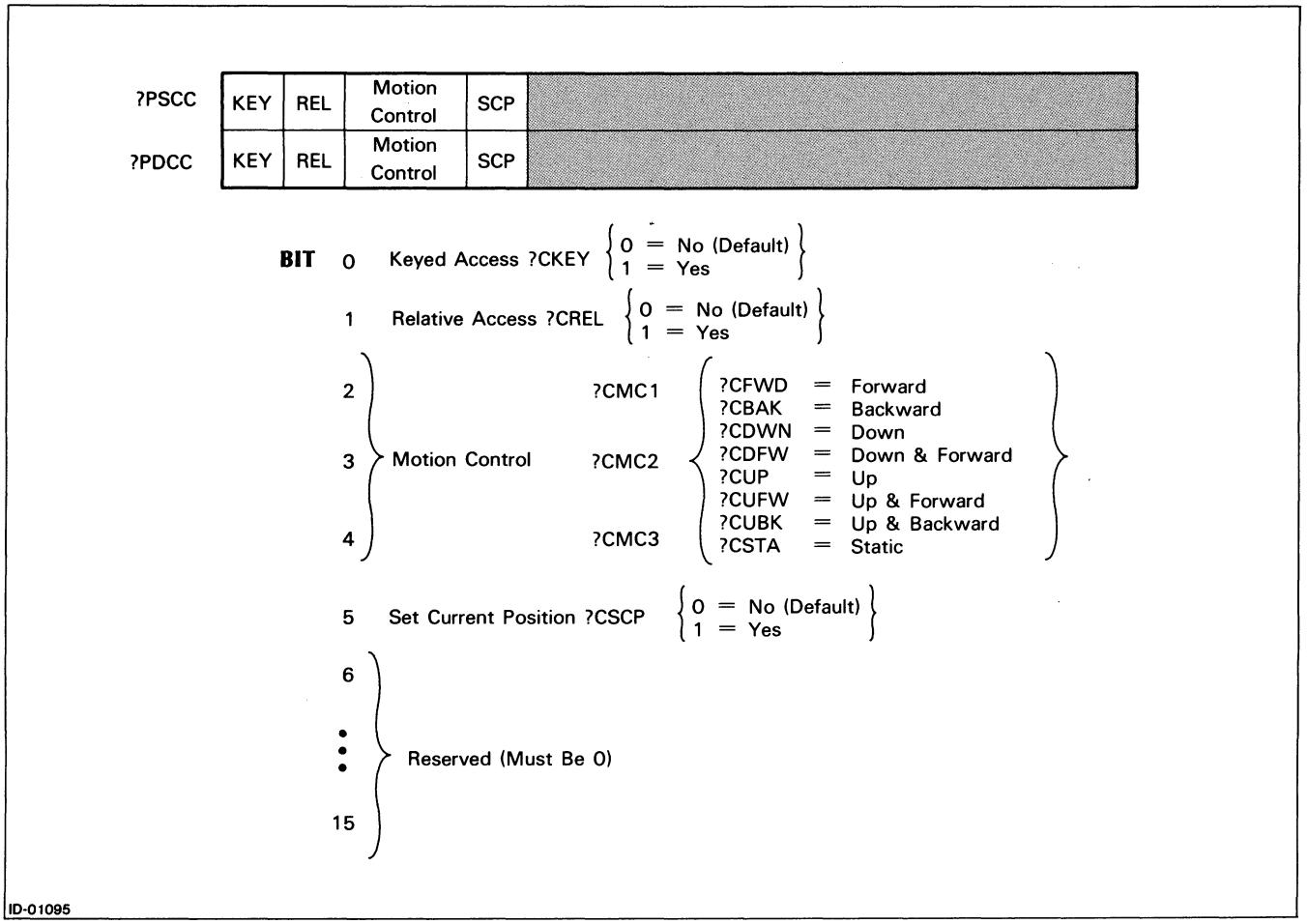


Figure 9-12. Source and Destination Command Control Words (?PSCC and ?PDCC)

Packet Generation

To generate any of the INFOS II packets, issue the macro call, ?PACKET, and specify the type of packet you want to generate. The ?PACKET macro, when expanded, reserves space for a packet of the length required by the type you specify. It will set all parameters to standard values. If you want to specify a nonstandard value for any parameter, you should follow the ?PACKET call with calls to the ?PINIT macro, specifying the parameter's symbolic name and the value you want. You can initialize parameters and set flags in a flag word in any order.

In the following examples, the names and values shown in brackets are optional.

Example of FDP to Create a Standard ISAM File

```
[IFDP]: ?PACKET          ?FDP              ;generates a File Definition Packet
                                           ;of length ?FDPL

        ?PINIT           ?FNAM,IFN+IFN     ;byte pointer to index filename

        ?PINIT           ?FFLG, ?FFCR[+FEIO][+FEDO]
                                           ;sets CREATE mode and, optionally,
                                           ;exclusive index and database use for
                                           ;this open

        ?PINIT           [?FNLR,nn]        ;optional number of locks for this open
                                           ;nn = 00 to 32

        ?PACKET          ?VDP              ;generates a Volume Definition Packet
                                           ;of length ?VDPL for the volume table

IFN:    .TXT             /indexfilename/   ;specify the name of file to be created
```

Example of FDP to Open an INFOS II File

```
[IFDP]: ?PACKET          ?FDP              ;generates an FDP of length ?FDPL
        ?PINIT           ?FNAM,IFN+IFN     ;byte pointer to filename
        ?PINIT          [?FNLR,5]          ;five locks for this open
        ?PACKET          ?VDP              ;generates a VDP of length
                                           ;?VDPL for the volume table

IFN:    .TXT             /indexfilename/   ;specify the name of file to be opened.
```

Example of a Processing Packet for a Keyed Read

```
        ?PACKET         ?PP             ;generates a Processing Packet
                                        ;of length ?PPL
        ?PINIT          ?PKPN,2         ;two keys in the key table
        ?PINIT          ?PDAT,DAT+DAT   ;byte pointer to data area
        ?PINIT          ?PCCW,?CKEY     ;specifies keyed access
        ?PACKET         ?KDP            ;generates first KDP (length
                                        ;?KDPL) in key table
        ?PINIT          ?KYLN,6         ;first key is 6 bytes long
        ?PINIT          ?KKYP,K1+K1     ;byte pointer to first key
        ?PACKET         ?KDP            ;generates second KDP (length
                                        ;?KDPL) in key table
        ?PINIT          ?KYLN,6         ;second key is 6 bytes long
        ?PINIT          ?KKYP,K2+K2     ;byte pointer to second key

DAT:    .BLK            40.             ;80-byte data area
K1:     .TXT            /SYSTEM/        ;first key
K2:     .TXT            /MANUAL/        ;second key
```

NOTE: At execution time, you load ?FCHN from the FDP and store the assigned channel number in ?PCHN of the Processing Packet.

Call Formats

You can execute an INFOS II function in two ways — with a command request or with a macro call. For a simple example, let's use the READ request. The command to execute a READ is ?CRED and the macro call is ?ERED. In both cases, you must generate a processing packet, specifying the required and optional parameters for the request. The processing packet must be followed immediately by a key table with at least one KDP. We include the call formats in Table 9-17. Here, PP represents the word address of your processing packet. Note that the packet address must be in AC2.

Table 9-17. Call Formats

Command (in AC0)	Macro	Macro
LEF 0, ?CRED	ELEF 2, PP	?ERED PP
EJSR INF0	?ERED	(error return)
(error return)	(error return)	(normal return)
(normal return)	(normal return)	

In all cases, if the INFOS II system takes the error return, it will return the error code in AC0. Upon expansion, the ?ERED macro is replaced with the following:

```
JSR     @?INFS
?CRED
```

We list the INFOS II error codes in Appendix A. If you get an error, you can issue the AOS call ?ERMSG to determine how to handle the error. The call ?ERMSG is described in your AOS Programmer's Manual.

Table 9-18 describes the INFOS II commands and macro calls.

Table 9-18. INFOS II Commands and Macro Calls

Command	Macro	Description
?COPN	?EOPN	Open an INFOS II file.
?CCLS	?ECLS	Close an INFOS II file.
?CRIT	?ERIT	Write.
?CRED	?ERED	Read.
?CRER	?ERER	Rewrite.
?CDLR	?EDLR	Delete.
?CDFN	?EDFN	Define subindex.
?CDLI	?EDLI	Delete subindex.
?CLNK	?ELNK	Link subindex.
?CRKY	?ERKY	Retrieve key.
?CRTH	?ERTH	Retrieve high key.
?CRTS	?ERTS	Retrieve status.
?CRSD	?ERSD	Retrieve subindex definition.
?CRIN	?ERIN	Reinstate.
?CRLP	?ERLP	Release position and/or locks.

End of Chapter

Chapter 10
Index and Database File Structures

In this chapter we describe the internal structure of INFOS II files. We discuss the physical properties of indexes and databases, and how to use these properties to optimize the design of your INFOS II file.

An INFOS II file is actually two independently defined files. At file creation, an index file and a database file link to form a single processible unit. This unit is the INFOS II file. Occasionally, you might link two or more index files to one particular database, but each unique index/database unit is considered a single INFOS II file.

Indexes and Subindexes

Normally, you'll want to design an index structure that can hold the maximum number of keys you expect to use. You will also want to access the information in your INFOS file as quickly as possible. These are reasonable goals to keep in mind, but you should remember that the application for which you are designing an index file ultimately determines how many subindexes you need and what their physical properties should be. To design the best possible INFOS II file for your needs, you must become familiar with the structure of a main index and subindexes.

Index Entries

Index entries contain your keys. There are two basic types of index entries: system-generated entries, and user-generated entries. We show these two index entry formats in Figure 10-1. We describe system-generated entries in "The Growth of Tree Levels" section, later in this chapter.

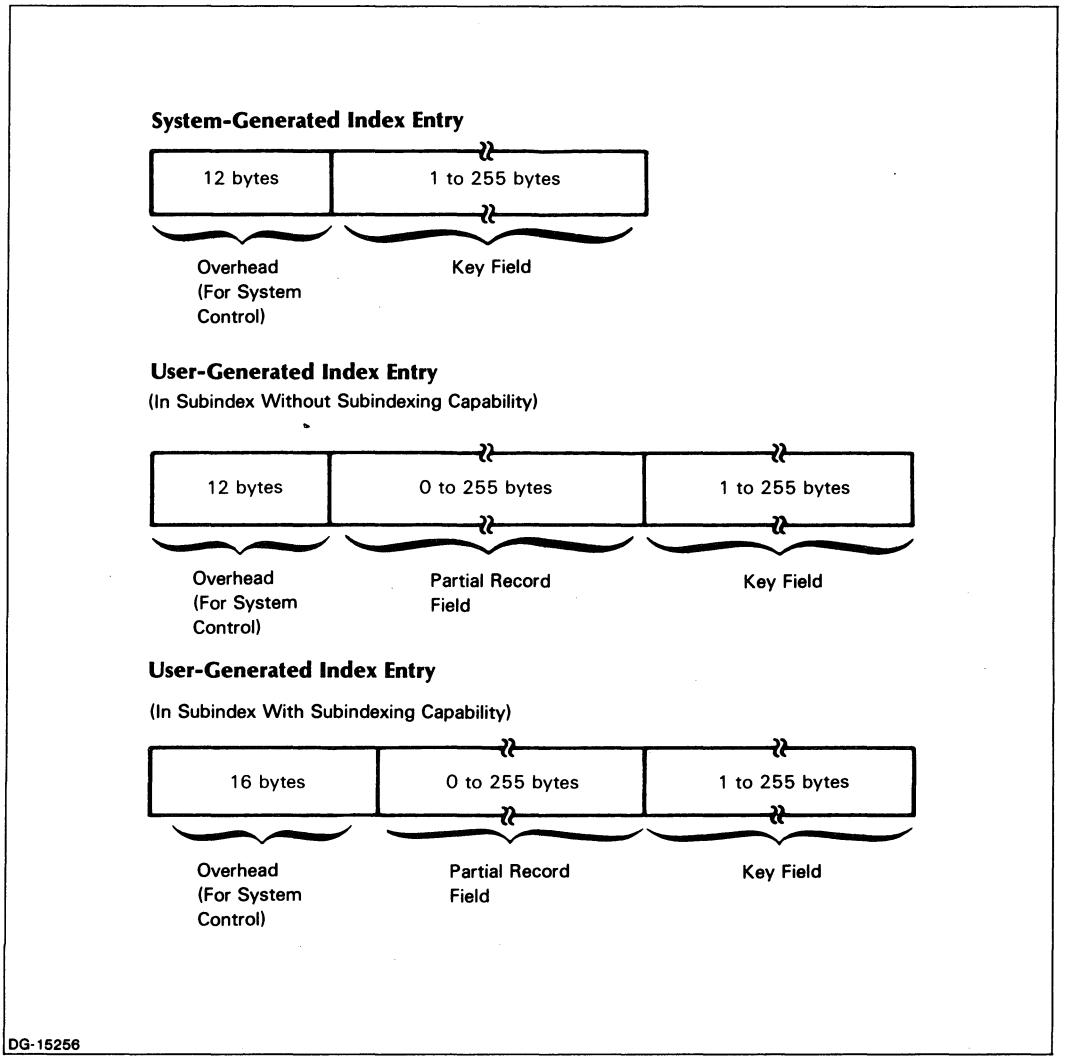


Figure 10-1. Index Entry Formats

Nodes

Index entries are contained in *nodes*. Nodes are the "building blocks" of a subindex. The INFOS II system allocates space for a subindex in node-sized blocks. As the number of entries in a subindex increases, so does the number of nodes.

The nodes reside in different levels of a subindex *tree*. (The subindex actually has an inverted tree structure, since the root is at the top and the branches below.) Any subindex can have up to 15 *tree levels*, but you will probably never have more than 3 or 4. The INFOS II system attempts to have as few tree levels as possible, to reduce the number of times it must gain access to the disk.

The system can create three different types of nodes: the *root* node, *intermediate level* nodes, and *level 0* nodes. We picture these node types in the three-tree-level subindex in Figure 10-2.

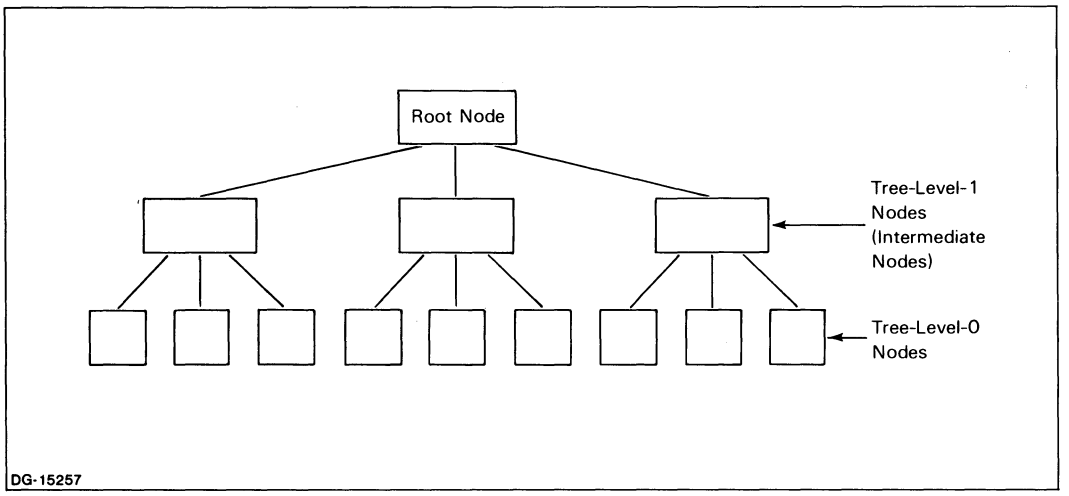


Figure 10-2. Three-Tree-Level Subindex

Notice that the tree levels are numbered sequentially from 0 at the bottom to the root node at the top. The nodes at the levels between 0 and the root are the intermediate level nodes.

Root Nodes

The root node is the first node that the system builds in a main index or subindex. While it is the only node, it is also the level 0 node. You specify the size of your index root node with your response to the ***ROOT NODE SIZE*** prompt in ICREATE. The smallest node size you can specify is one large enough to contain three maximum size entries. The largest is the default that appears in the ICREATE prompt (your index page size minus 6 bytes).

The Growth of Tree Levels

When you write entries into your subindex, the system places them in the root node. As you continue to write entries, the root node grows to its maximum size. Once the node becomes full, the INFOS II system splits the root node's entries and places them into two nodes. The root node contains a pointer to each new node. The two new nodes become tree level 0.

If you continue writing entries into this same subindex, the system places them in more nodes at tree level 0. For each new level 0 node that it creates, the INFOS II system writes an entry in the root node that points to it. Eventually, the root node will once again grow to its maximum size, and will split its entries in two again, to form two nodes at tree level 1. The nodes at this level are intermediate level nodes, between level 0 and the root node. Entries in the root node point to the nodes at tree level 1, whose entries in turn point to the nodes at level 0.

Example of Tree Building

In this example, we show how an index grows from one level (root node only) to three levels. Figure 10-3 pictures the root node with its keys A, D, and I.

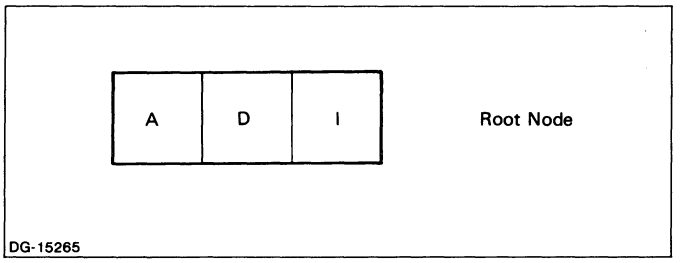


Figure 10-3. Root Node

We attempt, at this point, to enter another key, G. However, the root node is full. The INFOS II system locates the position where it would normally insert the key (in alphabetical order) and splits the root node there. It places the new nodes below the root, in index level 0, and places entries in the root node that point to the subordinate nodes. The pointers consist of the high key (largest binary value) in the subordinate nodes. Figure 10-4 illustrates this step.

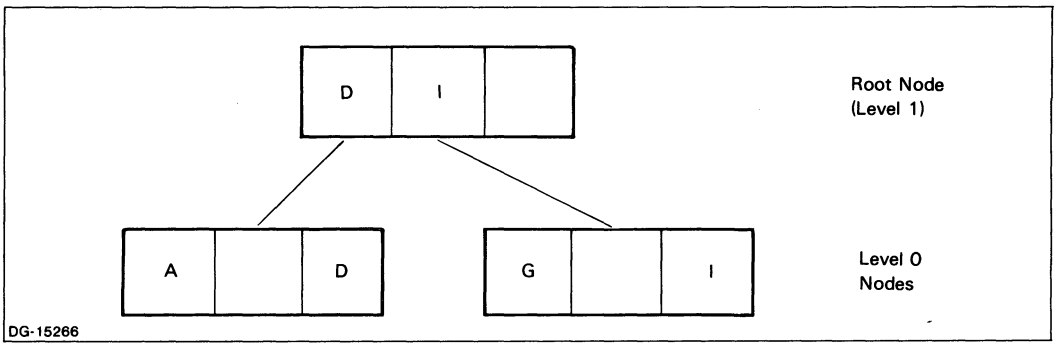


Figure 10-4. Two-level Node Structure

Now, assume that we want to add three more keys: C, B, and H. The C key will fit into the left-hand level 0 node. When we add the B key, there isn't any room in the left-hand node. The system will need room to place the B key in the correct place. It gets room by performing a shift right; it shifts the D key into the right-hand level 0 node. The system then inserts the B key in the left-hand node and updates the affected root node pointers. Figure 10-5 illustrates this.

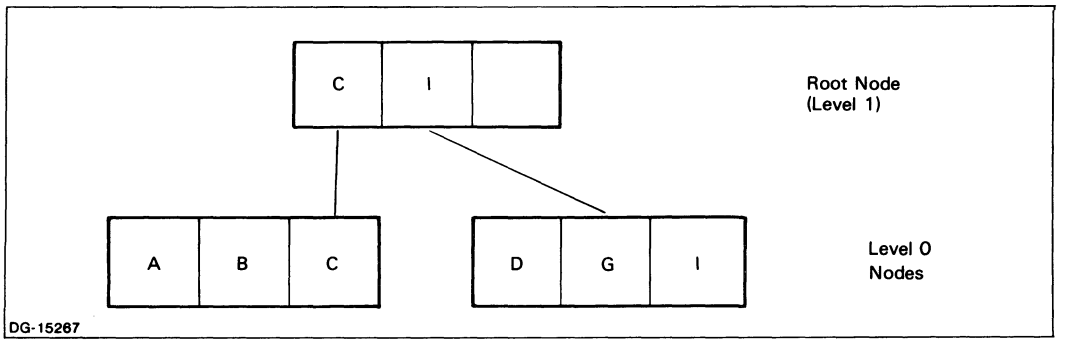


Figure 10-5. Extended Two-level Node Structure – I

When we enter the H key, the system attempts to place the key alphabetically into the second level 0 node. Since the node is full, the system tries to shift right, as described above. However, no node exists in this position, so the system tries to shift left. The left-hand node is full, so the system creates a new left-hand node for the present right-hand node. It never creates right-hand nodes. Figure 10-6 illustrates this root node structure.

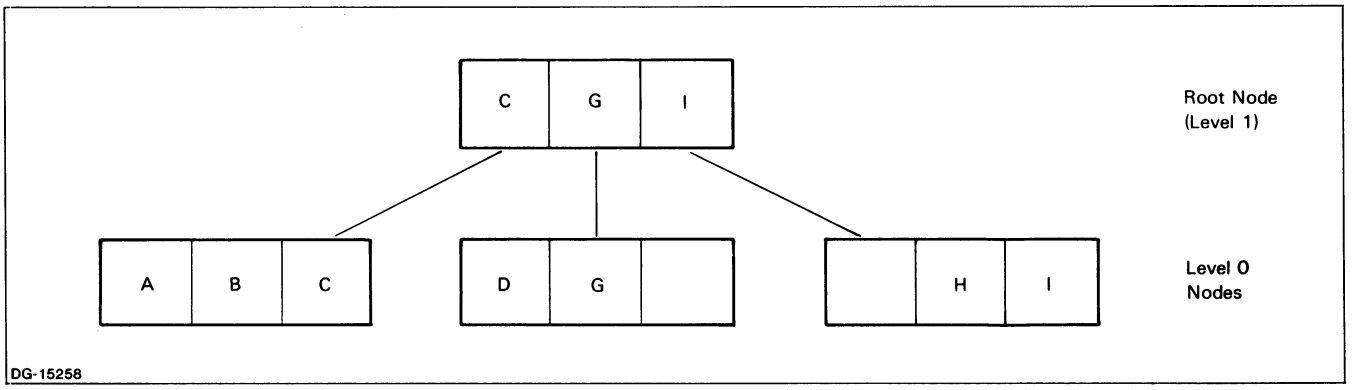


Figure 10-6. Extended Two-level Node Structure – II

When all the level 0 nodes are full, the INFOS II system creates intermediate level nodes.

In summary, the INFOS II system follows this procedure to fill nodes:

1. If there is room in the node to which the entry belongs, the system places it there.
2. If there is not room in the appropriate node, the system tries to shift entries to the next node to the right (the node alphabetically just after the full node).
3. If it cannot do this, then the system tries to shift entries to the node on the left-hand side (the node alphabetically just before the full node).
4. If the system still cannot find space for the entry, it creates a new left-hand node to the full node.
5. The high key of the new left-hand node must fit alphabetically into the node at the level above it. If there is no room in that node, the system follows the above steps to find room. Eventually, the system finds room for all the new node entries created in this manner.

The intermediate level nodes become important when we discuss optimized record distribution, later in this chapter.

Ordering Your Keys

You may conclude from this example that the order in which you initially enter your keys can affect the amount of disk space that you use. For example, if you are building a large file, you can save a significant amount of disk space by writing the keys in ascending or descending order, rather than randomly. Your database records will be sequentially ordered on the disk, which will improve your sequential accessing speed. Entering your keys in descending order gives you optimum performance, because it minimizes the average number of intermediate node and root node changes that the system must perform each time you write a key to the index.

However, you shouldn't be overly concerned with writing your keys in order. Your index nodes will probably contain many more entries than those we used in our example, thus making the wasted space much less, relative to the size of the nodes. Entering your keys in ascending or descending order will give you a compact file, but this won't be to your advantage if you intend to do a lot of updating on the file, unless your rewrites are the same size as your original entries. Just bear in mind that if you do have the option of entering keys sequentially, it could be beneficial, depending on your application.

Pages and Node Size

Nodes are contained in *pages,* the unit of transfer between the file and the INFOS II system. Remember from Chapter 2 that when you specify parameter values for an index, you can choose between a 2048-byte page and a 4096-byte page. Any single node, other than the root node, is 6 bytes less than page size. (Root nodes are whatever length you specified when you created/defined the index.) Thus, given one node per page, the number of index entries per node determines how much data is transferred for a single file access.

The Branching Factor

The number of index entries per node is called the node's *branching factor.* The minimum branching factor is 3.

When you define a main index or any particular subindex, you specify certain parameters either explicitly or by default: maximum key length, partial record length, and whether or not the subindex can be linked to lower level subindexes. Based on these specifications, the INFOS II system requires that you allocate a root node that is at least large enough to contain three maximum length index entries. Subordinate nodes default to the page size you specified minus 6 bytes.

The INDEXCALC utility (see Chapter 5) will calculate branching factors for you. If, however, you want to calculate them yourself, use the formulas below. The formulas assume that each page has one node, and they also account for page and node overhead space requirements. All numbers and constants are in decimal and reflect the required number of bytes. We use the following abbreviations in these formulas:

BF Branching Factor.

MKL Maximum Key Length, rounded up to the nearest even number of bytes.

PRL Partial Record Length, rounded up to the nearest number of bytes.

Branching Factors for an ISAM Index

Root Node (if not the only node):

BF = (node size − 40) / (MKL + 14)

Root Node (if the only node):

BF = (node size − 40) / (MKL + PRL + 14)

Intermediate Level Nodes:

BF = (node Size − 22) / (MKL + 14)

Level 0 Node:

BF = (node size − 22) / (MKL + PRL + 14)

Branching Factors for a DBAM Index

Root Node (if not the only node):

BF = (node size − 40) / (MKL + 14)

Root Node (if the only node, and without subindexing):

BF = (node size − 40) / (MKL + PRL + 14)

Root Node (if the only node and with subindexing):

BF = (node size − 40) / (MKL + PRL + 18)

Intermediate Level Nodes:

BF = (node size − 22) / (MKL + 14)

Level 0 Node (without subindexing):

BF = (node size − 22) / (MKL + PRL + 14)

Level 0 Node (with subindexing):

BF = (node size − 22) / (MKL + PRL + 18)

Note that the divisor in each formula reflects the index entry size and includes 2 bytes of overhead per index entry. If the sum of its components is not an even number, round up to an even number, because index entries begin on word boundaries, not byte boundaries.

Also note that the dividend in each formula is page size minus overhead. This is because you calculate branching factors on a node-per-page basis.

Tree Levels and Access Speed

Since user-supplied index entries are always stored in tree level 0, the number of tree levels per subindex can be an important factor in access speed. Given the preceding formulas, you can calculate how many tree levels you need for a given number of user-supplied index entries. You can also calculate how much disk space the system needs to store the subindex. In general, a keyed access to an index entry in any subindex requires one disk access per subindex tree level (assuming that there is one node per page).

Suppose, for example, that the most frequently accessed subindex will have 5000 user entries. These entries will have a maximum key length of 26 bytes and a 30-byte partial record; and the subindex has the subindexing capability. With the 2048-byte page, you will need 185 nodes at tree level 0.

$$\frac{2048 - 28}{26 + 30 + 18} = 27 \text{ entries per level 0 node}$$

5000/27 = 185 level 0 nodes

For a two-tree-level subindex, the root node needs a branching factor of 185. However, the root node's branching factor is actually 50.

$$\frac{2048 - 46}{26 + 14} = 50$$

Thus, the system needs at least three subindex tree levels to store 5000 user-supplied index entries.

If we use the larger page size, however, the system needs only two subindex tree levels: a root node and 92 level 0 nodes.

$$\frac{4096 - 28}{26 + 30 + 18} = 54 \text{ entries per level 0 node}$$

$5000/54 = 92$ (required number of level 0 nodes)

$$\frac{4096 - 46}{26 + 14} = 101 \text{ (maximum number of root node entries)}$$

With the larger page size, each node occupies 8 disk blocks. Thus, the system requires a minimum of 744 blocks of disk storage (or 93 pages) for this subindex.

(1 root node + 92 level 0 nodes) * 8 = 744

Selector Keys and Space Management

If you have a relatively small number of selector keys, say six, and these keys aren't too large, say 10 bytes, they will probably fit in a subindex that consists entirely of a root node. When you specify the parameters of any main index or subindex, you can specify a root node size appropriate for the selector subindex. Then, if you select the space management feature, the INFOS II system automatically collects on a single page as many subindexes as possible that consist only of a root node. Remember, however, that the root node must be at least large enough to contain three maximum-length index entries.

Space management also enables you to reuse space that is freed when you delete all the keys in a node.

Index Nodes and Optimized Record Distribution

If you have a large installation with different types of storage devices having different access speeds, you might want to place your most frequently used index files, root nodes, or volumes on faster devices. Optimized record distribution allows you to do this.

To use optimized record distribution for an index, you specify an index merit factor when the ICREATE utility gives you these prompts:

```
OPTIMIZE RECORD DISTRIBUTION? (Y OR [N]):  Y↓
ROOT NODE MERIT FACTOR: [1]:
```

Merit Factors

The root node merit factor is a number from 1 to 32. Assign the highest numbers to those nodes that you will access most frequently. Based on this merit factor, you should then assign a volume merit factor to an index volume that is equal to, or lower than, the root node merit factor. You assign the index volume merit factor in the DEFINE INDEX VOLUME(S) section of ICREATE when ICREATE issues the following prompt:

```
VOLUME MERIT FACTOR: [1]:
```

When the INFOS II system creates nodes, it assigns different merit factors to them. Your root node will always have the merit factor that you define. However, you have no way of selecting the value of a lower level node.

The INFOS II system assigns merit factors to lower level nodes as follows. When the system creates a new index node, it checks that node's tree level. The merit factor of that node becomes one plus the tree level of the index node at creation time. Therefore, as your index file grows, the INFOS II system assigns higher merit factors to the higher tree levels. This allows you to place those higher tree-level and more frequently accessed nodes on faster devices.

The INFOS II system places your index nodes on the first volume that has room for the node and that has a merit factor equal to or lower than the root node merit factor. If no appropriate volume has room, the node will go on the volume with the lowest merit factor that has room. You (or the system manager) must then set up a link structure so that your different volumes will reside on different access devices. To place a volume on an access device, you must take the following into account:

- You can assign volumes only to logical disks.
- Logical disks appear as directory names subordinate to the root directory in AOS.

Then follow these procedures:

1. Create the INFOS II file.
2. Move a volume into the logical disk directory for each volume you want to place there. Use only one disk per volume.
3. Delete the volume in the INFOS II system directory. In its place, create a link to the file assigned to it in the other disk.

The root node should have the highest merit factor, and therefore be put on the fastest access device, because the INFOS II system must gain access to the root node whenever it performs a keyed access. Intermediate nodes have lower merit factors because the system must gain access to them whenever a key resides below them on the same tree branch. Level 0 nodes have the lowest merit factors because the system must gain access to them only when it needs a key that they contain.

If you spread out your volume merit factors, you can group bunches of nodes with similar node merit factors on the same volume. In addition, you should consider the likely number of nodes and node levels in your index file before you select your volume merit factors. Use the equations given earlier in this chapter to help you. Not allowing for a sufficient range of merit factors can mean that you will have to allocate a large number of index nodes to one particular volume. Defining more volumes helps prevent this. In addition, it enables you to define finer distinctions between your volume merit factors and, as a result, between your index nodes.

The more volumes you have with different volume merit factors, the more types of access devices you can specify. However, if you specify more merit factors than there are access devices on the system, the system manager will obviously have to place two or more of those volumes on the same access device. You might wish to do this if you are planning to add different access devices later, and you do not want to rebuild your files.

When Not to Use Optimized Record Distribution

If you want to place your entire index on a fast (or slow) access device, you do not need optimized record distribution. Merely have your system manager link all your index volumes to the appropriate access device. Similarly, if you only have one access device, you don't need optimized record distribution. Just create the INFOS II file on that access device.

Do not use optimized record distribution unless you define more than one volume. The INFOS II system places your index nodes on the first volume it finds with both available space and a volume merit factor equal to or lower than the index node's merit factor. If the system cannot satisfy these criteria, it will place the node on the lowest value volume having a merit factor value greater than the one specified. If you have only one volume, the system will have no choice where to put your indexes.

Databases

Like index files, database files also use pages and nodes as building blocks. The INFOS II system writes one record per node and as many nodes as will fit on a page. There are 4 bytes of overhead per page and 4 more bytes of overhead per node. Thus, the system cannot write a record with a size greater than the page size minus 8 bytes.

The smallest amount of space the INFOS II system will allocate for a database record is 4 bytes. That is, if your data record length is less than or equal to 3 bytes, the system will allocate 4 bytes to store it. If you subsequently rewrite the record, increasing its length, the system might have to relocate the record. If this happens, the system will use the first 4 bytes of the old location as a pointer, storing the record's new address there. If you enabled space management for the database, the system will place the rest of the space used by the record (in the old location) into the available-space table. The system can then reuse this space for some other purpose.

Sequential Processing

If you use sequential processing on the records associated with a given subindex, we recommend that you sort the keys in descending order prior to writing the records. This minimizes the number of times the system must access the disk while building the file, and during subsequent sequential processing.

Space Management

If you plan to update your file frequently, we recommend that you use space management. With this feature, the system searches through the file for unused space where it can insert a new record. (Two other INFOS II features that save disk space are key compression and data record compression. See Chapter 2 for more information on these features.)

Space management begins to work whenever you open a file and want to insert more information. For example, suppose you want to write some data records. The space management option begins searching for file space on volume 0, the volume described by the first Volume Definition Packet at creation time. (See Chapter 9 for information on the Volume Definition Packet.) If it does not find adequate space on volume 0, it searches the next volume, and the next, and so on until it finds available space. Note that it only does this when you first reopen the file, and not for every write request you issue. After the first opening of the file, INFOS writes to the current volume.

The system will not reuse any available space from deleted records if you do not have space management enabled. It will simply add new records to the last page of the file. However, it will attempt to reinsert modified records in the same area it took them from.

Blocking Factor

You can calculate a page's blocking factor using this formula:

$$BF = \frac{\text{Page Size} - 4}{\text{Ave. Record Size} + 4}$$

For example, using this formula with 80-byte records you will get 24 records on a 2048-byte page:

$$\frac{2048 - 4}{80 + 4} = 24$$

Databases and Optimized Record Distribution

Optimized record distribution for your database works in a manner similar to how it worked for your index. However, the configurations are simpler. Also, with database optimized record distribution, you can specify a record merit factor for each record. You cannot do this with the index file version. If you do not specify a record merit factor, the system will assign your record the lowest merit factor.

You specify Optimized Record Distribution for a database in the ICREATE dialog. The system will prompt for it as follows:

In the DEFINE DATABASE FILE section:

```
OPTIMIZE RECORD DISTRIBUTION (Y OR [N]): Y↓
```

In the DEFINE DATABASE VOLUME(S) section:

```
VOLUME MERIT FACTOR: [1]:
```

Since you can have only 32 volumes in a database file, your volume merit factor must be between 1 and 32. You must define your volume merit factors in nonincreasing order. Two volumes can have the same merit factor.

When you write a record in your database, you can specify a record merit factor for the data record. The INFOS II system then places your record on the first volume that has both available space and a merit factor equal to or less than the record's merit factor. If the system cannot find a volume that satisfies these criteria, it places your record on the volume with the lowest value merit factor that is higher than the one you specified.

Your system manager has to place your volumes on different access devices using a link structure. See the previous section on index nodes and optimized record distribution for details.

End of Chapter

Chapter 11
Distributed INFOS II Processing

Distributed INFOS II processing enables you to gain access to INFOS II files on remote computers in a Data General X.25 network. The Remote INFOS Agent (RIA) manages remote INFOS II requests in such a network. This chapter describes distributed INFOS II processing and explains how RIA functions in a distributed INFOS II network.

In this chapter, we assume that you are familiar with the XODIAC™ Network Management System. Because of this, we use some XODIAC terminology. We will give brief definitions when necessary, but for more information, see the *XODIAC™ Network Management System Guide for Managers and Operators.*

Components of a Distributed INFOS II System

The minimum network software components you need for a distributed INFOS II network are RIA, INFOS II, and the Data General X.25 product, which includes X.25, NETOP, NETGEN, and NETERMES. You need the RIA, X.25, and NETOP processes on all host machines to or from which you want to make remote INFOS II requests. If you plan to use INFOS II utilities on remote hosts, you also need the XODIAC Virtual Terminal Agent (VTA). We explain more about utilities in the "Using RIA with Utilities and Related Software Products" section of this chapter.

RIA

The RIA process transports all INFOS II requests to and from different hosts in a distributed INFOS II network. It allows you to gain access to INFOS II files on remote hosts without changing your application programs. It allows you to open files on one remote host, or several remote hosts simultaneously. Furthermore, RIA can transport requests to or from hosts running AOS and the AOS INFOS II system, or AOS/VS and the AOS/VS INFOS II system.

The version of NETOP that supports the RIA process has an interprocess communication port (IPC) in :PER for use with RIA. This IPC port receives messages in the form of operator commands intended for RIA. We include a list of commands that you can issue to this IPC port at the end of this chapter.

If your local host doesn't have INFOS II files on it, you need not run the INFOS II process there. As long as the RIA, X.25, and NETOP processes are running on your local host, you can share the INFOS II files of other computers in the network, which also have these processes. Table 11-1 shows the different combinations of hosts and and INFOS II files you can have in a network.

Table 11-1. Combinations of Hosts and INFOS II Files

Local Host	Remote Host			
	AOS INFOS II RIA	**AOS/VS INFOS II RIA**	**AOS (INFOS II not running) RIA**	**AOS/VS (INFOS II not running) RIA**
AOS INFOS II RIA	⟷	⟷	←	←
AOS/VS INFOS II RIA	⟷	⟷	←	←
AOS (INFOS II not running) RIA	→	→	xxxxx	xxxxx
AOS/VS (INFOS II not running) RIA	→	→	xxxxx	xxxxx

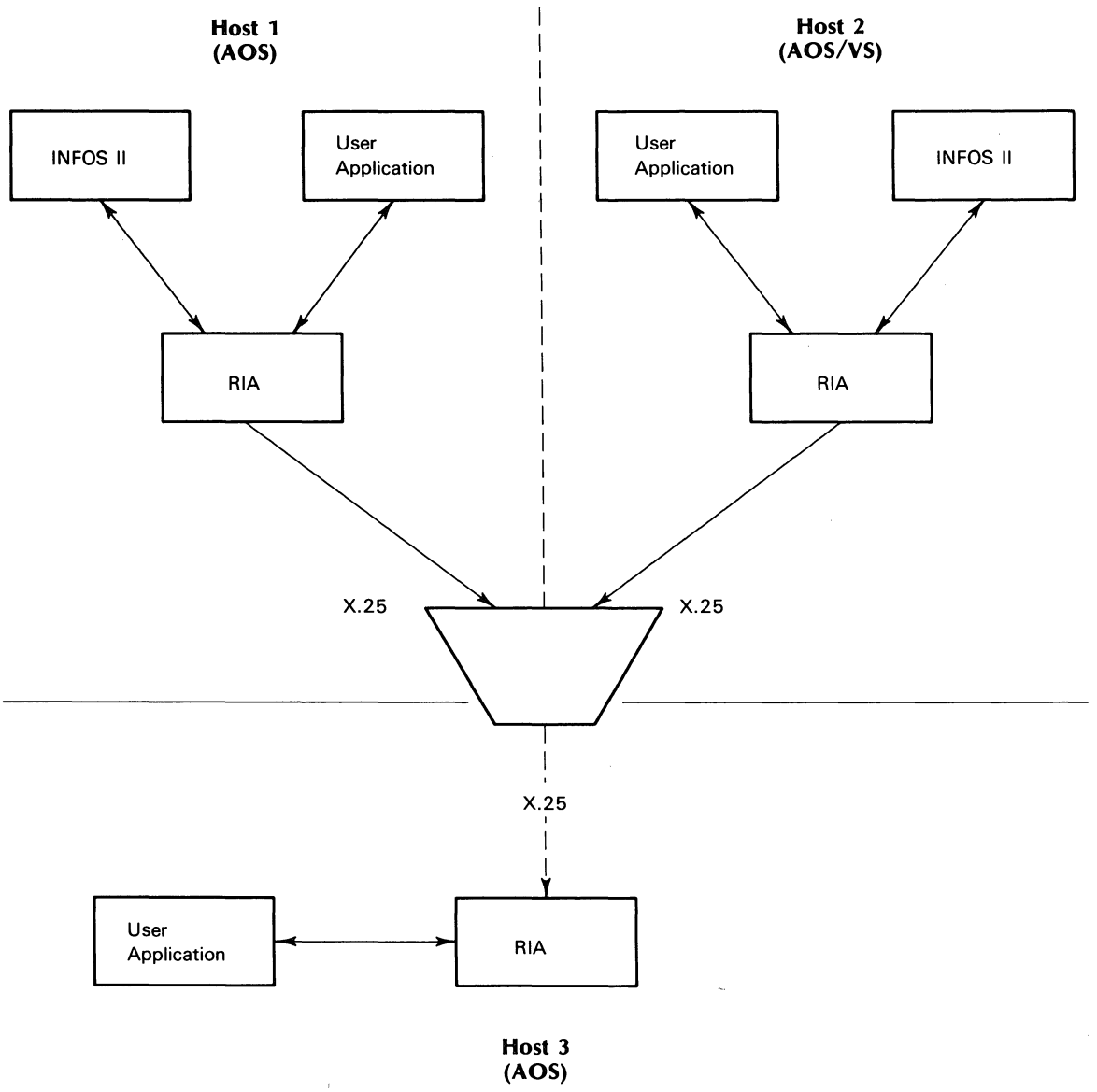
Key:

⟷ Access to INFOS II files on both hosts.

→ Access to INFOS II files on one host.
← (Arrowhead points to host with INFOS II files.)

xxxxx No file access.

Figure 11-1 shows a sample configuration with three hosts.



A user on Host 1 can gain access to local files and remote files on Host 2.
A user on Host 2 can gain access to local files and remote files on Host 1.
A user on Host 3 can gain access to remote files on Hosts 1 and 2.
Host 3 has no local INFOS II files of its own, and therefore does not need to run the INFOS II process.

ID-01147

Figure 11-1. A Distributed INFOS II System with a Three-Host Network

Loading and Bringing Up a Distributed INFOS II System

Before you can bring up a distributed INFOS II system, you need a properly installed XODIAC network. This must at least consist of AOS or AOS/VS, X.25 and NETOP at each node. The only network components that RIA interacts with are X.25 and NETOP.

Loading RIA

Where you load RIA depends on whether or not you also have the INFOS II system, and which operating system you use. Consult your RIA Release Notice documentation for information on where to load RIA on your system.

After you load the RIA release tape, use the network generation program, NETGEN, to include RIA as a new XODIAC network process name (NPN) in your network specification file. If you are bringing up a XODIAC network for the first time, include RIA as an NPN during the initial NETGEN session that creates the network.

If you are adding RIA to an existing network, we suggest that you bring down the network and reconfigure the specification file when you add the RIA NPN. At the end of this chapter, we list a sample NETGEN dialog showing how to incorporate RIA as a new NPN in a previously generated network.

If you have your network installed, you will already have the XODIAC error message file, NETERMES.OB linked into your system error message file. Otherwise, consult your NET documentation for information on how to do this.

If the INFOS II system is running on your host, you will already have the INFOS II error message file linked into your system error message file. On machines running AOS, this file is INFOERMES.OB; on machines running AOS/VS, this file is INFOERMESVS.OB. If you do not have this file, relink your system error message file to include the appropriate INFOS II error message file.

Bringing Up the RIA Process

Once you have loaded RIA onto your system and added the RIA NPN to your network specification file, you can bring up the RIA process. To do this, use the CRIA.CLI macro, supplied on your RIA release tape, with the START command:

```
CRIA START
```

RIA responds with this message:

```
FROM PID xx: (RIA)
     ENABLE REPORT
     URIA ENABLED
     SRIA ENABLED
     TIME hh:mm:ss; dd-mmm-yyyy
```

We explain URIA and SRIA later in this chapter.

You might want to include the NETOP START command for RIA in the UP.NETWORK.CLI macro that you use to bring up the XODIAC system. You can find an explanation of the UP.NETWORK.CLI macro in the *XODIAC™ Network Management System Guide for Managers and Operators.*

RIA runs in :NET. If needed, RIA creates break files in :NET with the following naming convention:

```
?RIA_mm.dd_hh.mm_errorcode.BRK
```

As we said before, every host in a distributed INFOS II network must run the RIA process. Remember, however, that not every host in the network must run the INFOS II system. Only

those hosts that have INFOS II files must have the INFOS II system. Also remember that other Data General products that the INFOS II system might require, such as the Common Logger (COMLOG) and Sort/Merge, must also reside on the host where the INFOS II system is running.

How to Issue Remote Requests

Using a Distributed INFOS II network does not affect your application program. You issue remote requests just as you issue local requests. Request packet formats are identical, regardless of the file location. You can also transport your AOS INFOS II application programs or your AOS INFOS II files to any network host without modifying the code or rebuilding the files. However, in order to move AOS application programs to an AOS/VS system, or vice versa, you must compile and link them again.

To issue a remote OPEN request, specify a filename just as you would in a local OPEN request. Because remote directories are not allowed on a search list, you must specify a fully qualified pathname, or a filename that is a link to a fully qualified pathname. A fully qualified pathname has this form:

```
:NET:hostname:infosindexpathname
```

The :NET:hostname portion of the pathname indicates that RIA will gain access to this file through X.25, hostname being the host where the file resides.

You need not include a database filename in the pathname, since a database must reside in the same directory as its associated index(es). In fact, the INFOS II File Definition Packet (?FDP) accepts index pathnames only, not database pathnames.

Using Link Files in a Distributed INFOS II System

You can take advantage of link files in a distributed INFOS II system by substituting them for index filenames. By doing this, you eliminate the need to change the filenames in your application programs when you change your host configurations or relocate files. You simply use the filenames as links to relocated INFOS II index files. As your distributed INFOS II configurations change, you adjust the link resolution rather than the index filename.

Assume, for example, that your application program uses an index filename as a link to :UDD:USER:FILE0 in an INFOS II OPEN request. You could relocate FILE0 (and its associated database and indexes) to the directory MYFILES in your user directory on the remote host, RHOST. By deleting the original file and creating a link to the remote file, your application program remains unchanged.

To create such a link, enter the following:

```
)CREATE/LINK :UDD:USER:FILE0 :NET:RHOST:UDD:USER:MYFILES:FILE0 ⤶
```

To relocate the file, we recommend that you use the DDUMP and DLOAD utilities. Or, if you have the XODIAC File Transfer Agent (FTA) as part of your system, you can use it to relocate your files.

RIA accepts links on both local and remote hosts. You cannot, however, use a link to a remote link file, which itself links to a file on a remote host. Consider, for instance, the previous example in which we created a link to RHOST. If :UDD:USER:MYFILES:FILE0 were a link to a file on a host remote from RHOST, RIA would return an error to your request.

How RIA Satisfies Remote Requests

RIA automatically sends all requests to the appropriate host, based on the INFOS II channel number that the system returns when you open the file. You issue requests, both remote and local, as you normally would. How RIA receives the remote request depends on whether you issue it from the AOS INFOS II system or the AOS/VS INFOS II system.

If you use the AOS INFOS II system, then ICALL.OB (the runtime routine you use to link INFOS II application programs) will communicate with RIA. ICALL.OB determines the destination of the request. If it is remote, the request will go to RIA. RIA, in turn, will forward the request to the appropriate remote INFOS II process. If it is local, the request will go directly from ICALL.OB to the local INFOS II process.

If you use the AOS/VS INFOS II system, then INFOS_LS (the INFOS II local server process), will communicate with RIA. INFOS_LS determines whether a request is local or remote. If it is remote, INFOS_LS creates a packet containing information that the remote process will need to perform the request, and then sends the packet to RIA. RIA will forward the packet to the appropriate remote INFOS II process. If it is local, the request will go directly to the local INFOS II process.

Each host in a distributed INFOS II system runs one RIA process. But, this single process can play either of two roles:

- Using RIA (URIA).
- Serving RIA (SRIA).

When ICALL or INFOS_LS encounters a remote request, the local RIA process assumes the URIA role, and the remote RIA process assumes the SRIA role. URIA establishes a connection with the remote host using X.25 and then communicates with SRIA, as shown in Figure 11-2.

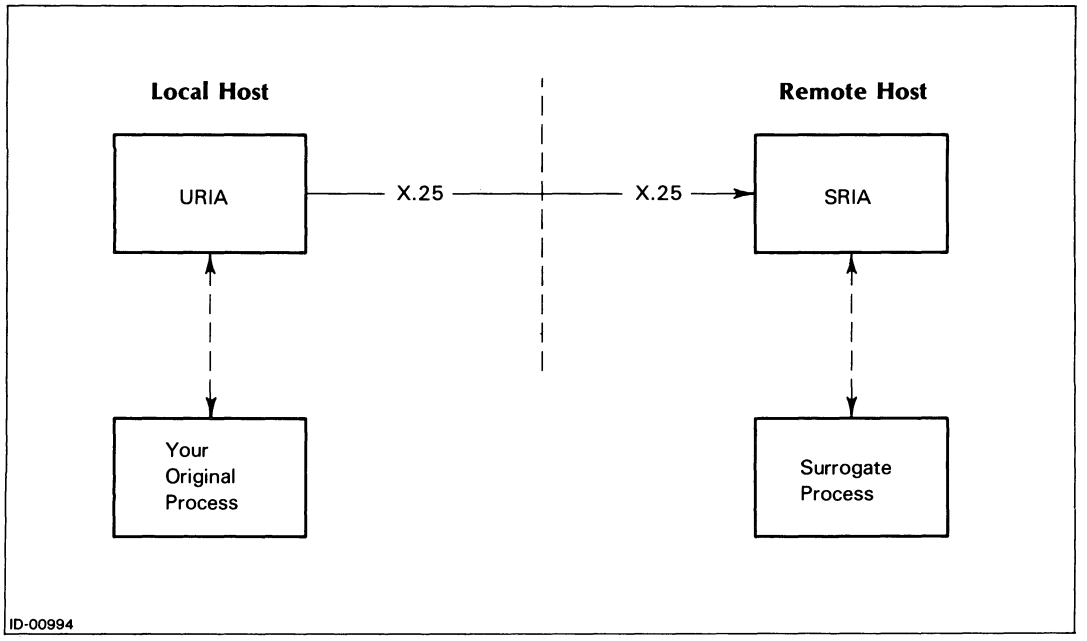


Figure 11-2. Obtaining a Remote Resource Using RIA

Next, SRIA verifies your remote host access privileges, and creates a surrogate process with your username and access privileges. The surrogate process runs as a son of SRIA. It controls the remote processing that your request requires, and returns a response to the local URIA process, via X.25. Then, URIA and the INFOS II system return this reply to the original local process. The surrogate process on the remote host continues to handle your remote requests without further SRIA assistance.

Errors

When the RIA process on your local host receives a request, it checks the request packet before sending it across the network. RIA generally will notify you immediately with the appropriate INFOS II error code if the request is invalid — the same way that INFOS II generally will notify you of an invalid local request. If it is invalid, RIA will not forward it to the remote host. For example, if you try to specify an illegal number of volumes in an FDP, RIA will notify you immediately, and will not send the request across the network.

Certain errors, however, are detected by the RIA or the INFOS II process on the remote host. For example, the local RIA process cannot tell whether a remote file exists or not. Therefore, the ***FILE DOES NOT EXIST*** error must come from the remote host.

Access Privileges

To gain access to remote INFOS II files, you must have certain access privileges on your local host and the remote host where the files reside. We explain these privileges in the following sections.

Local Privileges

You need these privileges on your local system to use the network:

- You must have at least the E (execute) access privilege to the :NET directory to gain access to the network.

  For example, if the ACL is

  ```
  ACL :NET USERA,,USERB,E
  ```

  then only USERB can gain access to the network.

- You must have at least the RE (read and execute) access privileges to the :NET:HOSTNAME file of any remote host to which you are trying to gain access. (The HOSTNAME file is an AOS/VS or AOS HST file type.) Without the RE access privileges, you cannot gain access to INFOS II files on any other host in your network.

  For example, if

  ```
  ACL :NET:HOST1 USERA,,USERB,RE
  ACL :NET:HOST2 USERB,E,USERA,RE
  ```

  then USERA can gain access to HOST2 only, and USERB can gain access to HOST1 only.

Remote Host Access Privileges

Before SRIA creates the surrogate process that will service your RIA request, it checks your access privileges on the remote host. If you own a user profile on the remote host, SRIA creates a surrogate process.

NOTE: Your remote profile must contain the same username/password pair as your local profile. It must also include the ACCESS LOCAL RESOURCES FROM REMOTE MACHINES privilege.

The network manager should ensure that each username/password pair is unique for each user and identical on all hosts where a user needs access. This security measure protects users on different hosts from inadvertently gaining access to other users' INFOS II files.

For example, suppose you have a profile on one machine and your username is SMITH and your password is JRS. Let's say you have a profile on another machine, but there is already a SMITH on that machine so your username is JANE. Even if your JANE profile has the password JRS, you won't be able to communicate between the two machines. In a case such as this, the network manager would have to ensure that the SMITH on the second machine did not have JRS for a password so you would not have remote access to his or her files.

Likewise, if you have profiles on two machines and your username is SMITH on each one, but your password on one is JRS and on the other is TJR, you won't be able to communicate between them.

ACLs on the Remote Host

Like the ACLs on a local INFOS II file, the ACLs on a remote file determine your access privileges to it. R access allows you to make nonmodifying requests; W allows modifying requests, and so on. All access privileges in existence when you issue an OPEN request remain valid until you close your connection to the file, regardless of anything you do while the file is open.

If, for example, you have opened an INFOS II file, and you change your username/password pair while the file is open, that change has no effect during the current session. If you close the file, however, and then attempt to reopen it, you will fail because your username/password pair is invalid.

If you are in Superuser mode on the local host, the INFOS II system does not grant you Superuser status on a remote host automatically. You already must have the Superuser privilege on a remote host to take advantage of it.

For example, assume that your username/password pair agrees on three hosts. You have the Superuser privilege on host 1 and 2, but not on host machine 3. If you are in Superuser mode when you make a remote request to host 2, you'll have full access to its files. If, however, you make a remote request to host 3, you'll have only the access privileges that the ACL on each file of that machine specifies for you.

Abnormal Terminations

If your application program terminates abnormally or without closing its INFOS II files, AOS or AOS/VS obituary messages will notify both RIA and the INFOS II system that this has occurred. The INFOS II system will then close both your local and remote INFOS II files.

In your application program, you can create a task to monitor obituary messages from the operating system. In the event that either the RIA process or the INFOS II process on your machine terminates, the monitor will notify you. You can then cancel any tasks with outstanding requests to the terminated process.

The ICALL.OB routine in the AOS INFOS II system has two entry points you can use to determine which process terminated. These entry points are ?IN.PID and ?II.PID. They contain the process IDs of RIA and the INFOS II system, respectively. If your application program monitors obituary messages, you can compare the process identification (PID) of the terminated process with the contents of ?IN.PID and ?II.PID to determine if either terminated. In the event that one has terminated, you can cancel the specific tasks waiting for a response from that process, or have your application take whatever action is appropriate. Certain applications might be able to continue with only the remaining process.

If communication between your local RIA process and the remote INFOS II process(es) breaks for any reason, your application program will receive error returns to its requests. These error returns describe the problem. When this happens, the affected INFOS II process closes any remote channels you have open.

If a remote INFOS II process terminates, you cannot close the remote file. Your application program, however, will receive an INFOS II or X.25 error code that indicates the termination. You should then proceed with the appropriate file recovery procedures as described in Chapter 7.

For more details about obituary messages and how to receive them, see the *AOS Programmer's Manual* or the *AOS/VS Programmer's Manual Volume 1, System Concepts.*

Using RIA with Utilities and Related Software Products

Unless you have the XODIAC system's Virtual Terminal Agent, you must run most INFOS II utilities on the same host machine as their target files. The INQUIRE utility, however, is an exception to this requirement. You can use INQUIRE to gain access to remote files running under RIA or VTA.

If you run the INFOS II utilities that use tape (DDUMP, IXLOAD, and IDUMP on the AOS INFOS II System), you must mount the tape on the machine where the utilities and files reside.

Performance and Application Considerations

Response time for remote requests could be longer than for local requests for several reasons. One is that remote requests travel a longer distance than local requests, and the type of devices over which the requests travel affects how quickly they arrive. Another is that when request packets refer to large amounts of information, they require more resources to transmit that information across the network. Therefore, large data records and processing packets with many levels of index information take longer to process. Other network activity, such as file transfers or other XODIAC applications, also slow down response time.

An appropriate network configuration, however, can help compensate for these factors. In general, we recommend that you place files on the host machine of the users who gain access to them most frequently. This way, most users have direct access to the files, and remote users do not have to compete as heavily for network resources.

Loopback Mode

For development and testing purposes, you can take advantage of XODIAC X.25 loopback mode (explained in the *XODIAC™ Network Management System Guide for Managers and Operators*. Loopback mode allows you to simulate a distributed INFOS II system without using the resources of another machine. To use loopback mode, you include the local machine's hostname in the fully qualified pathname to a local index file. You can then have local requests travel to and from your local INFOS II process via your local RIA and X.25 processes.

Instead of using :UDD:USER:MYFILE in a request, for example, use :NET:MYHOST:UDD:USER:MYFILE. The presence of :NET:MYHOST in the pathname tells ICALL or INFOS_LS that it needs to use RIA. The local host tells X.25 to return the request to the local RIA.

The local INFOS II process treats local requests made via X.25 loopback mode as if they were remote requests. Hence, an application that simultaneously attempts to open a local file both directly and using loopback mode appears as two unique OPEN requests to the INFOS II system. If both OPEN requests were exclusive, the INFOS II system would deny the second request after it had processed the first, and you would receive the following error message:

```
IOFO2 SOMEONE HAS THE INDEX EXCLUSIVELY OPENED, YOU CAN'T OPEN IT.
```

Errors

As we mentioned before, you might receive errors from your local host or the remote host. We list these errors and their causes here.

Local Errors

[IOINR] RIA PROCESS NOT RUNNING

RIA is not running on your local host. The INFOS II system detects this error when you make a remote request and RIA does not exist.

[IOCRD] CANNOT RETURN DATA - ILLEGAL ADDRESS

The pointers to the buffers in your local address space are invalid.

[IOMRH] REACHED LIMIT OF REMOTE HOSTS

You tried to exceed the maximum number of remote hosts you can gain access to simultaneously.

[IORFR] ACCESS TO REMOTE INFOS II FILES RESTRICTED TO VTA

You attempted to use a utility other than INQUIRE to gain access to an INFOS II file on a remote host. You must have the XODIAC Virtual Terminal Agent (VTA) as part of your network to use the other INFOS II utilities with remote files. If you do not have VTA, execute the utility from a CLI process on the host where the target files reside.

[IOSMF] RUNTIME VERSION ERROR -- INCOMPATIBLE SINGLE MESSAGE FORMAT

The version of RIA on your local host and the version of RIA or the INFOS II system on the remote host are incompatible.

[IOMRC] REACHED LIMIT OF CHANNELS ON REMOTE HOST

You tried to exceed the maximum number of remote channels, from one host to another, that RIA supports.

Remote Errors

[IODRS] REMOTE SURROGATE TERMINATED

The surrogate process on the remote host is terminated. You can find more information about what caused this error on the remote operator console and in the remote :NET:LOGFILES directory. You can also check for surrogate breakfiles on the remote host.

[IOINN] RUNTIME VERSION ERROR -- ICALL DOESN'T SUPPORT INFOS II NETWORKING

This error applies to the AOS INFOS II system only. If you get this error, you must relink your program with the latest version of ICALL.OB.

[IORNA] REMOTE RIA NOT ACCEPTING NETWORK REQUESTS

The remote host is not running the INFOS II system. The INFOS II system either terminated or was never brought up.

[IORNR] REMOTE RIA NOT RUNNING

No RIA process is running on the remote host.

[IOPNF] REMOTE USER PROFILE NOT FOUND

You used an invalid username. The username does not exist as a profile on the remote machine.

[IORVR] RUNTIME VERSION ERROR -- REBIND WITH NEW INFOS II INTERFACE MODULE

You used an old version of ICALL. Upgrade to the current revision.

Sample NETGEN Dialog

The following is an example of a NETGEN session that would add RIA as an NPN to a previously created network. For more information about this process, see the *XODIAC™ Network Management Guide for Managers and Operators.*

To begin the NETGEN session, type

```
)DIRECTORY :NET:NETGEN ↓

)XEQ NETGEN ↓
```

NETGEN presents a menu on your screen. Choose the "Access/Update Spec File" option from the menu.

NETGEN presents another menu on your screen. Choose the "Manage NPN Configurations" option from the menu.

NETGEN presents another menu on your screen. Choose the "Add NPN Configurations" option from the menu.

When you add a new NPN, NETGEN requests a filename that will identify the network process, usually the host name with the suffix *spec.* It then requests a network process name (NPN); in this case, RIA. The menu will also prompt you to list access privileges for this NPN. The access privileges for the RIA NPN are usually +,RE.

Note that network process names are case-sensitive; thus, ria will not match RIA. If the network process names do not match exactly, you will be unable to establish a connection with the remote process and will always receive the error, *CLEAR - No User Waiting* from the X.25 process.

When you complete this section of the NETGEN process, you can use the "Pop to Previous Screen" option for all menus until you reach the original menu. You should then select the "Create Configuration Files" option.

The "Create Configuration Files" option will request the spec filename that you specified when you added the RIA NPN. When you do this, NETGEN deletes the configuration files in :NET and recreates new files using the spec file you specified.

You can then use "Terminate Process" option (option "T") from the original menu, and terminate the NETGEN session.

RIA Operator Commands

NETOP has an IPC port in :PER that receives messages intended for either URIA or SRIA. To issue commands to this IPC port, use the CRIA macro with the command you want. For example:

```
CRIA command-line
```

The resulting execution reports arrive through NETOP's IPC port, @RIA_OPERATOR. NETOP then returns them to the terminal or file you have chosen to receive them. By default, NETOP sends these reports to the operator's console. You can change this default with the RIA SET command.

If RIA is not running, you must first bring up the agent with the START command, as we explained in the "Bringing Up the RIA Process" section, earlier in this chapter. NETOP's IPC port receives messages intended for XODIAC processes, whether they are running or not.

The next section includes NETOP commands. You can use unique abbreviations for these commands and switches.

ACCOUNT

Initiates the RIA accounting facility.

Format

CRIA ACCOUNT

Switches

The ACCOUNT command has no switches.

Description

The ACCOUNT command initiates the RIA accounting facility. This facility records the services performed for each user in the system log file for either billing purposes or system load analysis.

By default, accounting is off when a RIA process starts running. If you issue this command when accounting is already on, you will receive a successful execution reply, and accounting will stay on.

Example

```
)CRIA ACCOUNT ↓
)
 .
 .
 .
 From Pid 7: (NETOP)
          TIME: hh:mm:ss
)
 From Pid 7: (RIA)
 ACCOUNTING  ON
 TIME: hh:mm:s  ;dd-mmm-yyyy
```

CONNECTIONS (AOS/VS only)

Sets or returns the number of active virtual connections URIA can service.

Format

CRIA CONNECTIONS *[/switch] [maximum value]*

The *maximum value* argument is a decimal value.

Switches

/GLOBAL	Defines the total number of connections for all local (URIA) customers.
/CUSTOMER	Defines the total number of connections for any one local customer. This is the default.

Description

The CONNECTIONS command sets or returns the current value of the parameter that limits the number of active virtual connections the URIA agent can service at any time. If you omit the switch, the commmand will default to the /CUSTOMER option.

If you do not use an argument with this command, the system will return the current value of the parameter. If you use an argument, the system will interpret the number as a new decimal value for the parameter. If you use an argument of −1, the system will reset the parameter to the default value.

Example

```
)CRIA CONNECTIONS )
)
 .
 .
 .
From Pid 7 : (NETOP)
  TIME :hh:mm:ss
)
From Pid 7 : (RIA)
    CUSTOMER CONNECTION LIMIT
    SET TO: 10
    TIME: hh:mm:ss  ;dd-mmm-yyyy
```

DISABLE

Stops servicing local or remote requests, or disables accounting.

Format

CRIA DISABLE *[/switches]*

Switches

/URIA	Disables the Using RIA agent.
/SRIA	Disables the Serving RIA agent.
/ACCOUNT	Disables RIA accounting.

Description

The DISABLE command directs the RIA process to stop servicing local (URIA) or remote (SRIA) requests. You can also use it with the /ACCOUNT switch to stop the accounting function. If you do not specify the URIA or the SRIA command switches, you disable both agents.

Example

```
)CRIA DISABLE ↵
)
 .
 .
 .
 From Pid 7:  (NETOP)
       TIME:   hh:mm:ss
)
 From Pid 7:  (RIA)
       DISABLE REPORT
       SRIA DISABLED
       URIA DISABLED
       TIME: hh:mm:ss  ;dd-mmm-yyyy
```

ENABLE

Starts URIA or SRIA, or resets accounting.

Format

CRIA ENABLE *[/switches]*

Switches

/URIA	Enables the Using RIA agent for servicing local user requests.
/SRIA	Enables the Serving RIA agent for servicing remote user requests.
/ACCOUNT	Enables the RIA accounting function.

Description

The ENABLE command starts the URIA agent, the SRIA agent, or both, and resets the accounting function. You can use either or both of the URIA and SRIA command switches. If you don't specify one or the other, you will enable both agents. If you don't use any switches, you will enable both agents and reset accounting.

Example

```
)CRIA ENABLE ↓
)
 .
 .
 .
 From Pid 7: (NETOP)
       TIME:  hh:mm:ss
)
 FROM PID 7 (RIA)
       ENABLE REPORT
       URIA ENABLED
       SRIA ENABLED
       ACCOUNTING OFF
       TIME: hh:mm:ss ;dd-mmm-yyyy
```

NOACCOUNT

Turns off accounting.

Format

CRIA NOACCOUNT

Switches

The NOACCOUNT command does not have any switches.

Description

The NOACCOUNT command turns off the RIA accounting facility.

Example

```
)CRIA NOACCOUNT ↓
)
 .
 .
 .
 From Pid 7: (NETOP)
       TIME: hh:mm:ss
)
 FROM PID 7: (RIA)
        ACCOUNTING OFF
        TIME: hh:mm:ss ;dd-mmm-yyyy
```

RESET

Resets global statistics accumulators to initial values.

Format

CRIA RESET

Switches

The RESET command has no switches.

Description

The RESET command resets the global statistics accumulators to their initial values. You can use this command if you want to keep daily statistics.

Note that the /RESET switch for the STATUS command performs the same function as the RESET command.

Example

```
)CRIA RESET ↓
)
 .
 .
 .
 From Pid 7: (NETOP)
       TIME: hh:mm:ss
)
 FROM PID 7: (RIA)
       GLOBAL STATISTICS RESET
       TIME: hh:mm:ss ;dd-mmm-yyyy
```

SET

Specifies the destination for RIA command responses and exception reports.

Format

CRIA SET *[/switches]*

Switches

/TIME	Includes the time of day prompt on all RIA reports. This is the default.
/NOTIME	Deletes the time of day prompt from RIA reports.
/DATE	Includes the date on all RIA prompts.
/NODATE	Deletes the date prompt on RIA reports. This is the default.
/OUTPUT	Resets the output destination for RIA reports to the console owned by NETOP's father process.
/OUTPUT=n	Sends all RIA reports to the console owned by pid *n*, where *n* is a decimal number.
/OUTPUT=processname	Sends all RIA reports to the console owned by *processname.*
/OUTPUT=@consolename	Sends all RIA reports to the console *@consolename.*
/LIST	Writes all RIA reports to the default logfile in :NET:LOGFILES.
/LOG	Writes all RIA reports to the default logfile in :NET:LOGFILES.
/LIST=filename	Writes all RIA reports to the logfile you specify in *filename.*
/LOG=filename	Writes all RIA reports to the logfile you specify in *filename.*
/NOOUTPUT	Terminates output mode.
/NOLIST	Terminates the log function.
/NOLOG	Terminates the log function.
/PARAMETERS	Returns the current RIA logfile name and RIA output destination.

Description

The SET command specifies the destination for all RIA command responses and exception reports. You can send reports to both an output destination and a logfile destination. If the system encounters an error when issuing a report, it will reset the output destination to the console owned by NETOP's father process. If the system encounters an error when issuing a report to NETOP's father's console, the report will be lost. RIA will create a default logfile if one does not already exist and will name it RIA_mm_dd_yy.LOG.

Example

```
)CRIA SET/PARAMETERS ↓
)
 .
 .
 .
 From Pid 7: (NETOP)
       TIME: hh:mm:ss
)
 FROM PID 7: (RIA)

       PARAMETERS:

       LOGFILE = :NET:LOGFILES:RIA:RIA_12_29_83.LOG
       OUTPUT = FATHER'S CONSOLE
       TIME: hh:mm:ss ;dd-mmm-yyyy
```

START

Creates the RIA process and initializes the RIA parameters.

Format

CRIA START *[/switches]*

Switches

/SWAPPABLE	Initializes RIA as a swappable process. This is the default.
/PREEMPTIBLE	Initializes RIA as preemptible process.
/WSMIN=x	(AOS/VS only) Sets the minimum working set size for RIA to *x*.
/WSMAX=x	(AOS/VS only) Sets the maximum working set size for RIA to *x*.
/URIA	Enables Using RIA support for local users.
/SRIA	Enables Serving RIA support for remote users.

Description

The START command requests the NETOP process to create the RIA process and initialize RIA parameters. Only one RIA process can run at a time. If you do not use any switches with START, you will initialize RIA as a swappable process and enable both URIA and SRIA.

Example

```
)CRIA START ↓
)
 .
 .
 .
 From Pid 7: (NETOP)
       TIME: hh:mm:ss
)
 From Pid 7: (RIA)
       ENABLE REPORT
       URIA ENABLED
       SRIA ENABLED
       TIME: hh:mmm:ss ;dd-mmm-yy
```

STATUS

Generates a RIA status report.

Format

CRIA STATUS *[/RESET] [local pid, or process name]*

Switch

/RESET Resets global statistics.

Description

The STATUS command generates a RIA status report. If you don't specify an argument, RIA will generate a global status report. If you do include an argument, it must be either a local customer of the URIA agent, or a local surrogate of the URIA agent.

The RESET switch clears the statistics counters and sets the time to the current system time. RESET displays the previous contents of the counters (before clearing) and the time statistics at the console that NETOP has assigned for RIA output. RESET applies to the global statistics only.

Example

```
)CRIA STATUS ↓
)
 .
 .
 .
 From Pid 7: (NETOP)
       TIME: hh:mm:ss
)
 From Pid 7: (RIA)
       STATUS (GLOBAL); TIME: hh:mm:ss ;dd-mmm-yyyy
       STATUS INTERVAL: hh:mm:ss
       CUSTOMERS - CUR: 0 TOT: 0
       SURROGATES - CUR: 0 TOT: 0
       VIRTUAL CIRCUITS -
             NO ACTIVE CIRCUITS
       TIME: hh:mm:ss ;dd-mmm-yyyy
```

STATUS (continued)

```
From Pid 7: (RIA)
      STATUS (URIA)
      CONNECTIONS - CUR: 0 TOT: 0
      DURATION - CUSTOMERS: 0:00:00
           CONNECTIONS: 0:00:00
      BYTES - TRANSMITTED: 0
           RECEIVED: 0
      CUSTOMER LIST -
           NO ACTIVE CUSTOMERS
      TIME: hh:mm:ss ;dd-mmm-yyyy

From Pid 7: (RIA)
      STATUS (SRIA)
      DURATION - SURROGATES: 0:00:00
      BYTES - TRANSMITTED: 0
           RECEIVED: 0
      SURROGATE LIST -
           NO ACTIVE SURROGATES
      TIME: hh:mm:ss ;dd-mmm-yyyy
```

SURROGATES (AOS/VS only)

Sets or returns the maximum number of surrogates allowed on the local host at any one time.

Format

CRIA SURROGATES *[maximum value]*

where *maximum value* is a decimal value.

Switches

The SURROGATES command has no switches.

Description

The SURROGATES command sets or returns the current value of the maximum number of surrogates that can exist on the local host at any one time. If you do not use an argument, the system will return the current parameter value. If you do include an argument, the system will assume it specifies a new decimal value for the parameter. If the argument is −1, the system will reset the parameter to the default value.

Example

```
)CRIA SURROGATES ↓
)
 .
 .
 .
 From Pid 7: (NETOP)
        TIME: hh:mm:ss
)
 From Pid 7: (RIA)
        SURROGATE LIMIT
        SET TO: 20
        TIME: hh:mm:ss ;dd-mmm-yyyy
```

TERMINATE (AOS/VS only)

Terminates a local URIA customer or a surrogate remote SRIA customer.

Format

CRIA TERMINATE *[pid]*

where *pid* is the Process ID of a local customer.

Switches

The TERMINATE command has no switches.

Description

The TERMINATE command directs the RIA process to terminate a service for a local customer of the URIA agent, or to terminate a surrogate process for a remote customer of the SRIA agent.

You must include an argument with the TERMINATE command.

Example

```
)CRIA TERMINATE 13 ↓
)
 .
 .
 .
FROM PID 2: (RIA)
      TERMINATION OF SERVICE
      FOR PID: 13
      TIME: hh:mm:ss ;dd-mmm-yyyy
```

TIMEOUT

Specifies the time a surrogate process will wait between requests before RIA will terminate a surrogate.

Format

CRIA TIMEOUT *[maximum value]*

where *maximum value* indicates minutes in decimal.

Switches

The TIMEOUT command has no switches.

Description

The TIMEOUT command specifies the maximum time in minutes that a surrogate process will wait between successive user requests before RIA terminates a surrogate. If you do not include an argument, RIA returns the current value for this parameter. If you do specify an argument, RIA will assume it is a decimal value that specifies the new timeout value, unless you specify 0 (return the current value) or −1 (restore the default value). In any case, a surrogate will not terminate at the end of its timeout if it still has an open file.

Example

```
)CRIA TIMEOUT ↓
)
 .
 .
 .
 From Pid 7: (NETOP)
       TIME: hh:mm:ss
)
 From Pid 7: (RIA)
       SURROGATE TIMEOUT SET
       TO: 8 (MINUTES)
       TIME: hh:mm:ss ;dd-mmm-yyyy
```

End of Chapter

Appendix A
INFOS II Error Messages

All of the INFOS II errors are in the octal range 7000 to 7777. Within this broad range are subsets of related errors, which we show below. Many of the error messages are self-explanatory. For those that are not, we include a brief description at the end of this appendix.

General Errors (7000 - 7060)

```
7000 (IOUER) ILLEGAL COMMAND
7001 (IONMD) ALL VOLUMES AT MAXIMUM SIZE
7002 (IOCON) CONTINUING EXECUTION
7003 (IOINI) REVISION X.XX OF AOS INFOS II ERROR CODES
7004 (IOSPE) ILLEGAL RELATIVE MOTION
7005 (IOICE) INVALID CURRENT ENTRY
7006 (IOTLV) WARNING - POSITIONED ABOVE MAIN INDEX (TOP LEVEL)
7007 (IOSNA) SUBINDEXES NOT ALLOWED
7010 (IOSNP) SUBINDEX NOT DEFINED
7011 (IOEST) END OF SUBINDEX
7012 (IODPE) A REQUESTOR ON ANOTHER CHANNEL IS POSITIONED ON THE KEY TO DELETE
7013 (IOKAE) KEY ALREADY EXISTS
7014 (IONDR) WARNING - DATA BASE RECORD NOT PRESENT
7015 (IODRL) DATA RECORD LOCKED
7016 (IOSAE) ALREADY LINKED TO SUBINDEX
7017 (IOSTL) FILE CONSISTENCY ERROR
7020 (IOSLO) DEFINE SUBINDEX COMMAND WOULD EXCEED MAX. INDEX LEVELS FOR FILE
7021 (IOSST) ENTRY HAS SUBINDEX -- DELETE ERROR
7022 (IODWK) ATTEMPT TO DELETE ENTRY WITHOUT KEYED ACCESS
7023 (IOIRI) INDEX ENTRY ALREADY POINTS TO A DIFFERENT RECORD
7024 (IOWWK) ATTEMPT TO WRITE ENTRY WITHOUT KEYED ACCESS
7025 (IOENL) PARTIAL RECORD LOCKED
7026 (IOLVR) TOO MANY INDEX LEVELS
7030 (IOKPE) KEYED POSITIONING ERROR
7031 (IOIEN) INVALID ENTRY NUMBER IN INDEX
7032 (IOINA) INVALID NODE ADDRESS
7033 (IODIP) DELETE INDEX ERROR -- ANOTHER USER IS POSITIONED IN THAT SUBINDEX
7034 (IOTML) LOCK REQUEST EXCEEDS MAXIMUM NUMBER OF LOCKS REQUESTED AT OPEN
7035 (IOSYS) UNEXPECTED SYSTEM CALL ERROR RETURN
7036 (IODNS) DUPLICATE KEY NOT ALLOWED IN SUBINDEX
7037 (IOIDE) FILE CONSISTENCY ERROR
7040 (IOMAP) SPACE MANAGEMENT MAP CONSISTENCY ERROR
7041 (IOOCC) OCCURRED AT LOCATION
7042 (IOACE) CHANNEL OPENED READ-ONLY, OR READ-ONLY ACL; CANNOT MODIFY FILE
7043 (IONMT) SUBINDEX NOT EMPTY
7044 (IORTL) KEY FILE RECORD TOO LARGE
7045 (IONIO) KEY NOT IN ORDER
7046 (IOLPR) ILLEGAL PARTIAL RECORD LENGTH--USE 1 TO MAX ALLOWED IN SUBINDEX
7047 (IOPIU) COMMUNICATION FILE PAGE IN USE -- ACCESS DENIED
```

7050 (IOINF) INVALID IN-USE FLAG IN SINGLE MESSAGE REQUEST
7051 (IOTMU) MAXIMUM NUMBER OF USERS EXCEEDED
7052 (IOISF) INVALID SINGLE MESSAGE REQUEST FORMAT
7053 (IOMTE) MAXIMUM NUMBER OF USER TASKS EXCEEDED
7054 (IOFER) FILE ERROR -- PLEASE CLOSE
7055 (IOFE2) FILE ERROR -- CANNOT OPEN AT THIS TIME
7056 (IOFOW) FILE ALREADY OPENED WITH WRITE ACCESS, TPMS CANNOT OPEN IT
7057 (IOPOW) TPMS HAS FILE OPENED, YOU CANNOT OPEN IT FOR WRITE ACCESS
7060 (IOCOI) CANNOT ISSUE INVERTED CREATE ON FILE IN LOGGING MODE

Processing Packet Errors (7061 - 7073)

7061 (IOPCH) ?PCHN IS NOT YOUR CHANNEL NUMBER
7062 (IOPDA) ?PDAT DATA RECORD OR RETRIEVE HIGH KEY BYTE POINTER INVALID
7063 (IOPKP) ?PKPN ILLEGAL # OF KEYS -- USE 1 TO # OF SUBINDEX LEVELS IN THE INDEX
7064 (IOPLE) ?PLEN DATA RECORD BYTELENGTH EXCEEDS DATABASE PAGESIZE-8, OR IS ZERO
7065 (IOPLN) DATA RECORD LINK WORDS INVALID FOR INVERSION
7066 (IOIPS) ?PCCW NEITHER KEYED NOR RELATIVE PROCESSING SPECIFIED
7067 (IOPSI) ?PSID SUBINDEX DEFINITION PACKET ADDRESS INVALID
7070 (IOPPP) ?PPRA PARTIAL RECORD BYTE POINTER INVALID
7071 (IOLKX) LOCK/UNLOCK ERROR - EITHER BOTH LOCK AND UNLOCK SPECIFIED, OR NEITHER PARTIAL NOR DATA RECORD SELECTED
7072 (IOPNZ) INITIAL PACKET CONTAINED NONZERO RESERVED ENTRIES
7073 (IORSE) MAXIMUM REQUEST SIZE EXCEEDED

Key Definition Packet Errors (7103 - 7111)

7103 (IOKTY) ?KTYP INCONSISTENT KEY TYPE FLAGS
7104 (IOKYL) ?KYLN ILLEGAL KEY BYTELENGTH -- USE 1 TO MAXIMUM ALLOWED IN SUBINDEX
7105 (IOKKY) ?KKYP INVALID KEY BYTE POINTER
7106 (IOKDK) KEY NOT FOUND IN SUBINDEX
7107 (IOKNZ) KEY PACKET CONTAINED NONZERO RESERVED ENTRIES
7110 (IOKDO) KEY DESCRIPTOR LEVEL OVERFLOW
7111 (IOKPR) TOO MANY KEY DESCRIPTOR PACKETS FOR LINK SUBINDEX DESTINATION KEY PATH

Subindex Definition Packet Errors (7120 - 7124)

7120 (IONTS) ?FRNS SUBINDEX DEFINITION ROOT NODE SIZE WON'T HOLD 3 KEY ENTRIES
7121 (IONTL) ?FRNS SUBINDEX DEFINITION ROOT NODE SIZE TOO LARGE FOR PAGE SIZE
7122 (IOMKL) ?FMKL SUBINDEX DEFINITION MAXIMUM KEYLENGTH EXCEEDS ?MXKL BYTES
7123 (IOPRL) ?FPRL SUBINDEX DEFINITION PARTIAL RECORD LENGTH EXCEEDS ?MXPR BYTES
7124 (IOSNZ) SUBINDEX DEFINITION PACKET CONTAINED NONZERO RESERVED ENTRIES

Open Errors (7135 - 7165)

```
7135 (IOVER) FILE VERSION CONFLICT
7136 (IOPVR) PARAMETER VERSION ERROR -- REASSEMBLE WITH NEW USER PARAMETER
     FILES
7137 (IORVR) RUNTIME VERSION ERROR -- RELINK WITH NEW INFOS II ICALL RUNTIME
7140 (IODDM) INDEX AND DATABASE MUST BE IN SAME PARENT DIRECTORY
7141 (IONIV) NOT AN INFOS II VOLUME FILE
7142 (IOTMO) YOU ATTEMPTED TO OPEN MORE THAN ?MXCHN INFOS II CHANNELS
7143 (IODNC) DIRECTORY NAME DOESN'T BEGIN WITH COLON -- PROBABLE RUNTIME ERROR
7144 (IONLR) ?FNLR SPECIFIED MAXIMUM NUMBER OF SIMULTANEOUS LOCKS EXCEEDS ?MXLK
7146 (IODBF) ?FDBP DATABASE FILE DEFINITION PACKET ADDRESS INVALID
7147 (IOAMD) ?FAM1 ?FAM2 ACCESS METHOD ILLEGAL IN ?FFLG
7150 (IONIL) ?FNIL MAXIMUM INDEX LEVELS ILLEGAL -- USE 1 TO ?MXIL
7152 (IOIOO) ILLEGAL OPEN OPTIONS
7154 (IODNM) DATABASE MUST BE SIMPLE FILENAME WITH NO SPECIAL CHARACTERS
7155 (IOFDP) INFOS II INDEX FILE DEFINITION PACKET POINTER INVALID
7156 (IOIVP) IVERIFY IN PROGRESS
7157 (IOLOP) LOG FILE OPEN REQUEST EXCEEDS MAXIMUM NUMBER ALLOWED
7160 (IONCL) COMLOG PROCESS NOT RUNNING
7161 (IOOVE) RUNTIME VERSION ERROR -- INCOMPATIBLE WITH CURRENT REV OF SYSTEM
7162 (IOINR) RIA PROCESS NOT RUNNING
7163 (IOINN) RUNTIME VERSION ERROR -- ICALL DOESN'T SUPPORT INFOS II
     NETWORKING
7164 (IOPNQ) LOCAL FILE PATHNAME MUST NOT BE NETWORKING QUALIFIED
7165 (IORFR) ACCESS TO REMOTE INFOS II FILES RESTRICTED TO VTA
7166 (IOSMF) RUNTIME VERSION ERROR -- INCOMPATIBLE SINGLE MESSAGE FORMAT
```

General Error (7170)

```
7170 (IOSMR) SPURIOUS MESSAGE RECEIVED
```

Open Pair Errors (7171 - 7242)

```
7171 (IOFNI) NOT AN INFOS II INDEX FILE
7172 (IODNI) NOT AN INFOS II DATABASE FILE
7173 (IOFNA) ?FNAM INDEX FILENAME BYTE POINTER INVALID
7174 (IODNA) ?FNAM DATABASE FILENAME BYTE POINTER INVALID
7175 (IOFPA) ?FPAG INDEX PAGESIZE ILLEGAL -- USE 2048 OR 4096 BYTES
7176 (IODPA) ?FPAG DATABASE PAGESIZE ILLEGAL -- USE 2048 OR 4096 BYTES
7177 (IOFFL) ?FFLG RESERVED BITS SET IN INDEX FDP FLAG WORD
7200 (IODFL) ?FFLG RESERVED BITS SET IN DATABASE FDP FLAG WORD
7201 (IOFNV) ?FNVD INDEX VOLUME COUNT ILLEGAL -- USE 1 TO ?MXVOL
7202 (IODNV) ?FNVD DATABASE VOLUME COUNT ILLEGAL -- USE 1 TO ?MXVOL
7203 (IOFNZ) PACKET CONTAINED NONZERO RESERVED ENTRIES
7204 (IODNZ) DATABASE PACKET CONTAINED NONZERO RESERVED ENTRIES
7205 (IOVVN) ?VVNP INDEX VOLUME NAME BYTE POINTER INVALID
7206 (IODVN) ?VVNP DATABASE VOLUME NAME BYTE POINTER INVALID
7207 (IOVVS) ?VVSZ INDEX MAXIMUM VOLUME SIZE EXCEEDS 1,048,576 BLOCKS
7210 (IODVS) ?VVSZ DATABASE MAXIMUM VOLUME SIZE EXCEEDS 1,048,576 BLOCKS
```

```
7211 (IOFDE) INDEX FILE DOES NOT EXIST
7212 (IODDE) DATABASE FILE DOES NOT EXIST
7213 (IOFAE) INDEX FILENAME ALREADY EXISTS
7214 (IODAE) DATABASE FILENAME ALREADY EXISTS
7215 (IOFAD) INDEX FILE ACCESS DENIED
7216 (IODAD) DATABASE FILE ACCESS DENIED
7217 (IOFO1) SOMEONE HAS THE INDEX OPEN, YOU CAN'T OPEN IT EXCLUSIVELY
7220 (IODO1) SOMEONE HAS THE DATABASE OPEN, YOU CAN'T OPEN IT EXCLUSIVELY
7221 (IOFO2) SOMEONE HAS THE INDEX EXCLUSIVELY OPENED, YOU CAN'T OPEN IT
7222 (IODO2) SOMEONE HAS THE DATABASE EXCLUSIVELY OPEN, YOU CAN'T OPEN IT
7223 (IOVNZ) INDEX FILE DEFINITION PACKET CONTAINED NONZERO RESERVED ENTRIES
7224 (IODZN) DATABASE FILE DEFINITION PACKET CONTAINED NONZERO RESERVED ENTRIES
7225 (IOFTL) INDEX PATHNAME LONGER THAN ?MXPL BYTES INCLUDING TERMINATOR
7226 (IODTL) DATABASE PATHNAME LONGER THAN ?MXPL BYTES INCLUDING TERMINATOR
7227 (IOFNR) INDEX FILENAME REQUIRED
7230 (IODNR) DATABASE FILENAME REQUIRED FOR INVERSION
7231 (IOFDC) INDEX FILE OPEN DURING SYSTEM OR INFOS II CRASH -- RUN IVERIFY
7232 (IODDC) DATABASE FILE OPEN DURING SYSTEM OR INFOS II CRASH -- RUN IVERIFY
7235 (IOVDE) AN INDEX VOLUME DOES NOT EXIST
7236 (IODVE) A DATABASE VOLUME DOES NOT EXIST
7237 (IOVMF) INDEX VOLUME MERIT FACTOR OUT OF ORDER
7240 (IODVM) DATABASE VOLUME MERIT FACTOR OUT OF ORDER
7241 (IOIVA) INDEX VOLUME FILE ACCESS DENIED
7242 (IOIVA) DATABASE VOLUME FILE ACCESS DENIED
```

Fatal Errors (7265 - 7301)

```
7265 (IOITC) REGENERATE WITH AT LEAST 3 TASKS
7266 (IONSU) CREATED WITHOUT SUPERUSER PRIVILEGES
7267 (IONIP) CREATED WITHOUT IPC PRIVILEGES
7270 (IOVIF) CREATED WITHOUT SPECIAL PROCESS PRIVILEGE ?PVIF
7271 (IOILL) INTERNAL LOGIC ERROR
7272 (IOTER) TERMINATING -- TERMINATE USERS AND RESTART INFOS II
7273 (IOREA) REQUEST ENTRY @INFOS ALREADY EXISTS
7274 (IOPER) INITIAL DIRECTORY MUST BE :PER
7275 (IOTMI) ANOTHER INFOS II RUNNING
7300 (IOIDS) INVALID SYSTEM TIME DATE.
7301 (IOVMS) VM FILE SPACE EXHAUSTED
```

Backup/Recovery Errors (7400 - 7422)

```
7401 (IOCPC) DIFFERENTIAL FILE CRASHED DURING CHECKPOINT- RUN CHECKPOINT
     UTILITY
7402 (IOCNC) CRASHED FILE WAS NOT CHECKPOINTING - RUN IRECOVER
7404 (IOCND) DATABASE FILE IS NOT IN DIFFERENTIAL MODE
7405 (IOCDB) CHECKPOINT FILE MUST BE A DATABASE FILE
7406 (IOCNO) FILE IS NOT CURRENTLY OPEN TO INFOS II
7407 (IOODV) ?FNVD DIFFERENTIAL FILE VOLUME COUNT ILLEGAL -- USE 1 TO 16
7410 (IOTP1) DATABASE FILE IS OPEN TO TPMS, YOU CANNOT OPEN IT
7411 (IOTP2) DATABASE FILE IS ALREADY OPEN TO SOMEONE, YOU CANNOT OPEN IT
```

7412 (IOTP3) DATABASE FILE OPEN TO TPMS, YOU MAY NOT CHECKPOINT
7413 (IOOPU) FILE CURRENTLY OPEN TO A UTILITY OR INFOS II
7414 (IORCV) LOG MODE FILE OPEN DURING SYSTEM OR INFOS II CRASH- RUN IRECOVER
7415 (IOROC) FILE OPEN WITH READ-ONLY ACCESS DURING SYSTEM OR INFOS II CRASH
7416 (IOUCR) FILE OPEN TO UTILITY DURING SYSTEM OR INFOS II CRASH
7417 (IOOPI) FILE CURRENTLY OPEN TO INFOS II
7421 (IOCIP) CHECKPOINT ALREADY IN PROGRESS
7422 (IODVI) DIFFERENTIAL VOLUME INCONSISTENCIES; CANNOT RESTART CHECKPOINT

Control Errors (7430 - 7452)

7430 (IOBYE) INFOS PROCESS SHUTDOWN
7431 (IOINV) INVALID CONTROL MESSAGE
7432 (IOACT) INFOS USERS ACTIVE, TERMINATION REFUSED
7446 (IOTIP) TPMS TRANSACTION IN PROGRESS
7450 (IOITI) INVALID TPMS TRANSACTION ID
7451 (IONOT) FILE(S) NOT OPEN FOR TPMS TRANSACTIONS
7452 (IOLRQ) LOGGING REQUIRED FOR TPMS TRANSACTIONS

RIA Errors (7473 - 7502)

7473 (IOIRC) ILLEGAL REMOTE COMMAND
7474 (IOCRD) CANNOT RETURN DATA - ILLEGAL ADDRESS
7475 (IOMRH) REACHED LIMIT OF REMOTE HOSTS
7476 (IOMRC) REACHED LIMIT OF CHANNELS ON REMOTE HOST
7477 (IODRS) REMOTE SURROGATE TERMINATED
7500 (IORNR) REMOTE RIA NOT RUNNING
7501 (IOPNF) REMOTE USER PROFILE NOT FOUND
7502 (IORNA) REMOTE RIA NOT ACCEPTING NETWORK REQUESTS

Interface Errors (7503 - 7514)

7503 (IOI03) INVALID INFOS II INTERFACE ARRAY
7504 (IOI04) FIRST ELEMENT OF AN ARGUMENT-PAIR IS NOT AN INTERFACE VARIABLE
7505 (IOI05) SECOND ELEMENT OF AN ARGUMENT-PAIR IS TOO LARGE
7506 (IOI06) SECOND ELEMENT OF AN ARGUMENT-PAIR NOT ALLOWED TO SET THOSE BITS
7507 (IOI07) IOPEN CALL SPECIFIES INVALID NUMBER OF LEVELS
7510 (IOI10) FIRST ELEMENT OF AN ARGUMENT-PAIR NOT APPLICABLE TO THIS SUBROUTINE
7511 (IOI11) INTERFACE CALLS MUST HAVE AT LEAST 2 ARGUMENTS (ARRAY, ERROR)
7512 (IOI12) INFOS II INTERFACE INTERNAL CONSISTENCY ERROR
7513 (IOI13) F77 INTERFACE CALL MUST HAVE 'ENDLIST' TO MARK END OF ARGUMENT LIST
7514 (IOI14) INFOS II INTERFACE VERSION ERROR -- MUST USE LATEST INTERFACE

ICALL Errors (7700 - 7704)

7700 (IOTID) TASK ID CANNOT BE ZERO
7701 (IOIPR) INCOMPATIBLE PACKET AND RUNTIME VERSIONS
7702 (IOIPT) INVALID PACKET TYPE
7703 (IOBRD) BAD RUNTIME DATA (PROBABLY ICALL INTERNAL DATA IS INCORRECT)
7704 (IOIPP) ILLEGAL PACKET POSITION

Comments

7003 (IOINI) REVISION X.XX OF AOS INFOS II ERROR CODES

In this message, X.XX indicates the revision number of the INFOS II error messages that you have on your system. Check to be sure that you have the same revision of error codes as the revision of AOS INFOS II that you have on your system. If they aren't the same, load the correct version.

7005 (IOICE) INVALID CURRENT ENTRY

This error usually results when you attempt static relative access and the current position is not on a key. It often occurs when your current position is set above the main index or in front of a subindex.

7006 (IOTLV) WARNING - POSITIONED ABOVE MAIN INDEX (TOP LEVEL)

This warning often occurs when your current position is in the main index and you issue a command specifying upward relative motion. You will also get this warning when your current position is set above the index, as it is when the file is opened, and you request any relative motion other than Down or Down and Forward. Note that this is only a warning, and not an error. The INFOS II system will perform the positioning that you specified.

7011 (IOEST) END OF SUBINDEX

This message either means that your current position is on the lowest value key in an index and you have issued a relative backward command, or that your current position is on the highest value key in an index and you have issued a relative forward command. The INFOS II system cannot perform the positioning because there are no more keys in the specified direction.

7014 (IONDR) WARNING - DATA BASE RECORD NOT PRESENT

This warning indicates that the key to which you have gained access does not have a data record. Note that this is only a warning and not an error message; the INFOS II system performs the specified positioning and returns the appropriate data.

7030 (IOKPE) KEYED POSITIONING ERROR

This error simply means that the INFOS II system cannot find the key you have specified in a single level index file or the key in the key path you have specified in a multilevel file.

End of Appendix

Appendix B
INFOS II Parameter Files for the Scientific Languages

Figure B-1 contains the INFOS II interface parameter file for FORTRAN 5.

```
C       COPYRIGHT (C) DATA GENERAL CORPORATION 1977-1983
C       ALL RIGHTS RESERVED
C       LICENSED MATERIAL- PROPERTY OF DATA GENERAL CORPORATION


C               *****  IIPAR.FR   *****

C       INFOS II INTERFACE PARAMETER FILE FOR FORTRAN 5

C       REVISION 5.00

C       ************************************************************************

C               *****  INTERFACE VARIABLES   *****

        PARAMETER
     +  IIREC            =       160000K         ,
     +  IIKEY            =       160004K         ,
     +  IIKEY2           =       160404K         ,
     +  IIKEY3           =       161004K         ,
     +  IIKEY4           =       161404K         ,
     +  IIPREC           =       160010K         ,
     +
     +  IIRLEN           =        60014K         ,
     +  IIKLEN           =        60020K         ,
     +  IIKLEN2          =        60420K         ,
     +  IIKLEN3          =        61020K         ,
     +  IIKLEN4          =        61420K         ,
     +  IILRLEN          =        60024K         ,
     +  IIPRLEN          =        60074K         ,
     +  IIMRLEN          =        60070K         ,
     +  IIMKLEN          =        60072K         ,
     +
     +  IIAMODE          =        60030K         ,
     +  IIDMODE          =        60034K         ,
     +  IIKMODE          =        60050K         ,
     +  IIKMODE2         =        60450K         ,
     +  IIKMODE3         =        61050K         ,
     +  IIKMODE4         =        61450K         ,
     +  IILMODE          =        60040K         ,
     +  IIPMODE          =        60044K         ,
```

Figure B-1. INFOS II Interface Parameter File for FORTRAN 5 (continues)

```
+  IIRMODE        =       60144K      ,
+  IISMODE        =       60124K      ,
+
+  IILEV          =       60154K      ,
+  IIOCCH         =       60100K      ,
+  IIOCCH2        =       60500K      ,
+  IIOCCH3        =       61100K      ,
+  IIOCCH4        =       61500K      ,
+  IIOCCL         =       60102K      ,
+  IIOCCL2        =       60502K      ,
+  IIOCCL3        =       61102K      ,
+  IIOCCL4        =       61502K      ,
+  IIFDBKH        =       60120K      ,
+  IIFDBKL        =       60122K      ,
+  IIDUP          =       60076K      ,
+  IIDBDEL        =       60110K      ,
+  IIPRDEL        =       60112K      ,
+  IIOVFLO        =       60114K      ,
+  IISIP          =       60116K      ,
+
+  IIRMF          =       60212K      ,
+  IIVMF          =       60216K      ,
+
+  IINLEV         =       60054K      ,
+
+  IINLOX         =       60060K      ,
+  IIEXCLU        =       60064K      ,
+
+  IISALLOW       =       60140K      ,
+  IISMKLEN       =       60130K      ,
+  IISPRLEN       =       60134K      ,
+  IIRNS          =       60150K      ,
+
+  IIDNLEV        =       60172K      ,
+  IIDKEY         =      160156K      ,
+  IIDKEY2        =      160556K      ,
+  IIDKEY3        =      161156K      ,
+  IIDKEY4        =      161556K      ,
+  IIDKLEN        =       60162K      ,
+  IIDKLEN2       =       60562K      ,
+  IIDKLEN3       =       61162K      ,
+  IIDKLEN4       =       61562K      ,
+  IIDKMODE       =       60166K      ,
+  IIDKMODE2      =       60566K      ,
+  IIDKMODE3      =       61166K      ,
+  IIDKMODE4      =       61566K      ,
+  IIDPMODE       =       60206K      ,
+  IIDRMODE       =       60202K      ,
+  IIDSMODE       =       60176K
```

Figure B-1. INFOS II Interface Parameter File for FORTRAN 5 (continued)

```
C       ************************************************************************

C       VALUES FOR      IIAMODE


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  FULL ACC,
     +  SUPP DB,
     +  SUPP PR

        PARAMETER
     +  FULL ACC        =          0K        ,
     +  SUPP DB         =         20K        ,
     +  SUPP PR         =        100K        ,
     +  INVERT          =        220K        ,
     +  KEY ONLY        =        120K        ,
     +  LIM RLEN        =         40K        ,
     +  LIM SUP PR      =        140K

C       VALUES FOR      IIDMODE


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  REINST,
     +  RE PR,
     +  RE DB

        PARAMETER
     +  LOG DEL         =         16K        ,
     +  LD PR           =         14K        ,
     +  LD DB           =         12K        ,
     +  REINST          =          6K        ,
     +  RE PR           =          4K        ,
     +  RE DB           =          2K
```

Figure B-1. INFOS II Interface Parameter File for FORTRAN 5 (continued)

```
C       VALUES FOR      IILMODE


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  UNLOCK,
     +  UNL PR,
     +  UNL DB,
     +  REL LCK

        PARAMETER
     +  LCK             =          1006K          ,
     +  LCK PR          =          1004K          ,
     +  LCK DB          =          1002K          ,
     +  UNLOCK          =           406K          ,
     +  UNL PR          =           404K          ,
     +  UNL DB          =           402K          ,
     +  NO LOCKING      =             0K          ,
     +  REL LCK         =          1000K           ; FOR USE IN 'IRELE' ONLY

C       VALUES FOR      IIPMODE


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  SET POS,
     +  REL POS

        PARAMETER
     +  SET POS         =          2000K          ,
     +  NO CHANGE       =             0K          ,
     +  REL POS         =          2000K           ; FOR USE IN 'IRELE' ONLY

C       VALUES FOR      IIRMODE


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  FOR,
     +  BACK,
     +  DOWN,
     +  UP,
     +  DOWN FOR,
     +  UP FOR,
     +  UP BACK,
     +  STATIC
```

Figure B-1. INFOS II Interface Parameter File for FORTRAN 5 (continued)

```
      PARAMETER
     + FOR              =            0K          ,
     + BACK             =         4000K          ,
     + DOWN             =        10000K          ,
     + UP               =        30000K          ,
     + DOWN FOR         =        14000K          ,
     + UP FOR           =        20000K          ,
     + UP BACK          =        24000K          ,
     + STATIC           =        34000K
C       VALUES OF       IISMODE


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

      INTEGER
     + RELATIVE,
     + RELKEY

      PARAMETER
     + KEYED            =       100000K          ,
     + RELATIVE         =        40000K          ,
     + RELKEY           =       140000K

C       VALUES OF       IIKMODE , IIKMODE2 , ETC.


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

      INTEGER
     + EXACT,
     + DUP KEY,
     + GEN KEY,
     + APP KEY

      PARAMETER
     + EXACT            =            0K          ,
     + DUP KEY          =       100000K          ,
     + GEN KEY          =        40000K          ,
     + APP KEY          =        20000K
```

Figure B-1. INFOS II Interface Parameter File for FORTRAN 5 (continued)

```
C       VALUES OF       IISALLOW        -       FOR SUBINDEX DEFINITION


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  ALLOW SUB,
     +  ALLOW DUP,
     +  ALLOW NONE,
     +  ALLOW BOTH

        PARAMETER
     +  ALLOW SUB       =            0K          ,
     +  ALLOW DUP       =        44000K          ,
     +  ALLOW NONE      =        40000K          ,
     +  ALLOW BOTH      =         4000K

C       VALUES OF       IIEXCLU -       FOR EXCLUSIVE OPENS


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  EXC IND,
     +  EXC DB,
     +  EXC ALL

        PARAMETER
     +  EXC IND         =         2000K         ,
     +  EXC DB          =         1000K         ,
     +  EXC ALL         =         3000K

C       NEW (11/12/80) FOR 4.20  IIRDO  DECLARES INFOS II FILE OPEN
C       TO BE READONLY ACCESS

        PARAMETER
     +  IIRDO           =       60222K

C       VALUES OF       IIRDO -         FOR READ ONLY OPENS


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  READONLY,
     +  READWRITE

        PARAMETER
     +  READONLY        =       100K    ,
     +  READWRITE       =       0K
```

Figure B-1. INFOS II Interface Parameter File for FORTRAN 5 (continued)

```
C       NEW (11/12/80) FOR 4.20  IIFLEV  RETURNS MAXIMUM FILE LEVEL
C       IN THE IIGET COMMAND


        PARAMETER
     +  IIFLEV          =       60220K

C       USEFUL ERROR CODE NUMBERS ( F5 ADDS 3 TO SYSTEM ERROR CODES )

        PARAMETER
     +  IONORM  =    1          , ; NORMAL RETURN FROM SUBROUTINE
     +  IOIRM   = 3591          , ; ILLEGAL RELATIVE MOTION
     +  IOICE   = 3592          , ; INVALID CURRENT ENTRY
     +  IOTLV   = 3593          , ; TOP LEVEL ERROR
     +  IOSNA   = 3594          , ; SUBINDEXES NOT ALLOWED
     +  IOSNP   = 3595          , ; SUBINDEX NOT PRESENT
     +  IOESI   = 3596          , ; END OF SUBINDEX
     +  IOKAE   = 3598          , ; KEY ALREADY EXISTS
     +  IONDR   = 3599          , ; DATABASE RECORD NOT PRESENT
     +  IOSAE   = 3601          , ; SUBINDEX ALREADY EXISTS
     +  IOEHS   = 3604          , ; ENTRY HAS SUBINDEX, CAN NOT DELETE
     +  IOKPE   = 3611            ; KEYED POSITIONING ERROR
```

Figure B-1. INFOS II Interface Parameter File for FORTRAN 5 (concluded)

Figure B-2 shows the INFOS II interface parameter file for FORTRAN 77.

```
C       COPYRIGHT (C) DATA GENERAL CORPORATION 1977-1983
C       ALL RIGHTS RESERVED
C       LICENSED MATERIAL- PROPERTY OF DATA GENERAL CORPORATIION


C               *****  IIPAR.F77   *****

C       INFOS II INTERFACE PARAMETER FILE FOR FORTRAN 77

C       REVISION 5.00

C       ************************************************************************

        EXTERNAL        ENDLIST         !END OF ARGUMENTS MARKER

C               *****  INTERFACE VARIABLES   *****

        PARAMETER (
     +  IIREC            =       160000K         ,
     +  IIKEY            =       160004K         ,
     +  IIKEY2           =       160404K         ,
     +  IIKEY3           =       161004K         ,
     +  IIKEY4           =       161404K         ,
     +  IIPREC           =       160010K         ,
     +
     +  IIRLEN           =        60014K         ,
     +  IIKLEN           =        60020K         ,
     +  IIKLEN2          =        60420K         ,
     +  IIKLEN3          =        61020K         ,
     +  IIKLEN4          =        61420K         ,
     +  IILRLEN          =        60024K         ,
     +  IIPRLEN          =        60074K         ,
     +  IIMRLEN          =        60070K         ,
     +  IIMKLEN          =        60072K         ,
     +
     +  IIAMODE          =        60030K         ,
     +  IIDMODE          =        60034K         ,
     +  IIKMODE          =        60050K         ,
     +  IIKMODE2         =        60450K         ,
     +  IIKMODE3         =        61050K         ,
     +  IIKMODE4         =        61450K         ,
     +  IILMODE          =        60040K         ,
     +  IIPMODE          =        60044K         ,
     +  IIRMODE          =        60144K         ,
     +  IISMODE          =        60124K         ,
     +
```

Figure B-2. INFOS II Interface Parameter File for FORTRAN 77 (continues)

```
+  IILEV          =        60154K      ,
+  IIOCCH         =        60100K      ,
+  IIOCCH2        =        60500K      ,
+  IIOCCH3        =        61100K      ,
+  IIOCCH4        =        61500K      ,
+  IIOCCL         =        60102K      ,
+  IIOCCL2        =        60502K      ,
+  IIOCCL3        =        61102K      ,
+  IIOCCL4        =        61502K      ,
+  IIFDBKH        =        60120K      ,
+  IIFDBKL        =        60122K      ,
+  IIDUP          =        60076K      ,
+  IIDBDEL        =        60110K      ,
+  IIPRDEL        =        60112K      ,
+  IIOVFLO        =        60114K      ,
+  IISIP          =        60116K      ,
+
+  IIRMF          =        60212K      ,
+  IIVMF          =        60216K      ,
+
+  IINLEV         =        60054K      ,
+
+  IINLOX         =        60060K      ,
+  IIEXCLU        =        60064K      ,
+
+  IISALLOW       =        60140K      ,
+  IISMKLEN       =        60130K      ,
+  IISPRLEN       =        60134K      ,
+  IIRNS          =        60150K      ,
+
+  IIDNLEV        =        60172K      ,
+  IIDKEY         =       160156K      ,
+  IIDKEY2        =       160556K      ,
+  IIDKEY3        =       161156K      ,
+  IIDKEY4        =       161556K      ,
+  IIDKLEN        =        60162K      ,
+  IIDKLEN2       =        60562K      ,
+  IIDKLEN3       =        61162K      ,
+  IIDKLEN4       =        61562K      ,
+  IIDKMODE       =        60166K      ,
+  IIDKMODE2      =        60566K      ,
+  IIDKMODE3      =        61166K      ,
+  IIDKMODE4      =        61566K      ,
+  IIDPMODE       =        60206K      ,
+  IIDRMODE       =        60202K      ,
+  IIDSMODE       =        60176K
+ )
```

Figure B-2. INFOS II Interface Parameter File for FORTRAN 77 (continued)

```
C       ************************************************************************

C       VALUES FOR      IIAMODE


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  FULL ACC,
     +  SUPP DB,
     +  SUPP PR

        PARAMETER (
     +  FULL ACC        =          0K        ,
     +  SUPP DB         =         20K        ,
     +  SUPP PR         =        100K        ,
     +  INVERT          =        220K        ,
     +  KEY ONLY        =        120K        ,
     +  LIM RLEN        =         40K        ,
     +  LIM SUP PR      =        140K
     + )

C       VALUES FOR      IIDMODE


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  REINST,
     +  RE PR,
     +  RE DB

        PARAMETER (
     +  LOG DEL         =          16K        ,
     +  LD PR           =          14K        ,
     +  LD DB           =          12K        ,
     +  REINST          =           6K        ,
     +  RE PR           =           4K        ,
     +  RE DB           =           2K
     + )

C       VALUES FOR      IILMODE


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  UNLOCK,
     +  UNL PR,
     +  UNL DB,
     +  REL LCK
```

Figure B-2. INFOS II Interface Parameter File for FORTRAN 77 (continued)

```
      PARAMETER (
     + LCK             =         1006K         ,
     + LCK PR          =         1004K         ,
     + LCK DB          =         1002K         ,
     + UNLOCK          =          406K         ,
     + UNL PR          =          404K         ,
     + UNL DB          =          402K         ,
     + NO LOCKING      =            0K         ,
     + REL LCK         =         1000K          ! FOR USE IN 'IRELE' ONLY
     + )

C       VALUES FOR      IIPMODE


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     + SET POS,
     + REL POS

        PARAMETER (
     + SET POS         =         2000K         ,
     + NO CHANGE       =            0K         ,
     + REL POS         =         2000K          ! FOR USE IN 'IRELE' ONLY
     + )

C       VALUES FOR      IIRMODE


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     + FOR,
     + BACK,
     + DOWN,
     + UP,
     + DOWN FOR,
     + UP FOR,
     + UP BACK,
     + STATIC

        PARAMETER (
     + FOR             =            0K         ,
     + BACK            =         4000K         ,
     + DOWN            =        10000K         ,
     + UP              =        30000K         ,
     + DOWN FOR        =        14000K         ,
     + UP FOR          =        20000K         ,
     + UP BACK         =        24000K         ,
     + STATIC          =        34000K
     + )
```

Figure B-2. INFOS II Interface Parameter File for FORTRAN 77 (continued)

```
C       VALUES OF       IISMODE

C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  RELATIVE,
     +  RELKEY

        PARAMETER (
     +  KEYED          =       100000K         ,
     +  RELATIVE       =        40000K         ,
     +  RELKEY         =       140000K
     + )

C       VALUES OF       IIKMODE , IIKMODE2 , ETC.


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  EXACT,
     +  DUP KEY,
     +  GEN KEY,
     +  APP KEY

        PARAMETER (
     +  EXACT          =           0K          ,
     +  DUP KEY        =       100000K         ,
     +  GEN KEY        =        40000K         ,
     +  APP KEY        =        20000K
     + )

C       VALUES OF       IISALLOW        -       FOR SUBINDEX DEFINITION


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  ALLOW SUB,
     +  ALLOW DUP,
     +  ALLOW NONE,
     +  ALLOW BOTH

        PARAMETER (
     +  ALLOW SUB      =           0K          ,
     +  ALLOW DUP      =       44000K          ,
     +  ALLOW NONE     =       40000K          ,
     +  ALLOW BOTH     =        4000K
     + )
```

Figure B-2. INFOS II Interface Parameter File for FORTRAN 77 (continued)

```
C       VALUES OF       IIEXCLU -       FOR EXCLUSIVE OPENS

C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  EXC IND,
     +  EXC DB,
     +  EXC ALL

        PARAMETER (
     +  EXC IND         =         2000K         ,
     +  EXC DB          =         1000K         ,
     +  EXC ALL         =         3000K
     + )

C       NEW (11/12/80) FOR 4.20  IIRDO  DECLARES INFOS II FILE OPEN
C       TO BE READONLY ACCESS

        PARAMETER (
     +  IIRDO           =       60222K
     + )

C       VALUES OF       IIRDO -         FOR READ ONLY OPENS


C       BY DECLARING THESE PARAMETERS INTEGER, IMPLICIT STATEMENT IS
C       NOT NEEDED.

        INTEGER
     +  READONLY,
     +  READWRITE

        PARAMETER (
     +  READONLY        =       100K    ,
     +  READWRITE       =       0K
     + )

C       NEW (11/12/80) FOR 4.20  IIFLEV  RETURNS MAXIMUM FILE LEVEL
C       IN THE IIGET COMMAND

        PARAMETER (
     +  IIFLEV          =       60220K
     + )
```

Figure B-2. INFOS II Interface Parameter File for FORTRAN 77 (continued)

```
C       USEFUL ERROR CODES

        PARAMETER (
     +  IONORM  =    0          , ! NORMAL RETURN FROM SUBROUTINE
     +  IOIRM   = 3588          , ! ILLEGAL RELATIVE MOTION
     +  IOICE   = 3589          , ! INVALID CURRENT ENTRY
     +  IOTLV   = 3590          , ! TOP LEVEL ERROR
     +  IOSNA   = 3591          , ! SUBINDEXES NOT ALLOWED
     +  IOSNP   = 3592          , ! SUBINDEX NOT PRESENT
     +  IOESI   = 3593          , ! END OF SUBINDEX
     +  IOKAE   = 3595          , ! KEY ALREADY EXISTS
     +  IONDR   = 3596          , ! DATABASE RECORD NOT PRESENT
     +  IOSAE   = 3598          , ! SUBINDEX ALREADY EXISTS
     +  IOEHS   = 3601          , ! ENTRY HAS SUBINDEX, CAN NOT DELETE
     +  IOKPE   = 3608            ! KEYED POSITIONING ERROR
     + )
```

Figure B-2. INFOS II Interface Parameter File for FORTRAN 77 (concluded)

Figure B-3 contains the INFOS II interface parameter file for the DG/L language.

```
/*      COPYRIGHT (C) DATA GENERAL CORPORATION 1977-1983
        ALL RIGHTS RESERVED
        LICENSED MATERIAL- PROPERTY OF DATA GENERAL CORPORATION        */


%             *****  IIPAR.DG   *****

%       INFOS INTERFACE PARAMETER FILE FOR DG/L

%       REVISION 5.00

%       ****************************************************************

%             *****  INTERFACE VARIABLES   *****

        LITERAL
        IIREC           ( 160000R8P1 )  ,
        IIKEY           ( 160004R8P1 )  ,
        IIKEY2          ( 160404R8P1 )  ,
        IIKEY3          ( 161004R8P1 )  ,
        IIKEY4          ( 161404R8P1 )  ,
        IIPREC          ( 160010R8P1 )  ,

        IIRLEN          (  60014R8P1 )  ,
        IIKLEN          (  60020R8P1 )  ,
        IIKLEN2         (  60420R8P1 )  ,
        IIKLEN3         (  61020R8P1 )  ,
        IIKLEN4         (  61420R8P1 )  ,
        IILRLEN         (  60024R8P1 )  ,
        IIPRLEN         (  60074R8P1 )  ,
        IIMRLEN         (  60070R8P1 )  ,
        IIMKLEN         (  60072R8P1 )  ,

        IIAMODE         (  60030R8P1 )  ,
        IIDMODE         (  60034R8P1 )  ,
        IIKMODE         (  60050R8P1 )  ,
        IIKMODE2        (  60450R8P1 )  ,
        IIKMODE3        (  61050R8P1 )  ,
        IIKMODE4        (  61450R8P1 )  ,
        IILMODE         (  60040R8P1 )  ,
        IIPMODE         (  60044R8P1 )  ,
        IIRMODE         (  60144R8P1 )  ,
        IISMODE         (  60124R8P1 )  ,
```

Figure B-3. INFOS II Interface Parameter File for the DG/L language (continues)

```
        IILEV           (  60154R8P1 )  ,
        IIOCCH          (  60100R8P1 )  ,
        IIOCCH2         (  60500R8P1 )  ,
        IIOCCH3         (  61100R8P1 )  ,
        IIOCCH4         (  61500R8P1 )  ,
        IIOCCL          (  60102R8P1 )  ,
        IIOCCL2         (  60502R8P1 )  ,
        IIOCCL3         (  61102R8P1 )  ,
        IIOCCL4         (  61502R8P1 )  ,
        IIFDBKH         (  60120R8P1 )  ,
        IIFDBKL         (  60122R8P1 )  ,
        IIDUP           (  60076R8P1 )  ,
        IIDBDEL         (  60110R8P1 )  ,
        IIPRDEL         (  60112R8P1 )  ,
        IIOVFLO         (  60114R8P1 )  ,
        IISIP           (  60116R8P1 )  ,

        IIRMF           (  60212R8P1 )  ,
        IIVMF           (  60216R8P1 )  ,

        IINLEV          (  60054R8P1 )  ,

        IINLOX          (  60060R8P1 )  ,
        IIEXCLU         (  60064R8P1 )  ,

        IISALLOW        (  60140R8P1 )  ,
        IISMKLEN        (  60130R8P1 )  ,
        IISPRLEN        (  60134R8P1 )  ,
        IIRNS           (  60150R8P1 )  ,

        IIDKEY          ( 160156R8P1 )  ,
        IIDKEY2         ( 160556R8P1 )  ,
        IIDKEY3         ( 161156R8P1 )  ,
        IIDKEY4         ( 161556R8P1 )  ,
        IIDKLEN         (  60162R8P1 )  ,
        IIDKLEN2        (  60562R8P1 )  ,
        IIDKLEN3        (  61162R8P1 )  ,
        IIDKLEN4        (  61562R8P1 )  ,
        IIDKMODE        (  60166R8P1 )  ,
        IIDKMODE2       (  60566R8P1 )  ,
        IIDKMODE3       (  61166R8P1 )  ,
        IIDKMODE4       (  61566R8P1 )  ,
        IIDNLEV         (  60172R8P1 )  ,
        IIDPMODE        (  60206R8P1 )  ,
        IIDRMODE        (  60202R8P1 )  ,
        IIDSMODE        (  60176R8P1 )      ;

%       ****************************************************************
```

Figure B-3. INFOS II Interface Parameter File for the DG/L language (continued)

```
%       VALUES FOR      IIAMODE

        LITERAL
        FULL_ACC        (      0R8P1 )  ,
        SUPP_DB         (     20R8P1 )  ,
        SUPP_PR         (    100R8P1 )  ,
        INVERT          (    220R8P1 )  ,
        KEY_ONLY        (    120R8P1 )  ,
        LIM_RLEN        (     40R8P1 )  ,
        LIM_SUP_PR      (    140R8P1 )          ;

%       VALUES FOR      IIDMODE

        LITERAL
        LOG_DEL         (     16R8P1 )  ,
        LD_PR           (     14R8P1 )  ,
        LD_DB           (     12R8P1 )  ,
        REINST          (      6R8P1 )  ,
        RE_PR           (      4R8P1 )  ,
        RE_DB           (      2R8P1 )          ;

%       VALUES FOR      IILMODE

        LITERAL
        LCK             (   1006R8P1 )  ,
        LCK_PR          (   1004R8P1 )  ,
        LCK_DB          (   1002R8P1 )  ,
        UNLOCK          (    406R8P1 )  ,
        UNL_PR          (    404R8P1 )  ,
        UNL_DB          (    402R8P1 )  ,
        NO_LOCKING      (      0R8P1 )  ,
        REL_LCK         (   1000R8P1 )   ; % FOR USE IN 'IRELEASE'

%       VALUES FOR      IIPMODE

        LITERAL
        SET_POS         (   2000R8P1 )  ,
        NO_CHANGE       (      0R8P1 )  ,
        REL_POS         (   2000R8P1 )   ; % FOR USE IN 'IRELEASE'

%       VALUES FOR      IIRMODE

        LITERAL
        FORWARD         (      0R8P1 )  ,
        BACK            (   4000R8P1 )  ,
        DOWN            (  10000R8P1 )  ,
        UP              (  30000R8P1 )  ,
        DOWN_FOR        (  14000R8P1 )  ,
        UP_FOR          (  20000R8P1 )  ,
        UP_BACK         (  24000R8P1 )  ,
        STATIC          (  34000R8P1 )          ;
```

Figure B-3. INFOS II Interface Parameter File for the DG/L language (continued)

```
%       VALUES OF       IISMODE

        LITERAL
        KEYED           ( 100000R8P1 )  ,
        RELATIVE        (  40000R8P1 )  ,
        RELKEY          ( 140000R8P1 )          ;

%       VALUES OF       IIKMODE , IIKMODE2 , ETC.

        LITERAL
        EXACT           (      0R8P1 )  ,
        DUP_KEY         ( 100000R8P1 )  ,
        GEN_KEY         (  40000R8P1 )  ,
        APP_KEY         (  20000R8P1 )          ;

%       VALUES OF       IISALLOW        -       FOR SUBINDEX DEFINITION

        LITERAL
        ALLOW_SUB       (      0R8P1 )  ,
        ALLOW_DUP       (  44000R8P1 )  ,
        ALLOW_NONE      (  40000R8P1 )  ,
        ALLOW_BOTH      (   4000R8P1 )          ;

%       VALUES OF       IIEXCLU -       FOR EXCLUSIVE OPENS

        LITERAL
        EXC_IND         (   2000R8P1 )  ,
        EXC_DB          (   1000R8P1 )  ,
        EXC_ALL         (   3000R8P1 )          ;

%       NEW (11/12/80) FOR 4.20  IIRDO  DECLARES INFOS II FILE OPEN
%       TO BE READONLY ACCESS

        LITERAL
        IIRDO           (60222R8P1)     ;

%       VALUES OF       IIRDO -         FOR READ ONLY OPENS

        LITERAL
        READONLY        (100R8P1)       ,
        READWRITE       (0R8P1)         ;

%       NEW (11/12/80) FOR 4.20  IIFLEV  RETURNS MAXIMUM FILE LEVEL
%       IN THE  IIGET COMMAND

        LITERAL
        IIFLEV          (60220R8P1)     ;
```

Figure B-3. INFOS II Interface Parameter File for the DG/L language (continued)

```
%       USEFUL ERROR CODE NUMBERS ( INTERFACE ADDS 3 TO SYSTEM CODES )

        LITERAL
        IONORM  (    1 )        , % NORMAL RETURN FROM SUBROUTINE
        IOIRM   ( 3591 )        , % ILLEGAL RELATIVE MOTION
        IOICE   ( 3592 )        , % INVALID CURRENT ENTRY
        IOTLV   ( 3593 )        , % TOP LEVEL ERROR
        IOSNA   ( 3594 )        , % SUBINDEXES NOT ALLOWED
        IOSNP   ( 3595 )        , % SUBINDEX NOT PRESENT
        IOESI   ( 3596 )        , % END OF SUBINDEX
        IOKAE   ( 3598 )        , % KEY ALREADY EXISTS
        IONDR   ( 3599 )        , % DATABASE RECORD NOT PRESENT
        IOSAE   ( 3601 )        , % SUBINDEX ALREADY EXISTS
        IOEHS   ( 3604 )        , % SUBINDEX EXISTS , CAN NOT DELETE
        IOKPE   ( 3611 )         ;% KEYED POSITIONING ERROR
```

Figure B-3. INFOS II Interface Parameter File for the DG/L language (concluded)

Figure B-4 shows the INFOS II interface parameter file for PL/I.

```
/*      COPYRIGHT (C) DATA GENERAL CORPORATION 1977-1983
        ALL RIGHTS RESERVED
        LICENSED MATERIAL- PROPERTY OF DATA GENERAL CORPORATION          */


/*              *****  IIPAR.PL1   *****                                 */

/*      INFOS INTERFACE PARAMETER FILE FOR PL/I                          */

/*      REVISION 5.00                                                    */

/*      ******************************************************************/

/*              *****  INTERFACE VARIABLES   *****                       */

%REPLACE    IIREC               BY "E000"B4     ;
%REPLACE    IIKEY               BY "E004"B4     ;
%REPLACE    IIKEY2              BY "E104"B4     ;
%REPLACE    IIKEY3              BY "E204"B4     ;
%REPLACE    IIKEY4              BY "E304"B4     ;
%REPLACE    IIPREC              BY "E008"B4     ;

%REPLACE    IIRLEN              BY "600C"B4     ;
%REPLACE    IIKLEN              BY "6010"B4     ;
%REPLACE    IIKLEN2             BY "6110"B4     ;
%REPLACE    IIKLEN3             BY "6210"B4     ;
%REPLACE    IIKLEN4             BY "6310"B4     ;
%REPLACE    IILRLEN             BY "6014"B4     ;
%REPLACE    IIPRLEN             BY "603C"B4     ;
%REPLACE    IIMRLEN             BY "6038"B4     ;
%REPLACE    IIMKLEN             BY "603A"B4     ;

%REPLACE    IIAMODE             BY "6018"B4     ;
%REPLACE    IIDMODE             BY "601C"B4     ;
%REPLACE    IIKMODE             BY "6028"B4     ;
%REPLACE    IIKMODE2            BY "6128"B4     ;
%REPLACE    IIKMODE3            BY "6228"B4     ;
%REPLACE    IIKMODE4            BY "6328"B4     ;
%REPLACE    IILMODE             BY "6020"B4     ;
%REPLACE    IIPMODE             BY "6024"B4     ;
%REPLACE    IIRMODE             BY "6064"B4     ;
%REPLACE    IISMODE             BY "6054"B4     ;
```

Figure B-4. INFOS II Interface Parameter File for PL/I (continues)

```
%REPLACE    IILEV              BY "606C"B4    ;
%REPLACE    IIOCCH             BY "6040"B4    ;
%REPLACE    IIOCCH2            BY "6140"B4    ;
%REPLACE    IIOCCH3            BY "6240"B4    ;
%REPLACE    IIOCCH4            BY "6340"B4    ;
%REPLACE    IIOCCL             BY "6042"B4    ;
%REPLACE    IIOCCL2            BY "6142"B4    ;
%REPLACE    IIOCCL3            BY "6242"B4    ;
%REPLACE    IIOCCL4            BY "6342"B4    ;
%REPLACE    IIFDBKH            BY "6050"B4    ;
%REPLACE    IIFDBKL            BY "6052"B4    ;
%REPLACE    IIDUP              BY "603E"B4    ;
%REPLACE    IIDBDEL            BY "6048"B4    ;
%REPLACE    IIPRDEL            BY "604A"B4    ;
%REPLACE    IIOVFLO            BY "604C"B4    ;
%REPLACE    IISIP              BY "604E"B4    ;

%REPLACE    IIRMF              BY "608A"B4    ;
%REPLACE    IIVMF              BY "608E"B4    ;

%REPLACE    IINLEV             BY "602C"B4    ;

%REPLACE    IINLOX             BY "6030"B4    ;
%REPLACE    IIEXCLU            BY "6034"B4    ;

%REPLACE    IISALLOW           BY "6060"B4    ;
%REPLACE    IISMKLEN           BY "6058"B4    ;
%REPLACE    IISPRLEN           BY "605C"B4    ;
%REPLACE    IIRNS              BY "6068"B4    ;

%REPLACE    IIDKEY             BY "E06E"B4    ;
%REPLACE    IIDKEY2            BY "E16E"B4    ;
%REPLACE    IIDKEY3            BY "E26E"B4    ;
%REPLACE    IIDKEY4            BY "E36E"B4    ;
%REPLACE    IIDKLEN            BY "6072"B4    ;
%REPLACE    IIDKLEN2           BY "6172"B4    ;
%REPLACE    IIDKLEN3           BY "6272"B4    ;
%REPLACE    IIDKLEN4           BY "6372"B4    ;
%REPLACE    IIDKMODE           BY "6076"B4    ;
%REPLACE    IIDKMODE2          BY "6176"B4    ;
%REPLACE    IIDKMODE3          BY "6276"B4    ;
%REPLACE    IIDKMODE4          BY "6376"B4    ;
%REPLACE    IIDNLEV            BY "607A"B4    ;
%REPLACE    IIDPMODE           BY "6086"B4    ;
%REPLACE    IIDRMODE           BY "6082"B4    ;
%REPLACE    IIDSMODE           BY "607E"B4    ;
```

Figure B-4. INFOS II Interface Parameter File for PL/I (continued)

```
/*      ****************************************************************/

/* VALUES FOR IIAMODE              */

%REPLACE    INF_FULL_ACC        BY "0000"B4     ;
%REPLACE    INF_SUPP_DB         BY "0010"B4     ;
%REPLACE    INF_SUPP_PR         BY "0040"B4     ;
%REPLACE    INF_INVERT          BY "0090"B4     ;
%REPLACE    INF_KEY_ONLY        BY "0050"B4     ;
%REPLACE    INF_LIM_RLEN        BY "0020"B4     ;
%REPLACE    INF_LIM_SUP_PR      BY "0060"B4     ;

/* VALUES FOR IIDMODE              */

%REPLACE    INF_LOG_DEL         BY "000E"B4     ;
%REPLACE    INF_LD_PR           BY "000C"B4     ;
%REPLACE    INF_LD_DB           BY "000A"B4     ;
%REPLACE    INF_REINST          BY "0006"B4     ;
%REPLACE    INF_RE_PR           BY "0004"B4     ;
%REPLACE    INF_RE_DB           BY "0002"B4     ;

/* VALUES FOR IILMODE              */

%REPLACE    INF_LCK             BY "0206"B4     ;
%REPLACE    INF_LCK_PR          BY "0204"B4     ;
%REPLACE    INF_LCK_DB          BY "0202"B4     ;
%REPLACE    INF_UNLOCK          BY "0106"B4     ;
%REPLACE    INF_UNL_PR          BY "0104"B4     ;
%REPLACE    INF_UNL_DB          BY "0102"B4     ;
%REPLACE    INF_NO_LOCKING      BY "0000"B4     ;
%REPLACE    INF_REL_LCK         BY "0200"B4     ; /* FOR USE IN 'IRELEASE' */

/* VALUES FOR IIPMODE              */

%REPLACE    INF_SET_POS         BY "0400"B4     ;
%REPLACE    INF_NO_CHANGE       BY "0000"B4     ;
%REPLACE    INF_REL_POS         BY "0400"B4     ; /* FOR USE IN 'IRELEASE' */

/* VALUES FOR IIRMODE              */

%REPLACE    INF_FORWARD         BY "0000"B4     ;
%REPLACE    INF_BACK            BY "0800"B4     ;
%REPLACE    INF_DOWN            BY "1000"B4     ;
%REPLACE    INF_UP              BY "3000"B4     ;
%REPLACE    INF_DOWN_FOR        BY "1800"B4     ;
%REPLACE    INF_UP_FOR          BY "2000"B4     ;
%REPLACE    INF_UP_BACK         BY "2800"B4     ;
%REPLACE    INF_STATIC          BY "3800"B4     ;
```

Figure B-4. INFOS II Interface Parameter File for PL/I (continued)

```
/* VALUES OF IISMODE                */

%REPLACE    INF_KEYED           BY "8000"B4     ;
%REPLACE    INF_RELATIVE        BY "4000"B4     ;
%REPLACE    INF_RELKEY          BY "C000"B4     ;

/* VALUES OF IIKMODE , IIKMODE2 , ETC.      */

%REPLACE    INF_EXACT           BY "0000"B4     ;
%REPLACE    INF_DUP_KEY         BY "8000"B4     ;
%REPLACE    INF_GEN_KEY         BY "4000"B4     ;
%REPLACE    INF_APP_KEY         BY "2000"B4     ;

/* VALUES OF IISALLOW - FOR SUBINDEX DEFINITION */

%REPLACE    INF_ALLOW_SUB       BY "0000"B4     ;
%REPLACE    INF_ALLOW_DUP       BY "4800"B4     ;
%REPLACE    INF_ALLOW_NONE      BY "4000"B4     ;
%REPLACE    INF_ALLOW_BOTH      BY "0800"B4     ;

/* VALUES OF IIEXCLU - FOR EXCLUSIVE OPENS   */

%REPLACE    INF_EXC_IND         BY "0400"B4     ;
%REPLACE    INF_EXC_DB          BY "0200"B4     ;
%REPLACE    INF_EXC_ALL         BY "0600"B4     ;

/* NEW (11/12/80) FOR 4.20  IIRDO  DECLARES INFOS II FILE OPEN */
/* TO BE READONLY ACCESS   */

%REPLACE       IIRDO   BY      "6092"B4        ;

/* VALUES OF IIRDO    */

%REPLACE     INF_READONLY      BY      "0040"B4        ;
%REPLACE     INF_READWRITE     BY      "0000"B4        ;

/* NEW (11/12/80) FOR 4.20  IIFLEV  RETURNS MAXIMUM FILE LEVEL   */
/* IN THE  IIGET COMMAND   */

%REPLACE     IIFLEV             BY "6090"B4      ;
```

Figure B-4. INFOS II Interface Parameter File for PL/I (continued)

```
/*      USEFUL ERROR CODE NUMBERS */

%REPLACE    IONORM     BY    0 ; /* NORMAL RETURN FROM SUBROUTINE      */
%REPLACE    IOIRM      BY 3588 ; /* ILLEGAL RELATIVE MOTION            */
%REPLACE    IOICE      BY 3589 ; /* INVALID CURRENT ENTRY              */
%REPLACE    IOTLV      BY 3590 ; /* TOP LEVEL ERROR                    */
%REPLACE    IOSNA      BY 3591 ; /* SUBINDEXES NOT ALLOWED             */
%REPLACE    IOSNP      BY 3592 ; /* SUBINDEX NOT PRESENT               */
%REPLACE    IOESI      BY 3593 ; /* END OF SUBINDEX                    */
%REPLACE    IOKAE      BY 3595 ; /* KEY ALREADY EXISTS                 */
%REPLACE    IONDR      BY 3596 ; /* DATABASE RECORD NOT PRESENT        */
%REPLACE    IOSAE      BY 3598 ; /* SUBINDEX ALREADY EXISTS            */
%REPLACE    IOEHS      BY 3601 ; /* SUBINDEX EXIST - CAN NOT DELETE    */
%REPLACE    IOKPE      BY 3608 ; /* KEYED POSITIONING ERROR            */


DECLARE (IOPEN,
        ICLOS,
        IISET,
        IIGET,
        IKEYR,
        IRELR,
        IHIRE,
        IWRIT,
        IREWR,
        IDELE,
        ILDEL,
        ISDEF,
        ISDEL,
        ISLIN,
        IRELE)  ENTRY OPTIONS ( VARIABLE ) ;
```

Figure B-4. INFOS II Interface Parameter File for PL/I (concluded)

End of Appendix

Appendix C
Sample INFOS II Interface Programs for the Scientific Languages

Figures C-1 and C-2 contain sample FORTRAN 5 programs that use the INFOS II interface.

```
C-------------------------------------------------------------------------C
C                                                                         C
C          Sample FORTRAN 5 Program Using the INFOS II Interface          C
C                                                                         C
C       This program uses the FORTRAN 5 INFOS II interface to build a     C
C       test file.  Note that it is not an exhaustive demonstration       C
C       of the INFOS II interface.                                        C
C                                                                         C
C       Also note the techniques we use here to manipulate character data C
C       in FORTRAN 5.  We store characters in integer arrays and manipulate C
C       them with the BYTE function and other user-written subroutines    C
C       like BYTEMOVER.                                                   C
C                                                                         C
C       You should also note how we use the IER variable to handle errors. C
C                                                                         C
C-------------------------------------------------------------------------C

        INCLUDE "IIPAR.FR"

        INTEGER
     +  IARRAY  ( 56 )          ,  ; Interface array for 2 index levels.
     +
     +  KEY     ( 10 )          ,
     +  RECORD  ( 15 )          ,
     +
     +  SCRATCH ( 40 )             ; Scratch storage area.
```

Figure C-1. Sample FORTRAN 5 Program Using the INFOS II Interface – I (continues)

```
C-------------------------------------------------------------------------C
C                                                                         C
C       First we must open the INFOS II file, allowing two index levels.  C
C       Note that we have made IARRAY big enough for two levels.          C
C       After every INFOS operation, you should compare IER with IONORM   C
C       ( IONORM is defined in IIPAR.FR ).  If they are not equal, the    C
C       program should report the error.                                  C
C                                                                         C
C-------------------------------------------------------------------------C
        CALL IOPEN (    IARRAY ,
     +                  "INFOS_TEST_FILE" , 2 ,
     +                  IER     )
        IPT = 100
        WRITE ( 10 , 999 )  IPT , IER
        IF ( IER .NE. IONORM )  GO TO 950



C-------------------------------------------------------------------------C
C                                                                         C
C       Write five keys in the top-level index, and their records into the C
C       database:                                                         C
C                                                                         C
C               KEY: A          RECORD: AA                                C
C               KEY: AA         RECORD: CCDD                              C
C               KEY: AAA        RECORD: EEFFGG                            C
C               KEY: AAAA       RECORD: GGHHIIJJ                          C
C               KEY: AAAAA      RECORD: IIJJKKLLMM                        C
C                                                                         C
C-------------------------------------------------------------------------C

        CALL BYTEMOVER ( KEY , 1 , "AAAAA" , 1 , 5 )
        CALL BYTEMOVER ( RECORD , 1 , "AABBCCDDEEFFGGHHIIJJKKLLMMNNOO" ,1 ,30 )

        DO 200 I = 1 , 5

        CALL IWRITE (   IARRAY ,
     +                  IIKEY , KEY ,
     +                  IIKLEN , I ,
     +                  IIREC , RECORD ( 1 + 2*(I-1) ) ,
     +                  IIRLEN , 2 * I ,
     +                  IER     )
        IPT = 200 + I
        WRITE ( 10 , 999 )  IPT , IER
        IF ( IER .NE. IONORM )  GO TO 950

200     CONTINUE
```

Figure C-1. Sample FORTRAN 5 Program Using the INFOS II Interface – I (continued)

```
C------------------------------------------------------------------------------C
C                                                                              C
C       Do a generic keyed search, suppressing database access, to find out    C
C       if there is a key that starts with a 'B'.                              C
C                                                                              C
C------------------------------------------------------------------------------C
        BYTE ( SCRATCH , 1 )  =  1RB

        CALL IKEYREAD ( IARRAY ,
     +                  IIKEY , SCRATCH ,
     +                  IIKLEN , 1 ,
     +                  IIKMODE , GEN KEY ,
     +                  IIAMODE , SUPP DB ,
     +                  IER      )
        IPT = 300
        WRITE ( 10 , 999 )  IPT , IER
        IF (( IER .NE. IOKPE ) .AND. ( IER .NE. IONORM ))  GO TO 950
        IF ( IER .EQ. IOKPE )  WRITE FREE ( 10 ) "NO KEY STARTS WITH B"

C------------------------------------------------------------------------------C
C                                                                              C
C       Create a subindex under KEY: AA.                                       C
C       Then enter KEY: BBB in that subindex with no database record           C
C       and with PARTIAL RECORD: 12345.                                        C
C                                                                              C
C------------------------------------------------------------------------------C

        CALL ISDEFINE ( IARRAY ,
     +                  IIKEY , "AA" ,
     +                  IIKLEN , 2 ,
     +                  IISPRLEN , 5 ,
     +                  IER       )
        IPT = 400
        WRITE ( 10 , 999 )  IPT , IER
        IF ( IER .NE. IONORM )  GO TO 950

        CALL IWRITE (   IARRAY ,
     +                  IINLEV , 2 ,
     +                  IIAMODE , SUPP DB ,
     +                  IIKEY , "AA" ,
     +                  IIKLEN , 2 ,
     +                  IIKEY2 , "BBB" ,
     +                  IIKLEN2 , 3 ,
     +                  IIPREC , "12345" ,
     +                  IER     )
        IPT = 500
        WRITE ( 10 , 999 )  IPT , IER
        IF ( IER .NE. IONORM )  GO TO 950
```

Figure C-1. Sample FORTRAN 5 Program Using the INFOS II Interface – I (continued)

```
C------------------------------------------------------------------------C
C                                                                        C
C       You should always close an INFOS file when you are finished      C
C       processing.                                                      C
C                                                                        C
C------------------------------------------------------------------------C
900     CALL ICLOSE ( IARRAY , IER )
        IPT     =       900
        WRITE ( 10 , 999 )  IPT , IER

        CALL EXIT

C------------------------------------------------------------------------C
C                                                                        C
C       Set up an error-handler for unexpected INFOS II error code returns.   C
C                                                                        C
C------------------------------------------------------------------------C

950     WRITE ( 10 , 951 )  IER , IPT
951     FORMAT  ( // , "***** ABNORMAL ERROR RETURN *****" , / ,
     +            "RETURNED ERROR CODE = " , I7 , / ,
     +            " AT POINT " , I6 , / ,
     +            "***** WILL CLOSE ALL FILES AND TERMINATE PROGRAM *****" , / )

        GO TO 900

999     FORMAT ( "OPERATION AT POINT " , I5 , " RESULTED IN IER = " , I6 )

        END
```

Figure C-1. Sample FORTRAN 5 Program Using the INFOS II Interface – I (concluded)

```
C-------------------------------------------------------------------------C
C                                                                         C
C              Sample FORTRAN 5 Program Using the INFOS II Interface      C
C                                                                         C
C       This program uses the FORTRAN 5 INFOS II interface to display a test  C
C       file.  Note that it is not an exhaustive demonstration of the INFOS   C
C       interface.                                                        C
C                                                                         C
C       Note also the techniques we use here to manipulate character data     C
C       in FORTRAN 5.  We store characters in integer arrays and manipulate   C
C       them with the BYTE function and other user-written subroutines like   C
C       BYTEMOVER.                                                        C
C                                                                         C
C       You should also study the handling of the IER variable.           C
C                                                                         C
C-------------------------------------------------------------------------C

        INCLUDE "IIPAR.FR"

        INTEGER
     +  IARRAY  ( 44 )          ,     ; Interface array for 1 index level.
     +
     +  KEY     ( 10 )          ,
     +  RECORD  ( 15 )

        LOGICAL FLAG

C-------------------------------------------------------------------------C
C                                                                         C
C       First we must open the INFOS II file allowing one index level.  Note  C
C       that IARRAY is big enough for one level.                          C
C       After every INFOS operation you should compare IER with IONORM.       C
C       ( IONORM is defined in IIPAR.FR .)  If they are not equal, we will    C
C       have an error condition.                                          C
C                                                                         C
C-------------------------------------------------------------------------C

        CALL IOPEN (    IARRAY ,
     +                  "INFOS_TEST_FILE" , 1 ,
     +                  IER     )
        IPT = 100
        WRITE ( 10 , 999 )  IPT , IER
        IF ( IER .NE. IONORM )  GO TO 950
```

Figure C-2. Sample FORTRAN 5 Program Using the INFOS II Interface – II (continues)

```
C-------------------------------------------------------------------------C
C                                                                         C
C       Set up default locations for the key area and the data area.      C
C                                                                         C
C-------------------------------------------------------------------------C
        CALL IISET (    IARRAY ,
     +                  IIKEY , KEY ,
     +                  IIREC , RECORD ,
     +                  IER     )
        IPT = 200
        WRITE ( 10 , 999 )  IPT , IER
        IF ( IER .NE. IONORM )  GO TO 950

C-------------------------------------------------------------------------C
C                                                                         C
C       This is the main program loop.  It will read with relative motion C
C       and print out the information about the key and record that it    C
C       retrieved.                                                        C
C                                                                         C
C-------------------------------------------------------------------------C

300     CONTINUE

C       Read next key and record -- Exit if End of Subindex.

        CALL IRELREAD ( IARRAY , IER )
        IPT = 300
        WRITE ( 10 , 999 )  IPT , IER
        IF ( IER .EQ. IOESI )  GO TO 900                ; LOOP EXIT
        IF ( IER .NE. IONORM )  GO TO 950

C       Retrieve values of status information and lengths from the interface
C       array.

        CALL IIGET (    IARRAY ,
     +                  IISIP , FLAG ,
     +                  IIKLEN , KEYLEN ,
     +                  IIRLEN , RECLEN ,
     +                  IER     )
        IPT = 400
        WRITE ( 10 , 999 )  IPT , IER
        IF ( IER .NE. IONORM )  GO TO 950
```

Figure C-2. Sample FORTRAN 5 Program Using the INFOS II Interface – II (continued)

```
C       Print information about the current record.

        BYTE ( KEY , KEYLEN + 1 ) = 0
        BYTE ( RECORD , RECLEN + 1 ) = 0

        WRITE ( 10 , 401 )  KEY ( 1 ) , RECORD ( 1 ) , KEYLEN , RECLEN
  401   FORMAT ( // , "KEY       IS : " , S20 , / ,
     +             "RECORD    IS : " , S20 , / ,
     +             "KEY LENGTH    = " , I5 , / ,
     +             "RECORD LENGTH = " , I5 , / )
        IF ( FLAG )  WRITE ( 10 , 402 )
  402   FORMAT ( "SUBINDEX IS PRESENT" , / )

        GO TO 300
C-------------------------------------------------------------------------------C
C                                                                               C
C       You should always close an INFOS file when you are finished             C
C       processing.                                                             C
C                                                                               C
C-------------------------------------------------------------------------------C

900     CALL ICLOSE ( IARRAY , IER )
        IPT = 900
        WRITE ( 10 , 999 )  IPT , IER

        CALL EXIT

C-------------------------------------------------------------------------------C
C                                                                               C
C       Set up an error-handler for unexpected INFOS error code returns.       C
C                                                                               C
C-------------------------------------------------------------------------------C

950     WRITE ( 10 , 951 )  IER , IPT
951     FORMAT  ( // , "***** ABNORMAL ERROR RETURN *****" , / ,
     +            "RETURNED ERROR CODE = " , I7 , / ,
     +            " AT POINT " , I6 , / ,
     +            "***** WILL CLOSE ALL FILES AND TERMINATE PROGRAM *****" , / )

        GO TO 900

999     FORMAT ( "OPERATION AT POINT " , I5 , " RESULTED IN IER = " , I6 )

        END
```

Figure C-2. Sample FORTRAN 5 Program Using the INFOS II Interface – II (concluded)

Figure C-3 contains a sample FORTRAN 77 program that uses the INFOS II interface. The initial comment section of the program explains what the program does. You might find that you can adapt this program to your own application.

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                           C
C       PROGRAM CUSTOMERS                                                   C
C                                                                           C
C       This interactive program builds an INFOS II file of customer       C
C       records and allows you to modify it.  The DBAM file will have       C
C       two index levels.  The main (level 0) index will contain two       C
C       selector keys -- one pointing to a company names subindex           C
C       (NAM) and the other to a subindex of company numbers (NUM).         C
C       The keys in this subindex level will point to database              C
C       records that consist of the company's name, number, address,        C
C       telephone number, and person to contact.                            C
C                                                                           C
C       We will allow duplicate company name keys.  They will have          C
C       partial records consisting of the telephone number.  The NUM        C
C       keys will not include duplicates.  They will point to the           C
C       same database records as do the NAM keys.  They will also           C
C       have partial records which consist of the company telephone         C
C       number.                                                             C
C                                                                           C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC


        INCLUDE "COMMON_DATA.IN" ! Data file containing common variables
        INCLUDE "IIPAR.F77"      ! INFOS II parameter file
        INTEGER  MOD
C
C       Initialize the ERROR flag to .FALSE.
C
        ERROR = .FALSE.
C
C       Print a message on the screen to tell the user that the program
C       has begun, and explain what it will do.
C
        PRINT 10
10      FORMAT('1', 16X, 'PROGRAM CUSTOMERS',//)
        PRINT *, 'This program will build a database of your customers.'
        PRINT *, 'Enter the appropriate information when you receive'
        PRINT *, 'the prompts.'
        PRINT *, ' '
        PRINT *, 'If you don''t know the correct information for any'
        PRINT *, 'prompt, enter a question mark or a blank.  You can'
        PRINT *, 'substitute the correct information later.'
        PRINT *, ' '
        PRINT *, 'If your file is already built, you can use this program'
        PRINT *, 'to modify it.  Respond to the prompts appropriately.'
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continues)

```
C
C       Open the file, allowing 2 index levels.
C
        CALL IOPEN  (   IARRAY,
     +                  "CUSTOMERS", 2,
     +                  IER, ENDLIST    )
        IPT = 15
        IF ( IER .NE. IONORM ) GO TO 200
C
C       Test to see if the file is already built.  If it isn't, we will
C       get a KEYED POSITIONING ERROR, and this error return will send
C       us to the IWRITE subroutine to write the selector keys.
C
        CALL IKEYREAD ( IARRAY,
     +                  IIAMODE, KEY ONLY,
     +                  IIKEY, "NAM",
     +                  IIKLEN, 3,
     +                  IER, ENDLIST    )
        IPT = 25
        IF ( IER .EQ. IOKPE ) GO TO 30
        IF ( IER .NE. IONORM ) GO TO 200
        PRINT *, ' '
        PAUSE
        GO TO 80
C
C       Write 2 selector keys into the main index.
C
30      CALL IWRITE  (  IARRAY,
     +                  IIAMODE, SUPP DB,
     +                  IIKEY, "NAM",
     +                  IIKLEN, 3,
     +                  IER, ENDLIST    )
        IPT = 35
        IF ( IER .NE. IONORM ) GO TO 200
C
        CALL IWRITE  (  IARRAY,
     +                  IIAMODE, SUPP DB,
     +                  IIKEY, "NUM",
     +                  IIKLEN, 3,
     +                  IER, ENDLIST    )
        IPT = 45
        IF ( IER .NE. IONORM ) GO TO 200
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
C
C       Define a subindex under each selector key.  Allow each to
C       have partial records and allow the NAM subindex to have
C       duplicate keys.
C
        CALL ISDEFINE ( IARRAY,
     +                  IIKEY, "NAM",
     +                  IIKLEN, 3,
     +                  IISALLOW, ALLOW DUP,
     +                  IISPRLEN, 30,
     +                  IISMKLEN, 40,
     +                  IER, ENDLIST    )
        IPT = 55
        IF ( IER .NE. IONORM ) GO TO 200
C
        CALL ISDEFINE ( IARRAY,
     +                  IIKEY, "NUM",
     +                  IIKLEN, 3,
     +                  IISPRLEN, 30,
     +                  IISMKLEN, 40,
     +                  IER, ENDLIST    )
        IPT = 65
        IF ( IER .NE. IONORM ) GO TO 200
C
C       Call the ADD_REC subroutine to enter the name keys into the
C       NAM subindex and their corresponding records into the database.
C       Write inverted to place the company number keys in the NUM
C       subindex and make them point to the same data records.  The user
C       can enter all the information currently available, and later use
C       the same subroutine to add more customers or change records.
C
        FIRST_TIME = .TRUE.
        OPERATION = 'add    '
        CALL ADD_REC
        IF ( ERROR ) GO TO 200
C
C       Determine if the user wants to continue processing or exit
C       the program.
C
70      PRINT *, ' '
        PRINT *, 'Do you want to modify the file now, or exit the'
        PRINT *, 'program?  (1 = modify; 2 = exit)'
        PRINT *, ' '
        READ *, MOD
        IF ( MOD .NE. 1 ) GO TO 300
C
C       Call the MENU subroutine to print out the modification menu
C       and read the response.
C
80      CALL MENU
        IF ( ERROR ) GO TO 200
C
        GO TO ( 90, 100, 110, 120 ) MODTYPE
        GO TO 300
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                           C
C       Add customers to the database.                                      C
C                                                                           C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

90      FIRST_TIME = .TRUE.
        OPERATION = 'add    '
        CALL ADD_REC
        IF ( ERROR ) GO TO 200
        GO TO 80
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                           C
C       Change a customer's record.                                         C
C                                                                           C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

100     FIRST_TIME = .FALSE.
        OPERATION = 'modify'
        CALL SUBINDEX
        CALL READ_REL
        IF ( ERROR ) GO TO 200
        IF ( CORRECT .EQ. 1 ) THEN
           CALL ADD_REC
           IF ( ERROR ) GO TO 200
        ENDIF
        GO TO 80

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                           C
C       Delete a customer's record.                                         C
C                                                                           C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

110     CALL SUBINDEX
        OPERATION = 'delete'
        CALL READ_REL
        IF ( ERROR ) GO TO 200
        IF ( CORRECT .EQ. 1 ) THEN
           CALL DEL_REC
           IF ( ERROR ) GO TO 200
        ENDIF
        GO TO 70
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C       Read a customer's record.                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

120     CALL SUBINDEX
        OPERATION = 'read  '
        CALL READ_REL
        IF ( ERROR ) GO TO 200
        GO TO 70
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C       Error handler for unexpected INFOS error code returns.         C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

200     WRITE ( 10, 210 ) IER, IPT
210     FORMAT ( //, ' ****** ABNORMAL ERROR RETURN ******', / ,
     +        ' RETURNED ERROR CODE = ', I7, / , ' AT POINT = ', I6, / ,
     +        ' *** WILL CLOSE ALL FILES AND TERMINATE PROGRAM ***', / )

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C       Close the INFOS II file and exit the program.                   C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

300     CALL ICLOSE ( IARRAY,  ENDLIST )
        IPT = 305
        PRINT *, ' '
        PRINT *, 'The file is now closed and you are exiting the program.'
        PRINT *, ' '
        STOP
        END
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C       ADD_REC                                                        C
C                                                                      C
C       This subroutine prompts the user for the information to write  C
C       company name and number keys and the customer record.  It      C
C       writes the record into the database and the keys into the      C
C       correct subindexes.  The subroutine continues to loop until    C
C       the user enters a question mark to return to the main program. C
C       It also rewrites existing records if the user selected the     C
C       option to modify the record.                                   C
C                                                                      C
C       Line numbers are 400 - 490.                                    C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        SUBROUTINE ADD_REC

        INCLUDE "COMMON_DATA.IN"
        INCLUDE "IIPAR.F77"
        INTEGER N
C
        ERROR = .FALSE.
        IF ( .NOT. FIRST_TIME ) GO TO 420
400     PRINT *, ' '
        PRINT *, 'Company Name:'
        PRINT *, '(Enter a ? to return to the modification menu.)'
        PRINT *, ' '
        READ (5,410)  TEMPSTRING
410     FORMAT (A)
        IF ( TEMPSTRING .EQ. '?' ) GO TO 490
        CO_NAME = TEMPSTRING
        PRINT *, ' '
        PRINT *, 'Company Number: '
        PRINT *, ' '
        READ (5,410) TEMPSTRING
        IF ( TEMPSTRING .NE. ' ' ) CO_NUM = TEMPSTRING
420     PRINT *, ' '
        PRINT *, 'Address: '
        PRINT *, '  '
        READ (5,410) TEMPSTRING
        IF ( TEMPSTRING .NE. ' ' ) ADDRESS = TEMPSTRING
        PRINT *, ' '
        PRINT *, 'Phone Number (include area code and extension):  '
        PRINT *, ' '
        READ (5,410) TEMPSTRING
        IF ( TEMPSTRING .NE. ' ' ) PHONE = TEMPSTRING
        PRINT *, ' '
        PRINT *, 'Person to Contact: '
        PRINT *, ' '
        READ (5,410) TEMPSTRING
        IF ( TEMPSTRING .NE. ' ' ) CONTACT = TEMPSTRING
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
C
C       Store the information in RECORD.
C
430     RECORD = CO_NAME//CO_NUM//ADDRESS//PHONE//CONTACT
        IF ( .NOT. FIRST_TIME  ) GO TO 460
C
C       Write the record into the database and the number key into the
C       NUM subindex.
C
        CALL IWRITE  (  IARRAY,
     +                  IIKEY, "NUM",
     +                  IIKLEN, 3,
     +                  IIKEY2, CO_NUM,
     +                  IIKLEN2, 8,
     +                  IINLEV, 2,
     +                  IIPMODE, SET POS,
     +                  IIREC, RECORD,
     +                  IIRLEN, 188,
     +                  IER, ENDLIST    )
        IPT = 435
        IF ( IER .EQ. IOKAE ) THEN
           PRINT *, ' '
           PRINT *, 'You already have a company with this number.'
           PRINT *, 'Enter a different number, or press NEWLINE to'
           PRINT *, 'continue entering company names.'
           PRINT *, ' '
           READ(5,410) CO_NUM
           IF ( CO_NUM .NE. ' ' ) GO TO 430
           GO TO 400
        ENDIF
        IF ( IER .NE. IONORM ) THEN
           ERROR = .TRUE.
           GO TO 490
        ENDIF
C
C       Write inverted to include the company name key in the NAM
C       subindex and make it point to the appropriate data record.
C
        CALL IIGET  (   IARRAY,
     +                  IIFDBKH, DB_FDBK_H,
     +                  IIFDBKL, DB_FDBK_L,
     +                  IIOCCH2, OCC_NUM_H,
     +                  IIOCCL2, OCC_NUM_L,
     +                  IER, ENDLIST    )
        CALL IISET  (   IARRAY,
     +                  IIFDBKH, DB_FDBK_H,
     +                  IIFDBKL, DB_FDBK_L,
     +                  IIOCCH2, OCC_NUM_H,
     +                  IIOCCL2, OCC_NUM_L,
     +                  IER, ENDLIST    )
        IPT = 445
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
      IF ( IER .NE. IONORM ) THEN
         ERROR = .TRUE.
         GO TO 490
      ENDIF

      CALL IWRITE (    IARRAY,
     +                 IIAMODE, INVERT,
     +                 IIKEY, "NAM",
     +                 IIKLEN, 3,
     +                 IIKEY2, CO_NAME,
     +                 IIKLEN2, 40,
     +                 IINLEV, 2,
     +                 IIKMODE2, DUP KEY,
     +                 IIPREC, PHONE,
     +                 IER, ENDLIST    )
      IPT = 455
      IF ( IER .NE. IONORM ) THEN
         ERROR = .TRUE.
         GO TO 490
      ENDIF
      GO TO 470
460   CALL IREWRITE ( IARRAY,
     +                 IISMODE, RELATIVE,
     +                 IIRMODE, STATIC,
     +                 IIPREC, PHONE,
     +                 IIREC, RECORD,
     +                 IIRLEN, 188,
     +                 IER, ENDLIST    )
      IPT = 465
      IF ( IER .NE. IONORM ) THEN
         ERROR = .TRUE.
         GO TO 490
      ENDIF
470   CALL SET_PRINT_RECORD
      PRINT 480
480   FORMAT ( 1X, 'Here is the record you just wrote.  Do you want
     +  to make any changes to it?  (1 = YES; 2 = NO)', // )
      READ *,  N
      IF (N .EQ. 1) THEN
         FIRST_TIME = .FALSE.
         PRINT *, ' '
         PRINT *, 'Press NEWLINE in response to the prompt if you '
         PRINT *, 'don''t want to make any changes to that portion'
         PRINT *, 'of the record.'
         GO TO 420
      ENDIF
      FIRST_TIME = .TRUE.
      IF ( OPERATION .EQ. 'add    ' ) GO TO 400
490   RETURN
      END
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C       READ_REL                                                        C
C                                                                       C
C       This subroutine uses relative motion to locate the key that     C
C       the user wants to gain access to for the current operation.     C
C                                                                       C
C       Lines are numbered 500 - 580.                                   C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        SUBROUTINE READ_REL

        INCLUDE "COMMON_DATA.IN"
        INCLUDE "IIPAR.F77"
C
        CHARACTER * 3 NAM_NUM
        CHARACTER * 40 COMPANY
        INTEGER N
C
        IF ( WHICHSUB .EQ. 1 ) THEN
           NAM_NUM = "NAM"
           KEY_LEN = 40
        ELSE
           NAM_NUM = "NUM"
           KEY_LEN = 8
        ENDIF
C
500     PRINT 510, OPERATION
510     FORMAT (1X,'Which company''s record do you want to ', A6,'?',//)
        READ (5,520) COMPANY
520     FORMAT (A)
        PRINT *, ' '
        CALL IKEYREAD ( IARRAY,
     +                  IIKEY, NAM_NUM,
     +                  IIKLEN, 3,
     +                  IIKEY2, COMPANY,
     +                  IIKLEN2, KEY_LEN,
     +                  IIAMODE, SUPP PR,
     +                  IIPMODE, SET POS,
     +                  IINLEV, 2,
     +                  IIREC, RECORD,
     +                  IER, ENDLIST    )
        IPT = 525
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
530      IF ( IER .EQ. IOKPE ) GO TO 570
540      IF ( IER .NE. IONORM ) THEN
            ERROR = .TRUE.
            GO TO 580
         ENDIF
         CALL IIGET  (   IARRAY,
     +                   IIOCCH2, OCC_NUM_H,
     +                   IIOCCL2, OCC_NUM_L,
     +                   IER, ENDLIST    )
         IPT = 545
         IF ( IER .NE. IONORM ) THEN
            ERROR = .TRUE.
            GO TO 580
         ENDIF
         CALL SET_PRINT_RECORD
         PRINT 550, OPERATION
550      FORMAT (1X, 'Is this the record you want to ', A6, '?', /, 1X,
     +           '(1 = YES; 2 = NO)', //)
         READ *, CORRECT
         IF ( CORRECT .EQ. 1 ) THEN
            IF ( OPERATION .EQ. 'delete' .OR. OPERATION .EQ. 'read  ')
     +         GO TO 580
            IF ( OPERATION .EQ. 'modify' ) THEN
               PRINT *, ' '
               PRINT *,'When you receive each prompt, enter the modified'
               PRINT *, 'line, or press NEWLINE if you don''t want to'
               PRINT *, 'make any changes to that portion of the record.'
               CALL IISET   (    IARRAY,
     +                         IIOCCH2, OCC_NUM_H,
     +                         IIOCCL2, OCC_NUM_L,
     +                         IER, ENDLIST    )
               IPT = 555
               IF ( IER .NE. IONORM ) ERROR = .TRUE.
               GO TO 580
            ENDIF
         ELSE
            IF ( WHICHSUB .EQ. 2 ) GO TO 570
560         CALL IRELREAD  (     IARRAY,
     +                         IIRMODE, FOR,
     +                         IIAMODE, SUPP PR,
     +                         IIPMODE, SET POS,
     +                         IIKEY2, COMPANY,
     +                         IIKLEN2, KEY_LEN,
     +                         IIREC, RECORD,
     +                         IER, ENDLIST    )
            IPT = 565
            IF ( IER .EQ. IOESI ) THEN
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
570           PRINT *, 'Either you entered the company incorrectly, or'
              PRINT *, 'this customer is not on the customer list.'
              PRINT *, 'Do you want to try re-entering it?  '
              PRINT *, '(1 = YES; 2 = NO) '
              PRINT *, ' '
              READ *,  N
              IF ( N .EQ. 1 ) GO TO 500
            ENDIF
          ENDIF
580       RETURN
          END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                         C
C       SET_PRINT_RECORD                                                  C
C                                                                         C
C       This subroutine sets values to the current record and prints      C
C       out the record on the screen.                                     C
C                                                                         C
C       Lines are numbered in the 600s.                                   C
C                                                                         C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

          SUBROUTINE SET_PRINT_RECORD

          INCLUDE "COMMON_DATA.IN"
C
C         Initialize character strings to correct values.
C
          CO_NAME = RECORD(1:40)
          CO_NUM  = RECORD(41:48)
          ADDRESS = RECORD(49:118)
          PHONE   = RECORD(119:148)
          CONTACT = RECORD(149:188)
C
C         Print out the record.
C
          PRINT 600, CO_NAME, CO_NUM, ADDRESS, PHONE, CONTACT
600       FORMAT ( '1', A, /, 1X, A, /, 1X, A, /, 1X, A, /, 1X,
     +           'Contact: ', A, // )
C
          RETURN
          END
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C       SUBINDEX                                                        C
C                                                                       C
C       This subroutine determines which subindex we will reference     C
C       the data records through.                                       C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        SUBROUTINE SUBINDEX
C
        INCLUDE "COMMON_DATA.IN"
C
        PRINT *, ' '
        PRINT *, 'Do you want to reference the customer record by'
        PRINT *, 'company name or company number? (1 = NAME; 2 = NUMBER)'
        PRINT *, ' '
        READ *, WHICHSUB
        RETURN
        END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                       C
C       DEL_REC                                                         C
C                                                                       C
C       This subroutine physically deletes the record that we           C
C       positioned to with the READ_REL subroutine.                     C
C                                                                       C
C       Lines are numbered 700 - 730.                                   C
C                                                                       C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        SUBROUTINE DEL_REC

        INCLUDE "COMMON_DATA.IN"
        INCLUDE "IIPAR.F77"
C
C       Delete the NAM key.
C
        CALL IDELETE  ( IARRAY,
     +                  IIKEY, "NAM",
     +                  IIKLEN, 3,
     +                  IIKEY2, CO_NAME,
     +                  IIKLEN2, 40,
     +                  IINLEV, 2,
     +                  IIKMODE2, DUP KEY,
     +                  IISMODE, KEYED,
     +                  IER, ENDLIST    )
           IPT = 705
           IF (IER .NE. IONORM) THEN
              ERROR = .TRUE.
              GO TO 730
           ENDIF
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
C
C       Call the IDELETE subroutine again to delete the NUM key.
C
        CALL IDELETE  ( IARRAY,
     +                  IIKEY, "NUM",
     +                  IIKLEN, 3,
     +                  IIKEY2, CO_NUM,
     +                  IIKLEN2, 8,
     +                  IINLEV, 2,
     +                  IISMODE, KEYED,
     +                  IER, ENDLIST    )
           IPT = 715
           IF ( IER .NE. IONORM ) THEN
              ERROR = .TRUE.
              GO TO 730
           ENDIF
           PRINT 720
720        FORMAT ( /, 1X, 'The record has been deleted.', / )
730        RETURN
           END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                           C
C       MENU                                                                C
C                                                                           C
C       This subroutine initializes record fields to null strings,          C
C       sets occurrence numbers to zero, lists modification options         C
C       in a modification menu, and reads in the user's option choice.      C
C                                                                           C
C       Lines are numbered in the 800s.                                     C
C                                                                           C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
        SUBROUTINE MENU

        INCLUDE "COMMON_DATA.IN"
        INCLUDE "IIPAR.F77"
C
        CO_NAME = ' '
        CO_NUM  = ' '
        ADDRESS = ' '
        PHONE   = ' '
        CONTACT = ' '
        RECORD  = ' '
        CALL IISET  (   IARRAY,
     +                  IIOCCH, 0,
     +                  IIOCCL, 0,
     +                  IER, ENDLIST    )
           IPT = 805
           IF ( IER .NE. IONORM ) THEN
              ERROR = .TRUE.
              GO TO 820
           ENDIF
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (continued)

```
C
      PRINT 810
810   FORMAT('1', 5X, 'Modification Menu',//)
      PRINT *, 'What type of operation do you want to do?'
      PRINT *, ' '
      PRINT *, '1  Add customers to the database'
      PRINT *, '2  Change a customer''s record'
      PRINT *, '3  Delete a customer from the database'
      PRINT *, '4  Read a customer''s record'
      PRINT *, '5  Exit the program'
      PRINT *, ' '
      PRINT *, '(Enter the correct number only.  Any other response'
      PRINT *, ' will terminate this program.)'
      PRINT *, ' '
      READ *,  MODTYPE
820   RETURN
      END
```

Figure C-3. Sample FORTRAN 77 Program Using the INFOS II Interface (concluded)

Figures C-4 and C-5 contain sample programs in the DG/L language that use the INFOS II interface.

```
/*                                                          IN FILE SAMP1.DG
------------------------------------------------------------------------------

               Sample DG/L Program Using the INFOS II Interface

        This program uses the DG/L INFOS II interface to build a
        test file.  Note that it is not an exhaustive demonstration
        of the INFOS interface.

        Be sure to study the manner in which we handle the IER variable.

------------------------------------------------------------------------------
                                                                             */
        BEGIN

        INCLUDE IIPAR.DG;


        INTEGER ARRAY IARRAY[56];       % Interface array for 2 index levels.
        STRING (10) KEY;
        STRING (30) RECORD;
        STRING (80) TEMP_STR;           % String for some, but not all, of REC.
        STRING (5)  NUM_STR;            % String containing numbers (1-5).
        STRING (80) SCRATCH;            % Scratch storage area.
        INTEGER IER ;                   % Error return on INFOS calls.
        INTEGER KEY_NUM ;               % Looping variable when writing keys.

        EXTERNAL STRING PROCEDURE GETCOUTPUT;
        EXTERNAL PROCEDURE SYSRETURN;
        EXTERNAL PROCEDURE IOPEN , IWRITE , IKEYREAD , ISDEFINE , ICLOSE ;


/*
------------------------------------------------------------------------------

        PROCEDURE TYPE(STR), writes the string, STR, to output,
        followed by a NEW LINE <NL>.

------------------------------------------------------------------------------
                                                                             */
                        PROCEDURE TYPE (STR);
                                STRING (80) STR;
                        BEGIN WRITE (1,STR!!"<NL>");
                        END OF PROCEDURE TYPE;
```

Figure C-4. Sample DG/L Program Using the INFOS II Interface – I (continues)

```
/*
-------------------------------------------------------------------------------

        PROCEDURE ERROR_TEST(IER,STR), tests IER against IONORM.  If there
        is no error, then it will print the command and continue with the
        program.  If there is an error, it will type the INFOS command and
        the returned error.  It then will close the INFOS file, and stop
        program execution.

-------------------------------------------------------------------------------
                                                                              */
                        PROCEDURE ERROR_TEST (IER,STR);
                                STRING (80) STR;
                                INTEGER     IER;
                        BEGIN   IF IER <> IONORM THEN
                                BEGIN
                                     TYPE ("ERROR ON INFOS CALL "!!STR);
                                     WRITE (1,"ERROR CODE = ",IER);
                                     ICLOSE ( IARRAY , IER );
                                     SYSRETURN;
                                END
                                ELSE TYPE ("NORMAL RETURN ON INFOS CALL "!!STR);

                        END OF PROCEDURE ERROR_TEST;


/*
-------------------------------------------------------------------------------

        First we must open the INFOS file, allowing two index levels.
        (Note that we have made IARRAY large enough for two levels.)
        After every INFOS operation we will compare IER with IONORM
        to determine the error status.  ( IONORM is defined in IIPAR.DG. )

-------------------------------------------------------------------------------
                                                                              */

        OPEN (1,(GETCOUTPUT));

        IOPEN(IARRAY , "INFOS_TEST_FILE<0>" , 2 , IER);
        ERROR_TEST (IER,"IOPEN");               % Error check on opening file.
```

Figure C-4. Sample DG/L Program Using the INFOS II Interface – I (continued)

```
/*
-------------------------------------------------------------------------------

        Write five keys in the top-level index and their records into the
        database:

                KEY: A          RECORD: AA
                KEY: AA         RECORD: CCDD
                KEY: AAA        RECORD: EEFFGG
                KEY: AAAA       RECORD: GGHHIIJJ
                KEY: AAAAA      RECORD: IIJJKKLLMM


-------------------------------------------------------------------------------
                                                                              */
        KEY := "AAAAA";
        RECORD := "AABBCCDDEEFFGGHHIIJJKKLLMMNNOO";
        NUM_STR := "12345";
        FOR KEY_NUM := 1 STEP 1 UNTIL 5 DO
                BEGIN   TEMP_STR := SUBSTR( RECORD,1+4*(KEY_NUM-1),30);
                        IWRITE(         IARRAY ,
                                        IIKEY , KEY ,
                                        IIKLEN , KEY_NUM ,
                                        IIREC , TEMP_STR ,
                                        IIRLEN , KEY_NUM * 2 ,
                                        IER );
                        ERROR_TEST( IER , "IWRITE NUMBER : "!!SUBSTR(NUM_STR,
                                        KEY_NUM) );
                END OF WRITING RECORDS & KEYS;
/*
-------------------------------------------------------------------------------

        Do a generic keyed search, suppressing the database, to find out if
        there is a key that starts with a 'B'.

-------------------------------------------------------------------------------
                                                                              */
        SCRATCH := "B";

        IKEYREAD (      IARRAY ,
                        IIKEY , SCRATCH ,
                        IIKLEN , 1 ,
                        IIKMODE , GEN_KEY ,
                        IIAMODE , SUPP_DB ,
                        IER             );
        IF IER = IOKPE THEN TYPE("NO KEY STARTS WITH B")
                       ELSE ERROR_TEST( IER , "IKEYREAD" );
```

Figure C-4. Sample DG/L Program Using the INFOS II Interface – I (continued)

```
/*
-------------------------------------------------------------------------------

        Create a subindex under KEY : AA ,
        then enter KEY : BBB in that subindex with
        no database record and with PARTIAL RECORD : 12345 .

-------------------------------------------------------------------------------
                                                                              */
        ISDEFINE (      IARRAY ,
                        IIKEY , "AA" ,
                        IIKLEN , 2 ,
                        IISPRLEN , 5 ,
                        IER             );
        ERROR_TEST( IER , "ISDEFINE" );

        IWRITE (        IARRAY ,
                        IINLEV , 2 ,
                        IIAMODE , SUPP_DB ,
                        IIKEY , "AA" ,
                        IIKLEN , 2 ,
                        IIKEY2 , "BBB" ,
                        IIKLEN2 , 3 ,
                        IIPREC , "12345" ,
                        IER             );
        ERROR_TEST( IER , "IWRITE (SUBINDEX)" );
/*
-------------------------------------------------------------------------------

        You should always close an INFOS file when you are
        finished processing.

-------------------------------------------------------------------------------
                                                                              */
        ICLOSE( IARRAY , IER );
        ERROR_TEST( IER , "ICLOSE" );

        END OF TEST PROGRAM;
```

Figure C-4. Sample DG/L Program Using the INFOS II Interface – I (concluded)

```
/*
------------------------------------------------------------------------

               Sample DG/L Program Using the INFOS II Interface

        This program uses the DG/L INFOS interface to display a
        test file.  Note that it is not an exhaustive demonstration
        of the INFOS interface.

        You also should study the handling of the IER variable.

------------------------------------------------------------------------
                                                                        */
BEGIN
        INCLUDE IIPAR.DG;

        INTEGER ARRAY IARRAY[44] ;      % Interface array for 1 index level.
        INTEGER  KEYLEN ;               % The returned key length from READ.
        INTEGER  RECLEN ;               % The returned record length from READ.
        INTEGER  KEY_NUM ;              % Looping variable when reading keys.
        INTEGER  IER ;                  % Error return variable on INFOS calls.
        STRING (20) KEY;
        STRING (30) RECORD;
        STRING (20) NUM_STR;            % String containing numbers (1-9).
        BOOLEAN FLAG ;                  % Logical flag, set true if SI present.

        EXTERNAL STRING PROCEDURE GETCOUTPUT;
        EXTERNAL PROCEDURE SYSRETURN;
        EXTERNAL PROCEDURE IOPEN , IISET , IRELREAD , IIGET , ICLOSE;

/*
------------------------------------------------------------------------

        PROCEDURE TYPE(STR) writes the string, STR, to output,
        followed by a NEW LINE <NL>.

------------------------------------------------------------------------
                                                                        */
                        PROCEDURE TYPE (STR);
                                STRING (80) STR;
                        BEGIN WRITE (1,STR!!"<NL>");
                        END OF PROCEDURE TYPE;
```

Figure C-5. Sample DG/L Program Using the INFOS II Interface – II (continues)

```
/*
-------------------------------------------------------------------------------

        PROCEDURE ERROR_TEST(IER,STR) tests IER against IONORM.  If there
        is no error, it will print the command and continue with the program.
        If there is an error, it will type the INFOS command and the returned
        error.  It then will close the INFOS file, and stop program execution.

-------------------------------------------------------------------------------
                                                                              */
                        PROCEDURE ERROR_TEST (IER,STR);
                                STRING (80) STR;
                                INTEGER     IER;
                        BEGIN   IF IER <> IONORM THEN
                                BEGIN
                                     TYPE ("ERROR ON INFOS CALL "!!STR);
                                     WRITE (1,"ERROR CODE = ",IER);
                                     ICLOSE ( IARRAY , IER );
                                     SYSRETURN;
                                END
                                ELSE TYPE("NORMAL RETURN ON INFOS CALL "!!STR);
                        END OF PROCEDURE ERROR_TEST;

/*
-------------------------------------------------------------------------------

        First we must open the INFOS file, allowing one index level.  Note
        that we defined IARRAY big enough for one level.
        After every INFOS operation, we will compare IER with IONORM, to see
        if there is an error.  ( IONORM is defined in IIPAR.DG . )

-------------------------------------------------------------------------------
                                                                              */
        OPEN(1,(GETCOUTPUT));
        IOPEN ( IARRAY , "INFOS_TEST_FILE<0>" , 1 , IER );
        ERROR_TEST (IER , "IOPEN");
/*
-------------------------------------------------------------------------------

        Set up default locations for the key area and the data area.

-------------------------------------------------------------------------------
                                                                              */
        IISET (         IARRAY ,
                        IIKEY , KEY ,
                        IIREC , RECORD ,
                        IER     );
        ERROR_TEST( IER , "IISET" );
```

Figure C-5. Sample DG/L Program Using the INFOS II Interface – II (continued)

```
/*
-------------------------------------------------------------------------

        This is the main program loop.  It will read with relative motion
        and print out the information about the key and record that it
        retrieved.

-------------------------------------------------------------------------
                                                                        */
        KEY_NUM := 0;
        NUM_STR := "123456789";

        UNTIL IER = IOESI DO
        BEGIN

/*              Read the next key and record - Exit if End of Subindex  */

                KEY_NUM := KEY_NUM + 1;
                IRELREAD ( IARRAY , IER );
                IF  IER = IOESI THEN  GOTO END_OF_SI;          % LOOP EXIT
                ERROR_TEST( IER , "IRELREAD NUMBER "!!SUBSTR(NUM_STR,KEY_NUM) );

/*              Retrieve values of status information and lengths from
                the interface array.                                    */

                IIGET ( IARRAY ,
                        IISIP , FLAG ,
                        IIKLEN , KEYLEN ,
                        IIRLEN , RECLEN ,
                        IER     );
                ERROR_TEST( IER , "IIGET NUMBER "!!SUBSTR(NUM_STR,KEY_NUM) );

/*              Print information about the current record.             */

                SETCURRENT( KEY , KEYLEN );
                SETCURRENT( RECORD , RECLEN );

                TYPE( "<NL><NL>KEY      IS : "!!KEY );
                TYPE( "RECORD   IS : "!!RECORD );
                WRITE( 1 , "KEY LENGTH  = " , KEYLEN , "<NL>" );
                WRITE( 1 , "RECORD LENGTH       = " , RECLEN , "<NL>" );
                IF  FLAG  THEN TYPE( "SUBINDEX IS PRESENT" );

END_OF_SI: END;
```

Figure C-5. Sample DG/L Program Using the INFOS II Interface – II (continued)

```
/*
-------------------------------------------------------------------------

        You should always close an INFOS file when you are finished
        processing.

-------------------------------------------------------------------------
                                                                        */
        ICLOSE ( IARRAY , IER );
        ERROR_TEST( IER , "ICLOSE" );

END OF PROGRAM SAMP2.DG;
```

Figure C-5. Sample DG/L Program Using the INFOS II Interface – II (concluded)

Figures C-6 and C-7 contain sample programs using the PL/I-INFOS II interface.

```
/*****************************************************************

            Sample PL/I Program Using the INFOS II Interface

        This program uses the PL/I INFOS II interface to build a
        test file.  Note that it is not an exhaustive demonstration
        of the INFOS interface.

        You should study the handling of the IER variable.

*****************************************************************/

        %INCLUDE "IIPAR.PL1";

        DECLARE IARRAY(56)      FIXED BINARY;
        DECLARE KEY             CHAR(20) ALIGNED ;
        DECLARE  RECORD         CHAR(30) ALIGNED ;

        DECLARE   SCRATCH       CHAR(80) ALIGNED  ;
        DECLARE (IER,IPT,I)     FIXED BINARY;

/*************************************************************************

        First we must open the INFOS file, allowing two index levels.
        Note that IARRAY is large enough for two levels.
        After every INFOS operation, we will compare IER with IONORM
        to see if there is an error.  ( IONORM is defined in IIPAR.PL1.)

*************************************************************************/

        PUT SKIP LIST("STARTING TEST");

        CALL IOPEN ( IARRAY , "INFOS_TEST_FILE"!!ASCII(0), 2 , IER ) ;
        IPT = 100  ;
        IF ( IER ^= IONORM ) THEN GO TO SURPRISE ;
         PUT SKIP LIST("OPERATION AT POINT ",IPT," IS O.K.");
```

Figure C-6. Sample PL/I Program Using the INFOS II Interface – I (continues)

```
/*****************************************************************************

        Write five keys in the top-level index and their records into the
        database:

                KEY: A          RECORD: AA
                KEY: AA         RECORD: CCDD
                KEY: AAA        RECORD: EEFFGG
                KEY: AAAA       RECORD: GGHHIIJJ
                KEY: AAAAA      RECORD: IIJJKKLLMM

*****************************************************************************/

        KEY = "AAAAA" ;
        RECORD = "AABBCCDDEEFFGGHHIIJJKKLLMMNNOO" ;

        DO      I = 1 TO 5 ;

        SCRATCH = SUBSTR ( RECORD , 1+4*(I-1) , 2*I ) ;

        CALL IWRIT (    IARRAY ,
                        IIKEY , KEY ,
                        IIKLEN , I ,
                        IIREC , SCRATCH ,
                        IIRLEN , 2 * I ,
                        IER     )  ;
        IPT = 200 + I  ;
        IF ( IER ^= IONORM ) THEN GO TO SURPRISE ;
        ELSE PUT SKIP LIST("OPERATION AT POINT ",IPT," IS O.K.");

        END ;  /* of DO-LOOP */

/*****************************************************************************

        Do a generic keyed search, suppressing the database, to find out
        if there is a key that starts with a 'B' .

*****************************************************************************/

        SUBSTR(SCRATCH,1,1) =  "B" ;

        CALL IKEYR (            IARRAY ,
                                IIKEY , SCRATCH ,
                                IIKLEN , 1 ,
                                IIKMODE , INF_GEN_KEY ,
                                IIAMODE , INF_SUPP_DB ,
                                IER     )  ;
        IPT     =       300   ;
        IF ( IER ^= IONORM ) THEN BEGIN;
                IF IER=IOKPE THEN PUT SKIP LIST("NO KEY STARTS WITH B");
                ELSE GO TO SURPRISE ;
                END ;
        ELSE PUT SKIP LIST("OPERATION AT POINT ",IPT," IS O.K.");
```

Figure C-6. Sample PL/I Program Using the INFOS II Interface – I (continued)

```
/*************************************************************************

        Create a subindex under KEY: AA ,
        then enter KEY: BBB in that subindex with no database record
        and with PARTIAL RECORD: 12345 .

*************************************************************************/

        CALL ISDEF (    IARRAY ,
                        IIKEY , "AA" ,
                        IIKLEN , 2 ,
                        IISPRLEN , 5 ,
                        IER     )  ;
        IPT = 400  ;
        IF ( IER ^= IONORM ) THEN GO TO SURPRISE ;
        ELSE PUT SKIP LIST("OPERATION AT POINT ",IPT," IS O.K.");

        CALL IWRIT (    IARRAY ,
                        IINLEV , 2 ,
                        IIAMODE , INF_SUPP_DB ,
                        IIKEY , "AA" ,
                        IIKLEN , 2 ,
                        IIKEY2 , "BBB" ,
                        IIKLEN2 , 3 ,
                        IIPREC , "12345" ,
                        IER     )  ;
        IPT = 500  ;
        IF ( IER ^= IONORM ) THEN GO TO SURPRISE ;
        ELSE PUT SKIP LIST("OPERATION AT POINT ",IPT," IS O.K.");

/*************************************************************************

        You should always close an INFOS file when you are finished
        processing.

*************************************************************************/

EXIT:
        CALL ICLOS ( IARRAY , IER )  ;
        IPT = 900  ;
        PUT SKIP LIST ("OPERATION AT POINT 900 RESULTED IN #",IER);

        RETURN ;
```

Figure C-6. Sample PL/I Program Using the INFOS II Interface – I (continued)

```
/*************************************************************************

        Set up an error-handler for unexpected INFOS error code returns.

*************************************************************************/

SURPRISE:
                PUT SKIP LIST (
                     "***** ABNORMAL ERROR RETURN *****" ,
                  "RETURNED ERROR CODE = " , IER,
                  " AT POINT " , IPT ) ;


        GO TO EXIT ;


        END  ;
```

Figure C-6. Sample PL/I Program Using the INFOS II Interface – I (concluded)

```
/*********************************************************************

            Sample PL/I Program Using INFOS II Interface

        This program uses the PL/I INFOS interface to display a
        test file.  Note that it is not an exhaustive demonstration
        of the INFOS interface.

        You also should study the handling of the IER variable.

*********************************************************************/

        %INCLUDE "IIPAR.PL1" ;

DECLARE IARRAY  ( 44 )          FIXED BINARY ;

DECLARE  (IER,IPT)              FIXED BINARY ;

DECLARE KEY                     CHAR(20) ALIGNED ;
DECLARE RECORD                  CHAR(30) ALIGNED ;
DECLARE  (RECLEN,KEYLEN)        FIXED BINARY ;

DECLARE  FLAG                   FIXED BINARY ;

/*********************************************************************

        First we must open the INFOS file allowing one index level.
        Note that IARRAY is big enough for one level.
        After every INFOS operation you should compare IER with IONORM,
        to see if there is an error.  ( IONORM is defined in IIPAR.PL1.)

 *********************************************************************/

        CALL IOPEN ( IARRAY , "INFOS_TEST_FILE"||ASCII(0), 1, IER ) ;
        IPT     =       100 ;
        PUT SKIP LIST ( IPT , IER ) ;
        IF ( IER ^= IONORM )  THEN GO TO SURPRISE ;
```

Figure C-7. Sample PL/I Program Using the INFOS II Interface – II (continues)

```
/*************************************************************************

        Set up default locations for the key area and the data area.

*************************************************************************/

        CALL IISET (            IARRAY ,
                                IIKEY , KEY ,
                                IIREC , RECORD ,
                                IER )  ;
        IPT     =       200  ;
        PUT SKIP LIST ( IPT , IER ) ;
        IF ( IER ^= IONORM )  THEN GO TO SURPRISE ;

/*************************************************************************

        This is the main program loop.  It will read with relative motion and
        print out the information about the key and record that it retrieved.

*************************************************************************/

MAINLOOP:

/*      Read next key and record - Exit if End of Subindex  */

        CALL IRELR ( IARRAY , IER ) ;
        IPT     =       300 ;
        PUT SKIP LIST ( IPT , IER  ) ;
        IF ( IER = IOESI )  THEN GO TO EXIT     ;       /* LOOP EXIT */
        IF ( IER ^= IONORM )  THEN GO TO SURPRISE ;

/*      Retrieve values of status information and lengths         */
/*      from the interface array.                                 */

        CALL IIGET (    IARRAY ,
                        IISIP , FLAG ,
                        IIKLEN , KEYLEN ,
                        IIRLEN , RECLEN ,
                        IER     )  ;
        IPT = 400   ;
        PUT SKIP LIST  ( IPT , IER  ) ;
        IF ( IER ^= IONORM )  THEN GO TO SURPRISE  ;
```

Figure C-7. Sample PL/I Program Using the INFOS II Interface – II (continued)

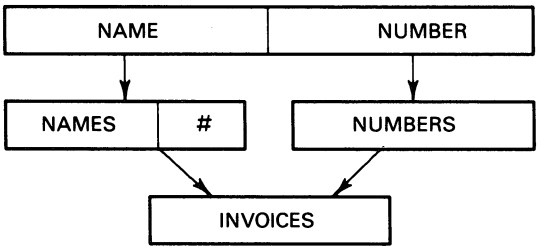
```
/*      Print information about current record.                 */

PUT SKIP LIST(      "KEY       IS : " , SUBSTR(KEY,1,KEYLEN),
                    "RECORD    IS : " , SUBSTR(RECORD,1,RECLEN),
                    "KEY LENGTH    = " , KEYLEN ,
                    "RECORD LENGTH = " , RECLEN )  ;

        IF ( FLAG = 1 )  THEN PUT SKIP LIST  ( "SUBINDEX IS PRESENT" )   ;

        GO TO MAINLOOP ;


/*********************************************************************

        You should always close an INFOS file when you are finished
        processing.

*********************************************************************/

EXIT:
        CALL ICLOS ( IARRAY , IER )  ;
        IPT     =       900 ;
        PUT  SKIP LIST("CLOSED FILE AT LOCATION ",IPT," WITH IER= ",IER);

        RETURN ;

/*************************************************************************

        Set up an error-handler for unexpected INFOS error code returns.

 *************************************************************************/

SURPRISE:
        PUT SKIP LIST(" ERROR RETURN FROM LOCATION ",IPT,
                        " WITH ERROR VALUE ",IER) ;

        GO TO EXIT ;

        END ;
```

Figure C-7. Sample PL/I Program Using the INFOS II Interface – II (concluded)

End of Appendix

Appendix D
Sample COBOL Program Using the INFOS II Interface

This COBOL program reads an AOS file named SAM1 and creates the following DBAM structure:



ID-01077

SAM1 has one type of record:

```
IN-TOY-RECORD
   POS  1-20  TOY NAME
   POS 21-24  TOY NUMBER
   POS 25-44  TOY COMPANY NAME
   POS 45-50  TOY QUANTITY
   POS 51-56  INVOICE NUMBER
   POS 57-60  INVOICE QUANTITY
```

The sequential file SAM1 looks like this:

```
CONEHEADS FOUR      4330WILTIN BADLEY       000250WB10050080
CONDEMNATION        4950WILTIN BADLEY       001520WB25600125
TOSS-A-BOSS GAME    2140HIGHDEAL            000685HI21180223
RATTRAP MARATHON    7085SPARKER SISTERS     015023SS10990572
SCRIBBLE            5200BELCHCOW & FIGHTER  055000BF20208050
SCRIBBLE            7150BELCHCOW & FIGHTER  006789BF20304230
DELUXE IRRITATION   8321FLAKESLIDE          000890FL50750200
FLUKES OF HAZARD    2158HIGHDEAL            123045HI45456000
CHATTER MONOTONE    7471FRISKER-TWICE       012605FT63020842
NERD BASKETBALL     1840SPARKER SISTERS     004324SS21251671
```

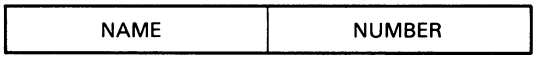
Below, we describe briefly what we accomplish with each step of the COBOL program.

STEP 1. OPEN-FILES

We open the sequential file SAM1 and open the INFOS file so we can build it through this COBOL program.

STEP 2. CREATE-SELECTORS

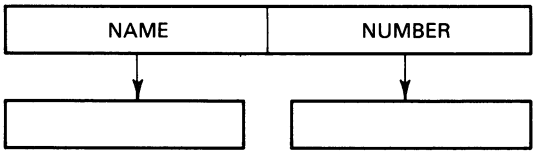
We write the selector keys, suppressing the database so that the system doesn't expect us to write a data record.



ID-01078

STEP 3. DEFINE-SUBINDEXES

We define a subindex under each of the selector keys and allow both of them to have subindexes as well. We also allow the subindex under the NAME selector key to have duplicate keys and partial records.

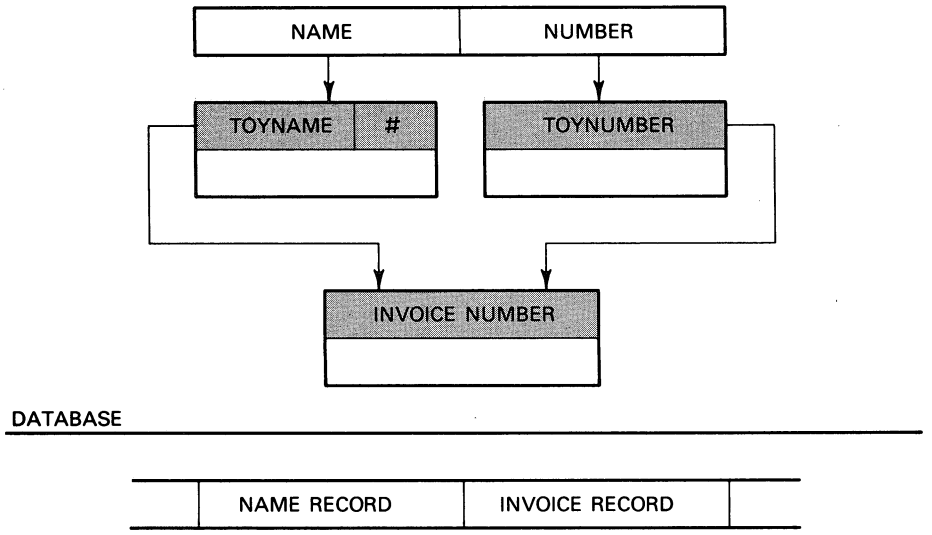


ID-01079

STEP 4. WRITE-KEYS

In this step, we write a name key into the NAME subindex, its corresponding record into the database, and a number into its partial record. We then perform an inverted write to place a number key into the NUMBER subindex and make it point to the name record in the database.

Next, we define a subindex under the name key we just wrote. We then link subindexes so that the number key points to the same subindex that the name key points to. Finally, we write an invoice key into this subindex and its invoice record into the database. We follow this procedure for every record in the sequential file.



STEP 5. READ-DOWN and READ-LEVEL1-KEYS

In this step, we read sequentially through the LEVEL1 name keys using relative access. First we read forward to set the current position on the NAME selector key. We then read down to set the current position just outside the NAME subindex. Next we move forward, fixing the current position on every key that we read until there are no more keys.

STEP 6. READ-PART-REC, READ-LEVEL2-KEY, READ-APPROXIMATE, and READ-GENERIC

In Step 6 we do a series of keyed read examples. First we use keyed access, suppressing the data record, to read a partial record. Then we do a keyed read of a LEVEL2 invoice key to obtain its invoice record.

Next we do an approximate read, which will retrieve the record having a key value equal to or greater than the key specified. Finally, we do a generic read, which retrieves the record having a key value equal to the specified key, up to the length of the key.

STEP 7. CLOSE-FILES

We close both the sequential file and the INFOS file.

STEP 8. OPEN-FILE

We open the INFOS file for both input and output.

STEP 9. ERROR-HANDLING

Remember that when we do a generic read, the INFOS II system locates the first key in the index that exactly matches the specified key, up to the length of that key. When we do a generic read and no key exists that matches the specified key, the INFOS II system returns a keyed positioning error.

In this step, we check the INFOS status variable I-STAT, and if an error occurs, we go to a routine that displays the error. Because we use an INFOS status variable to handle the error, the application program does not terminate abnormally.

STEP 10. LOGICAL-DELETE

We perform a logical deletion, causing both the partial record and the data record to be marked as logically deleted.

STEP 11. RETRIEVE-COMMAND

In this step, we use the RETRIEVE command to obtain information about a key. First, we retrieve a key, which moves the key value of the selected record into DEST1. Next, we retrieve a high key, moving the value of the highest key in the NAME subindex into DEST2. Finally, we do a RETRIEVE STATUS request, which places the status of the high key of the NAME subindex into the four-character data item, STAT1.

STEP 12. CHANGE-KEY

To change a key, we use the following series of operations:

1. Read the key and lock its record.
2. Write a name key, specifying inversion, because we want to link the key to the existing locked data record, instead of writing a new record.
3. Link the new name key to the subindex of the key we want changed.
4. Rewrite the data record, replacing the value of the old key in the record with the value of the new name key.
5. Delete the subindex of the old name key.
6. Delete the old name key.

STEP 13. DELETE-RECORD

In order to completely wipe out a toy from the INFOS file, we must first delete the subindex under the appropriate key through both the NAME and NUMBER selectors. We then delete the keys themselves through the selectors. This deletes the name and invoice records that the keys pointed to.

STEP 14. CLOSE-FILE

We close the INFOS file.

```
*****************************************************************
*                                                               *
*        Sample COBOL Program Using the INFOS II Interface      *
*                                                               *
*       This program uses the INFOS interface to build up and   *
*       display a toy file.                                     *
*                                                               *
*****************************************************************


IDENTIFICATION DIVISION.
PROGRAM-ID.  SAMPLE1-COBOL.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT IN-FILE ASSIGN "SAM1".
     SELECT OUT-FILE
          ASSIGN INDEX "DBAMX"
          ASSIGN DATA "DBAMX.DB"
          ORGANIZATION IS INDEXED
          ACCESS MODE DYNAMIC
          RECORD KEYS ARE
                NAME-KEY WITH DUPLICATES OCCURRENCE IS OCC-NUM
                NUMBER-KEY
                NAME-SELECTOR-KEY
                NUMBER-SELECTOR-KEY
                INVOICE-KEY
                PART-KEY
                APPROX-KEY
                GEN-KEY
                LEVEL2-KEY
          ALLOW SUB-INDEX
          LEVELS 3.

DATA DIVISION.
FILE SECTION.

FD  IN-FILE
    RECORD CONTAINS 60 CHARACTERS
    RECORDING MODE IS FIXED
    DATA RECORD IS IN-TOY-RECORD.

01  IN-TOY-RECORD.
    05  TOY-NAME          PIC X(20).

    05  TOY-NO            PIC 9(04).
    05  TOY-COMPANY       PIC X(20).
    05  TOY-QUANTITY      PIC 9(06).
    05  INV-NO            PIC X(06).
    05  INV-QUANTITY      PIC 9(04).
```

```
FD  OUT-FILE
    INDEX BLOCK CONTAINS 2048 CHARACTERS
    DATA BLOCK CONTAINS 2048 CHARACTERS
    INDEX NODE SIZE IS 2042
    RECORDING MODE VARIABLE
    RECORD LENGTH IS 50
    DATA RECORDS ARE
         OUT-TOY-RECORD
         OUT-INV-RECORD
    FEEDBACK IS REC-ADDR
    PARTIAL RECORD IS PART-REC.

01  OUT-TOY-RECORD.
    05  TOY-NAME          PIC X(20).
    05  TOY-NO            PIC 9(04).
    05  TOY-COMPANY       PIC X(20).
    05  TOY-QUANTITY      PIC 9(06).

01  OUT-INV-RECORD.
    05  OUT-INV-NO            PIC X(06).
    05  OUT-INV-QUANTITY      PIC 9(04).


WORKING-STORAGE SECTION.

01  NAME-SELECTOR-KEY     PIC X(04) VALUE "NAME".
01  NUMBER-SELECTOR-KEY   PIC X(06) VALUE "NUMBER".
01  NAME-KEY              PIC X(20).
01  NUMBER-KEY            PIC 9(04).
01  INVOICE-KEY           PIC X(06).
01  REC-ADDR              PIC X(04).
01  PART-REC              PIC 9(04).
01  PART-KEY              PIC X(20) VALUE "DELUXE IRRITATION".
01  APPROX-KEY            PIC X(20) VALUE "MONOTONY".
01  GEN-KEY               PIC X(08) VALUE "SCRIBBLE".
01  LEVEL2-KEY            PIC X(06) VALUE "WB2560".
01  OCC-NUM               PIC 9(04).

SCREEN SECTION.

01  MENU-1.
    05  BLANK SCREEN.
    05  LINE 11 COLUMN 14.
    05  "CREATING AND BUILDING THE MULTILEVEL DBAM FILE".
    05  LINE PLUS 2 COLUMN 33.
    05  VALUE IS '"DBAMX"'.
    05  LINE PLUS 3 COLUMN 36 "+" BLINK.
    05  LINE COLUMN 36 "+" BLINK.
    05  LINE COLUMN 36 "+" BLINK.
    05  LINE COLUMN 34 "+ + +" BLINK.
    05  LINE COLUMN 35 "+++" BLINK.
    05  LINE COLUMN 36 "+" BLINK.
```

```
01  MENU-2.
    05  BLANK SCREEN.

PROCEDURE DIVISION.

MAIN-PARAGRAPH.

*  The main paragraph is the control area.  It calls routines that
*  perform the tasks needed to complete the demonstration.

*  Perform STEP 1 -- Open the INFOS file and fixed-length file.
    PERFORM OPEN-FILES.

*  Perform STEP 2 -- Create the selector keys.
    PERFORM CREATE-SELECTORS.

*  Perform STEP 3 -- Define subindexes under the selector keys.
    PERFORM DEFINE-SUBINDEXES.

*  Perform STEP 4 -- Write LEVEL 1 and LEVEL 2 keys.
    PERFORM WRITE-KEYS.

*  Perform STEP 5 -- Read sequentially through LEVEL 1 keys.
    PERFORM READ-DOWN.
    PERFORM READ-LEVEL1-KEYS.

*  Perform STEP 6 -- Do keyed READ examples.
    PERFORM READ-PART-REC.
    PERFORM READ-LEVEL2-KEY.
    PERFORM READ-APPROXIMATE.
    PERFORM READ-GENERIC.

*  Perform STEP 7 -- Close the INFOS file and fixed-length file.
    PERFORM CLOSE-FILES.

*************
*  STEP 1   *
*************

OPEN-FILES.

    DISPLAY MENU-1.
    OPEN INPUT IN-FILE.
    OPEN OUTPUT OUT-FILE.
    CLOSE OUT-FILE.
    OPEN I-O OUT-FILE.
```

```
*************
*  STEP 2   *
*************

CREATE-SELECTORS.

    WRITE OUT-TOY-RECORD
    SUPPRESS DATA RECORD
    KEY IS NAME-SELECTOR-KEY.

    WRITE OUT-TOY-RECORD
    SUPPRESS DATA RECORD
    KEY IS NUMBER-SELECTOR-KEY.

*************
*  STEP 3   *
*************

DEFINE-SUBINDEXES.

    DEFINE SUB-INDEX OUT-FILE
         KEY IS NAME-SELECTOR-KEY
         INDEX NODE SIZE IS 2042
         PARTIAL RECORD LENGTH IS 4
         MAXIMUM KEY LENGTH IS 20
         ALLOW SUB-INDEX
         ALLOW DUPLICATES.

    DEFINE SUB-INDEX OUT-FILE
         KEY IS NUMBER-SELECTOR-KEY
         INDEX NODE SIZE IS 2042
         MAXIMUM KEY LENGTH IS 4
         ALLOW SUB-INDEX.

*************
*  STEP 4   *
*************

WRITE-KEYS.

    READ IN-FILE  AT END GO TO READ-DOWN.
    MOVE SPACES TO NAME-KEY.
    MOVE SPACES TO NUMBER-KEY.
    MOVE SPACES TO INVOICE-KEY.
    MOVE CORR IN-TOY-RECORD TO OUT-TOY-RECORD.
    MOVE TOY-NAME OF IN-TOY-RECORD TO NAME-KEY.
    MOVE TOY-NO OF IN-TOY-RECORD TO NUMBER-KEY.
    MOVE TOY-NO OF IN-TOY-RECORD TO PART-REC.
    WRITE OUT-TOY-RECORD
    KEYS ARE NAME-SELECTOR-KEY, NAME-KEY.
    WRITE INVERTED OUT-TOY-RECORD
    SUPPRESS DATA RECORD
    KEYS ARE NUMBER-SELECTOR-KEY, NUMBER-KEY.
```

```
*DEFINE-SUBINDEX.

    DEFINE SUB-INDEX OUT-FILE
    KEYS ARE NAME-SELECTOR-KEY, NAME-KEY
    INDEX NODE SIZE IS 2042
    MAXIMUM KEY LENGTH IS 6.

*LINK-SUBINDEX.

    LINK SUB-INDEX OUT-FILE
    SOURCE
         KEYS ARE NAME-SELECTOR-KEY, NAME-KEY
    DESTINATION
         KEYS ARE NUMBER-SELECTOR-KEY, NUMBER-KEY.

*WRITE-LEVEL2-KEYS.

    MOVE SPACES TO OUT-TOY-RECORD.
    MOVE INV-NO OF IN-TOY-RECORD TO INVOICE-KEY.
    MOVE INV-NO OF IN-TOY-RECORD TO OUT-INV-NO.
    MOVE INV-QUANTITY OF IN-TOY-RECORD TO OUT-INV-QUANTITY.
    WRITE OUT-INV-RECORD
    KEYS ARE NAME-SELECTOR-KEY, NAME-KEY, INVOICE-KEY.

    GO TO WRITE-KEYS.

*************
*  STEP 5   *
*************

READ-DOWN.

    DISPLAY MENU-2.
    DISPLAY "Sequential Read :".
    DISPLAY "---------- ----".
    READ OUT-FILE
    FIX POSITION FORWARD
    SUPPRESS DATA RECORD
    READ OUT-FILE
    FIX POSITION DOWN.

READ-LEVEL1-KEYS.

    READ OUT-FILE
    FIX POSITION FORWARD
    AT END GO TO READ-PART-REC.
    DISPLAY OUT-TOY-RECORD.
    GO TO READ-LEVEL1-KEYS.
```

```
*************
*  STEP 6   *
*************

READ-PART-REC.

    READ OUT-FILE
    SUPPRESS DATA RECORD
    KEYS ARE NAME-SELECTOR-KEY, PART-KEY.
    DISPLAY " ".
    DISPLAY "Partial Record For Key Path : NAME,",PART-KEY," is ",
    PART-REC.
    DISPLAY " ".

READ-LEVEL2-KEY.

    MOVE 4950 TO NUMBER-KEY.
    READ OUT-FILE
    KEYS ARE NUMBER-SELECTOR-KEY, NUMBER-KEY, LEVEL2-KEY.
    DISPLAY "Invoice Record For Key Path : NUMBER,",NUMBER-KEY,",",
    LEVEL2-KEY," is ",OUT-INV-RECORD.
    DISPLAY " ".

READ-APPROXIMATE.

    READ OUT-FILE
    KEYS ARE NAME-SELECTOR-KEY, APPROX-KEY APPROXIMATE.
    DISPLAY "Read Approximate On Key Path : NAME,",APPROX-KEY.
    DISPLAY OUT-TOY-RECORD.
    DISPLAY " ".

READ-GENERIC.

    READ OUT-FILE
    KEYS ARE NAME-SELECTOR-KEY, GEN-KEY GENERIC.
    DISPLAY "Read Generic On Key Path : NAME,",GEN-KEY.
    DISPLAY OUT-TOY-RECORD.

*************
*  STEP 7   *
*************

CLOSE-FILES.

    CLOSE IN-FILE.
    CLOSE OUT-FILE.
```

```
*************************************************************************
*                                                                       *
*       Sample COBOL Program Using the INFOS II Interface               *
*                                                                       *
*      This program uses the INFOS interface to alter a toy file.       *
*                                                                       *
*************************************************************************


IDENTIFICATION DIVISION.
PROGRAM-ID.  SAMPLE2-COBOL.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT DBAM-FILE ASSIGN "DBAMX"
            ORGANIZATION IS INDEXED
            ACCESS MODE DYNAMIC
            RECORD KEYS ARE
                   NAME-KEY WITH DUPLICATES OCCURRENCE IS OCC-NUM
                   NUMBER-KEY
                   NAME-SELECTOR-KEY
                   NUMBER-SELECTOR-KEY
                   INVOICE-KEY
                   GEN-KEY
                   LOGIC-KEY
                   OLD-NAM-KEY
                   NEW-NAM-KEY
                   DEL-NAM-KEY
                   DEL-NUM-KEY
                   OLD-NUM-KEY
                   RETRIEVE-KEY
            FILE STATUS F-STAT
            INFOS STATUS I-STAT
            ALLOW SUB-INDEX
            LEVELS 3.

DATA DIVISION.
FILE SECTION.

FD  DBAM-FILE
    DATA RECORDS ARE
         OUT-TOY-RECORD
         OUT-INV-RECORD
    FEEDBACK IS REC-ADDR
    PARTIAL RECORD IS PART-REC.

01  OUT-TOY-RECORD.
    05  TOY-NAME            PIC X(20).
    05  TOY-NO              PIC 9(04).
    05  TOY-COMPANY         PIC X(20).
    05  TOY-QUANTITY        PIC 9(06).

01  OUT-INV-RECORD.
    05  OUT-INV-NO          PIC X(06).
    05  OUT-INV-QUANTITY    PIC 9(04).
```

```
WORKING-STORAGE SECTION.

01  NAME-SELECTOR-KEY      PIC X(04) VALUE "NAME".
01  NUMBER-SELECTOR-KEY    PIC X(06) VALUE "NUMBER".
01  NAME-KEY               PIC X(20).
01  NUMBER-KEY             PIC 9(04).
01  INVOICE-KEY            PIC X(06).
01  REC-ADDR               PIC X(04).
01  PART-REC               PIC 9(04).
01  GEN-KEY                PIC X(08) VALUE "MONOTONY".
01  REPLY                  PIC X.
01  I-STAT                 PIC X(11).
01  DEST1                  PIC X(20).
01  SUB1                   PIC 9(12).
01  DEST2                  PIC X(20).
01  STAT1                  PIC X(04).
01  LOGIC-KEY              PIC X(20) VALUE "TOSS-A-BOSS GAME".
01  F-STAT                 PIC X(02).
01  OLD-NAM-KEY            PIC X(20) VALUE "NERD BASKETBALL".
01  NEW-NAM-KEY            PIC X(20) VALUE "NERD FOOTBALL".
01  DEL-NAM-KEY            PIC X(20) VALUE "CONDEMNATION".
01  DEL-NUM-KEY            PIC 9(04) VALUE 4950.
01  OLD-NUM-KEY            PIC 9(04) VALUE 1840.
01  OCC-NUM                PIC 9(04).
01  RETRIEVE-KEY           PIC X(20) VALUE "SCRIBBLE".
01  SAVE-OCCURRENCE        PIC 9(04).


SCREEN SECTION.

01  MENU-1.
    05  BLANK SCREEN.
    05  LINE 10 COLUMN 15 BELL.
    05  "Read Generic On Key Path : NAME,".
    05  PIC X(08) FROM GEN-KEY.
    05  " Gives :".
    05  LINE PLUS 3 COLUMN 20.
    05  PIC X(11) FROM I-STAT.
    05  " (IOKPE) KEYED POSITIONING ERROR".
    05  LINE PLUS 10 COLUMN 40.
    05  "HIT NEW LINE TO CONTINUE".
    05  PIC X TO REPLY.
```

```
01  MENU-2.
    05  BLANK SCREEN BELL.
    05  LINE 5 COLUMN 10.
    05  "RETRIEVE KEY ON NAME-SELECTOR-KEY,RETRIEVE-KEY YIELDS : ".
    05  PIC X(20) FROM DEST1.
    05  LINE COLUMN 10.
    05  "OCCURRENCE NUMBER IS  ".
    05  PIC 9(04) FROM SAVE-OCCURRENCE.
    05  LINE PLUS 3 COLUMN 10.
    05  "RETRIEVE HIGH KEY ON NAME-SELECTOR-KEY'S SUBINDEX YIELDS : ".
    05  LINE PLUS 2 COLUMN 30.
    05  PIC X(20) FROM DEST2.
    05  LINE PLUS 3 COLUMN 10.
    05  "RETRIEVE STATUS ON HIGH KEY YIELDS : ".
    05  PIC X(04) FROM STAT1.
    05  LINE PLUS 2 COLUMN 10.
    05  "THE 1 IN THE LEFTMOST CHARACTER OF THE STATUS REPRESENTS A".
    05  LINE COLUMN 10.
    05  "LOGICAL LOCAL DELETE AND THE 1 IN THE RIGHTMOST CHARACTER".
    05  LINE COLUMN 10.
    05  "REPRESENTS A LOGICAL GLOBAL DELETE".
    05  LINE PLUS 5 COLUMN 40.
    05  "HIT NEW LINE TO CONTINUE".
    05  PIC X TO REPLY.


PROCEDURE DIVISION.

MAIN-PARAGRAPH.

*  The main paragraph is the control area.  It calls routines that
*  perform the tasks needed to complete the demonstration.

*  Perform STEP 8 -- Open the INFOS file.
    PERFORM OPEN-FILE.

*  Perform STEP 9 -- Do a generic READ on a key that does not exist
*                    (MONOTONY).
    PERFORM ERROR-HANDLING.

*  Perform STEP 10 -- Logically delete a record.
    PERFORM LOGICAL-DELETE.

*  Perform STEP 11 -- Retrieve information about a key.
    PERFORM RETRIEVE-COMMAND.

*  Perform STEP 12 -- Change the value of a key.
    PERFORM CHANGE-KEY.

*  Perform STEP 13 -- Delete a database record.
    PERFORM DELETE-RECORD.

*  Perform STEP 14 -- Close the INFOS file.
    PERFORM CLOSE-FILE.
    STOP RUN.
```

```
*************
*  STEP 8   *
*************

OPEN-FILE.

    OPEN I-O DBAM-FILE.


*************
*  STEP 9   *
*************

ERROR-HANDLING.

    READ DBAM-FILE
    KEYS ARE NAME-SELECTOR-KEY, GEN-KEY  GENERIC.
    IF I-STAT NOT = "00000000000" THEN
       DISPLAY MENU-1.
       ACCEPT MENU-1.

*************
*  STEP 10  *
*************

LOGICAL-DELETE.

    DELETE DBAM-FILE
    RECORD LOGICAL LOCAL GLOBAL
    KEYS ARE NAME-SELECTOR-KEY, LOGIC-KEY.

*************
*  STEP 11  *
*************

RETRIEVE-COMMAND.

    MOVE RETRIEVE-KEY TO NAME-KEY.
    RETRIEVE DBAM-FILE KEY
     KEYS ARE NAME-SELECTOR-KEY NAME-KEY INTO DEST1.
    MOVE OCC-NUM TO SAVE-OCCURRENCE.

    RETRIEVE DBAM-FILE HIGH KEY
    INTO DEST2.

    RETRIEVE DBAM-FILE STATUS
    INTO STAT1.

    DISPLAY MENU-2.
    ACCEPT MENU-2.
```

```
*************
*  STEP 12  *
*************

CHANGE-KEY.

    READ DBAM-FILE
    RECORD LOCK
    KEYS ARE NAME-SELECTOR-KEY,OLD-NAM-KEY.

    WRITE INVERTED OUT-TOY-RECORD
    KEYS ARE NAME-SELECTOR-KEY,NEW-NAM-KEY.

    LINK SUB-INDEX DBAM-FILE
    SOURCE
         KEYS ARE NAME-SELECTOR-KEY,OLD-NAM-KEY
    DESTINATION
         KEYS ARE NAME-SELECTOR-KEY,NEW-NAM-KEY.

    MOVE NEW-NAM-KEY TO TOY-NAME.
    REWRITE OUT-TOY-RECORD
    SUPPRESS PARTIAL RECORD
    KEYS ARE NAME-SELECTOR-KEY,NEW-NAM-KEY.

    EXPUNGE SUB-INDEX DBAM-FILE
    KEYS ARE NAME-SELECTOR-KEY,OLD-NAM-KEY.

    DELETE DBAM-FILE RECORD
    KEYS ARE NAME-SELECTOR-KEY,OLD-NAM-KEY.

*************
*  STEP 13  *
*************

DELETE-RECORD.

    EXPUNGE SUB-INDEX DBAM-FILE
    KEYS ARE NAME-SELECTOR-KEY,DEL-NAM-KEY.

    EXPUNGE SUB-INDEX DBAM-FILE
    KEYS ARE NUMBER-SELECTOR-KEY,DEL-NUM-KEY.

    DELETE DBAM-FILE RECORD
    KEYS ARE NAME-SELECTOR-KEY,DEL-NAM-KEY.

    DELETE DBAM-FILE RECORD
    KEYS ARE NUMBER-SELECTOR-KEY,DEL-NUM-KEY.

*************
*  STEP 14  *
*************

CLOSE-FILE.

    CLOSE DBAM-FILE.
```

End of Appendix

Appendix E
Standard ASCII Character Set

DECIMAL	OCTAL	HEX	KEY SYMBOL	MNEMONIC
0	000	00	↑@	NUL
1	001	01	↑A	SOH
2	002	02	↑B	STX
3	003	03	↑C	ETX
4	004	04	↑D	EOT
5	005	05	↑E	ENQ
6	006	06	↑F	ACK
7	007	07	↑G	BEL
8	010	08	↑H	BS (BACKSPACE)
9	011	09	↑I	TAB
10	012	0A	↑J	NEW LINE
11	013	0B	↑K	VT (VERT TAB)
12	014	0C	↑L	FORM FEED
13	015	0D	↑M	CARRIAGE RETURN
14	016	0E	↑N	SO
15	017	0F	↑O	SI
16	020	10	↑P	DLE
17	021	11	↑Q	DC1
18	022	12	↑R	DC2
19	023	13	↑S	DC3
20	024	14	↑T	DC4
21	025	15	↑U	NAK
22	026	16	↑V	SYN
23	027	17	↑W	ETB
24	030	18	↑X	CAN
25	031	19	↑Y	EM
26	032	1A	↑Z	SUB
27	033	1B	ESC	ESCAPE
28	034	1C	↑\	FS
29	035	1D	↑]	GS
30	036	1E	↑↑	RS
31	037	1F	↑—	US

DECIMAL	OCTAL	HEX	KEY SYMBOL
32	040	20	SPACE
33	041	21	!
34	042	22	" (QUOTE)
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	' (APOS)
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	, (COMMA)
45	055	2D	-
46	056	2E	. (PERIOD)
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?
64	100	40	@

DECIMAL	OCTAL	HEX	KEY SYMBOL
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	↑ OR ^
95	137	5F	← OR —
96	140	60	` (GRAVE)

DECIMAL	OCTAL	HEX	KEY SYMBOL
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~ (TILDE)
127	177	7F	DEL (RUBOUT)

DG-05495

End of Appendix

Appendix F
Operating the INFOS II Process

This appendix describes how to bring up the AOS INFOS II system. We have divided it into four sections:

- Operating environment requirements.
- How to initially install INFOS II into your operating environment.
- How to install a new release or update of INFOS II on your system.
- How to move your INFOS II work files — INFOS.VM and INFOS.COMM — to any directory you want.

Operating Environment Requirements

To use AOS INFOS II, you must have AOS and its utilities on your system. AOS must be running on one of the following computers:

- S-series ECLIPSE® computer with the character instruction set option.
- M/600 computer.
- ECLIPSE MV/8000® computer.
- C-series ECLIPSE computer.
- DESKTOP GENERATION™ computer.

If you intend to use INFOS II request logging (see Chapter 7), you must also have COMLOG and COMUNLOG. Finally, you must have RIA if you want to have remote access to INFOS II databases (see Chapter 11). See the latest AOS INFOS II Release Notice to determine which revision of each product you need.

Initially Installing INFOS II

If you are installing INFOS II on your system for the first time, follow the instructions in this section.

1. Issue the following command line, or edit your system's UP.CLI macro to contain it. Note that if you are going to use the Comprehensive Electronic Office (CEO®) software, you must bring up INFOS II *before* the CEO system.

   ```
   PROCESS/DEFAULT/DIRECTORY=@/NAME=INFOS_II INFOS_II.PR
   ```

2. If you want to start COMLOG at the same time you start up INFOS II, issue the following command or include it in the UP.CLI macro:

   ```
   PROCESS/DEFAULT/DIRECTORY=@/NAME=COMLOG COMLOG.PR
   ```

3. To terminate the INFOS II process, use the following command or edit your system's DOWN.CLI macro to include it. If you have the EXEC and global Idea process terminations in your DOWN.CLI macro, place the INFOS II termination after them. Also, if you use CEO, be sure to shut it down before you terminate INFOS II. Use the following command to terminate INFOS II:

   ```
   CONTROL @INFOS STOP
   ```

4. To terminate COMLOG, issue the following command line or include it in your DOWN.CLI macro after the INFOS II termination command:

   ```
   CONTROL @COMLOG STOP
   ```

Before you execute the DOWN.CLI macro, you must check to be sure that INFOS II has finished processing user requests. Checkpoint and IXLOAD processing and INFOS activity can continue after all users have terminated.

Installing a New Release or Update of INFOS II

Use the following directions to install a new version of INFOS II on your system.

1. Turn Superuser on.
2. Change your working directory to the root directory (:), and load the release tape. Use the following command line:

   ```
   LOAD/V/R @MTcx:0 #
   ```

   where c is the class of the tape drive you are using (e.g., A or B) and x is the tape unit number.

   This loads INFOS II into the root, and the utilities and accompanying software into the utilities directory (:UTIL).

3. Be sure to include the new INFOERMES.OB file in your system's :UTIL:NEWERMES.CLI macro. Execute the macro :UTIL:NEWERMES.CLI , which creates a new ERMES file in :UTIL.

   Move the new ERMES into the root directory to make it accessible to all users. Use the following command while in the :UTIL directory:

   ```
   MOVE/V/R : ERMES
   ```

   Either log off and log on again, or use the CLI command CHAIN, as follows:

   ```
   CHAIN :CLI.PR
   ```

   You will then be in the CLI with the new error messages. Check them by using the CLI MESSAGE command. For example:

   ```
   ) MESSAGE 7003 ↓
   ```

   The system will respond by telling you what revision of INFOS II your error messages came out with:

   ```
   7003 (IOINI) REVISION X.XX OF AOS INFOS II ERROR CODES
   ```

4. Optionally, you can load the INFOS II parameter files — PARAU.SR, IUMAC.SR, and PARIR.SR — and the INFOS II interface module, ICALL.OB, into the :UTIL directory, or elsewhere. Whether or not you want to do this depends on your own requirements. We recommend that you use the library file editor (LFE.PR) to insert ICALL.OB into the user runtime library (:UTIL:URT.LB) whenever you get a new version of ICALL.OB or URT.LB. This eliminates the need to include ICALL.OB on your command line when linking a program.

   ICALL.OB is not distributed in URT.LB, so be certain that you always have the correct version of ICALL.OB loaded on your system for the revision of INFOS II that you are running.

Relocating the INFOS II Work Files

You have the option of moving the INFOS II work files INFOS.VM and INFOS.COMM into any directory. If you want to do this, first *be sure* that INFOS II is not running. Then proceed as follows.

Create a link in the directory :PER to the pathname that you want the file to have. For example, if you want to move INFOS.VM to the directory :DTD, you would use the following command line:

```
CREATE/LINK :PER:INFOS.VM :DTD:INFOS.VM
```

You must still bring up INFOS II in :PER; however, INFOS II will follow the link and create INFOS.VM in the directory :DTD.

This option might be useful for systems with :PER on a fixed-head disk, or in situations where you could improve system performance by alleviating disk contention between INFOS.VM and other system files.

Note that :PER is deleted and recreated in the event of a system failure, so you should use a macro to create the link every time you bring the system up. For example, you might include the following lines in your UP.CLI macro:

```
CREATE/LINK/1=WARN <:PER,:DTD>:INFOS.(VM,COMM)
PROCESS/DEFAULT/DIRECTORY=@/NAME=INFOS_II :INFOS_II.PR
```

(The information in this appendix was current when this manual was printed. However, be sure to check your Release Notice for any changes concerning the operation of INFOS II.)

End of Appendix

Appendix G
Performance Tips

The performance of the INFOS II system is affected by many individual environmental characteristics. The following tips might help you to improve your system performance.

Process Type

Running INFOS II as a resident process, rather than a pre-emptible or swappable process, might help INFOS II application environments. If you do so, then you tell AOS that INFOS II is a highest memory contention process. INFOS II will always reside in main memory.

Page Size

Small page files generally provide better performance than large page files. Therefore, you might improve your system performance if you make your index files small page, and your database files small page unless you have data records larger than 2040 bytes.

Volume Element Size

You can create each of your INFOS II files with at least two volumes of differing element sizes. If you choose your first volume to have a large element size, the operating system will not have to allocate elements as often. Pages of the volume are more likely to be contiguous, and less time will be spent seeking across the disk.

You can then create at least one other volume with a small element size, so that when the large contiguous elements of the first volume cannot be allocated, due to disk fragmentation or lack of free space, your file can still grow.

Locks

With some programming languages you can specify when opening a file the maximum number of outstanding locks that you intend to use. If your language supports this, you might want to specify the lowest number you can in the parameter ?FLNR in the index file definition packet. You should also note that the record locking algorithm of INFOS II optimizes the special case of a file with no locks set, regardless of how you open the file.

Number of Opens

Minimizing the number of opens to a database, and the number of open channels in general, could improve performance. Three tips apply in this respect:

- Merging related files into one large database can help performance, especially when index root nodes reside on the same page.
- Using a single database rather than multiple smaller ones might allow INFOS II requests to be completed more quickly.
- Merging existing databases into two databases — one read-only, and the other read-write — could improve performance. You can process the read-only database in Standard File Mode for maximum performance, and you will not have to run the IVERIFY utility against it in the event of a system failure. For the read-write database, you can use one of the processing modes that provides recovery features.

Recovery Options

There are three things you might want to keep in mind when considering a recovery option with respect to performance.

- Differential File (DF) Mode generally provides better performance than Modified Page Flush (MPF), due to the expense of gaining access to the disk. Since a system failure can disrupt your MPF file, but it does not disrupt your file in DF mode, you might find better performance with DF mode than with MPF.
- If your application requires that you are able to recover the last INFOS request made, you might decide to use MPF or request logging as your recovery option. Which one you choose will probably be based on measurements of performance for applications in each mode. You should also consider the possibility that MPF files might not be recoverable.
- If you choose the request logging feature, buffered logging tends to provide better performance than immediate logging. If you do choose buffered logging, maximizing the COMLOG buffer size can also improve your performance. See the COMLOG documentation for information on how to specify a nondefault buffer size.

End of Appendix

Index

Within this index, "f" or "ff" after a page number means "and the following page" (or "pages"). Commands, calls, and acronyms are in uppercase letters (e.g., CREATE); all others are lowercase.

B

C

D

F

M

N

O

Q

R

CUT ALONG DOTTED LINE

Data General

users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
MV/Series ECLIPSE®		
Commercial ECLIPSE		
Scientific ECLIPSE		
Array Processors		
CS Series		
NOVA®4 Family		
Other NOVAs		
microNOVA® Family		
MPT Family		
Other (Specify) _____		

3. Software

- ☐ AOS
- ☐ AOS/VS
- ☐ AOS/RT32
- ☐ MP/OS
- ☐ MP/AOS
- ☐ RDOS
- ☐ DOS
- ☐ RTOS
- ☐ Other

Specify _____

4. Languages

- ☐ ALGOL
- ☐ DG/L
- ☐ COBOL
- ☐ Interactive COBOL
- ☐ PASCAL
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ FORTRAN 77
- ☐ FORTRAN 5
- ☐ RPG II
- ☐ PL/1
- ☐ APL
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communication

- ☐ HASP
- ☐ HASP II
- ☐ RJE80
- ☐ RCX 70
- ☐ RSTCP
- ☐ 4025
- ☐ X.25
- ☐ SAM
- ☐ CAM
- ☐ XODIAC™
- ☐ DG/SNA
- ☐ 3270
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

Data General

Data General Corporation, Westboro, Massachusetts 01580, (617) 366-8911

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

Data General

TP_____

TIPS ORDER FORM
Technical Information & Publications Service

BILL TO:
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

SHIP TO: (if different)
COMPANY NAME_____
ADDRESS _____
CITY _____
STATE_____ ZIP _____
ATTN: _____

QTY	MODEL #	DESCRIPTION	UNIT PRICE	LINE DISC	TOTAL PRICE

(Additional items can be included on second order form)

[Minimum order is $50.00]

Tax Exempt #_____
or Sales Tax (if applicable)

TOTAL	
Sales Tax	
Shipping	
TOTAL	

METHOD OF PAYMENT

☐ Check or money order enclosed
For orders less than $100.00

☐ Charge my ☐ Visa ☐ MasterCard
Acc't No._____ Expiration Date_____

☐ Purchase Order Number:_____

SHIP VIA

☐ DGC will select best way (U.P.S or Postal)

☐ Other:
☐ U.P.S. Blue Label
☐ Air Freight
☐ Other _____
_____

NOTE: ORDERS LESS THAN $100, INCLUDE $5.00 FOR SHIPPING AND HANDLING.

Person to contact about this order _____ Phone _____ Extension _____

Mail Orders to:

Data General Corporation
Attn: Educational Services/TIPS F019
4400 Computer Drive
Westboro, MA 01580
Tel. (617) 366-8911 ext. 4032

Buyer's Authorized Signature Date
(agrees to terms & conditions on reverse side)

Title

DGC Sales Representative (If Known) Badge #

DISCOUNTS APPLY TO MAIL ORDERS ONLY

012-1780

educational services

CUT ALONG DOTTED LINE

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form shown on the reverse hereof which is accepted by DGC.

1. **PRICES**
   Prices for DGC publications will be as stated in the Educational Services Literature Catalog in effect at the time DGC accepts Buyer's order or as specified on an authorized DGC quotation in force at the time of receipt by DGC of the Order Form shown on the reverse hereof. Prices are exclusive of all excise, sales, use or similar taxes and, therefore are subject to an increase equal in amount to any tax DGC may be required to collect or pay on the sale, license or delivery of the materials provided hereunder.

2. **PAYMENT**
   Terms are net cash on or prior to delivery except where satisfactory open account credit is established, in which case terms are net thirty (30) days from date of invoice.

3. **SHIPMENT**
   Shipment will be made F.O.B. Point of Origin. DGC normally ships either by UPS or U.S. Mail or other appropriate method depending upon weight, unless Customer designates a specific method and/or carrier on the Order Form. In any case, DGC assumes no liability with regard to loss, damage or delay during shipment.

4. **TERM**
   Upon execution by Buyer and acceptance by DGC, this agreement shall continue to remain in effect until terminated by either party upon thirty (30) days prior written notice. It is the intent of the parties to leave this Agreement in effect so that all subsequent orders for DGC publications will be governed by the terms and conditions of this Agreement.

5. **CUSTOMER CERTIFICATION**
   Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

6. **DATA AND PROPRIETARY RIGHTS**
   Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

7. **DISCLAIMER OF WARRANTY**
   DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS SUPPLIED HEREUNDER.

8. **LIMITATIONS OF LIABILITY**
   IN NO EVENT SHALL DGC BE LIABLE FOR (I) ANY COSTS, DAMAGES OR EXPENSES ARISING OUT OF OR IN CONNECTION WITH ANY CLAIM BY ANY PERSON THAT USE OF THE PUBLICATION OF INFORMATION CONTAINED THEREIN INFRINGES ANY COPYRIGHT OR TRADE SECRET RIGHT OR (II) ANY INCIDENTIAL, SPECIAL, DIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOSS OF DATA, PROGRAMS OR LOST PROFITS.

9. **GENERAL**
   A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer.

DISCOUNT SCHEDULES

DISCOUNTS APPLY TO MAIL ORDERS ONLY.

LINE ITEM DISCOUNT

5-14 manuals of the same part number - 20%
15 or more manuals of the same part number - 30%

DISCOUNTS APPLY TO PRICES SHOWN IN THE CURRENT TIPS CATALOG ONLY.

Data General

TIPS ORDERING PROCEDURE:

Technical literature may be ordered through the Customer Education Service's Technical Information and Publications Service (TIPS).

1. Turn to the TIPS Order Form.
2. Fill in the requested information. If you need more space to list the items you are ordering, use an additional form. Transfer the subtotal from any additional sheet to the space marked "subtotal" on the form.
3. Do not forget to include your MAIL ORDER ONLY discount. (See discount schedules on the back of the TIPS Order Form.)
4. Total your order. (MINIMUM ORDER/CHARGE after discounts of $50.00.)

   If your order totals less than 100.00, enclose a certified check or money order for the total (include sales tax, or your tax exempt number, if applicable) plus $5.00 for shipping and handling.
5. Please indicate on the Order Form if you have any special shipping requirements. Unless specified, orders are normally shipped U.P.S.
6. Read carefully the terms and conditions of the TIPS program on the reverse side of the Order Form.
7. Sign on the line provided on the form and enclose with payment. Mail to:

   TIPS
   Educational Services – M.S. F019
   Data General Corporation
   4400 Computer Drive
   Westboro, MA 01580
8. We'll take care of the rest!

educational services

Data General Corporation, Westboro, MA 01580

093-000152-02