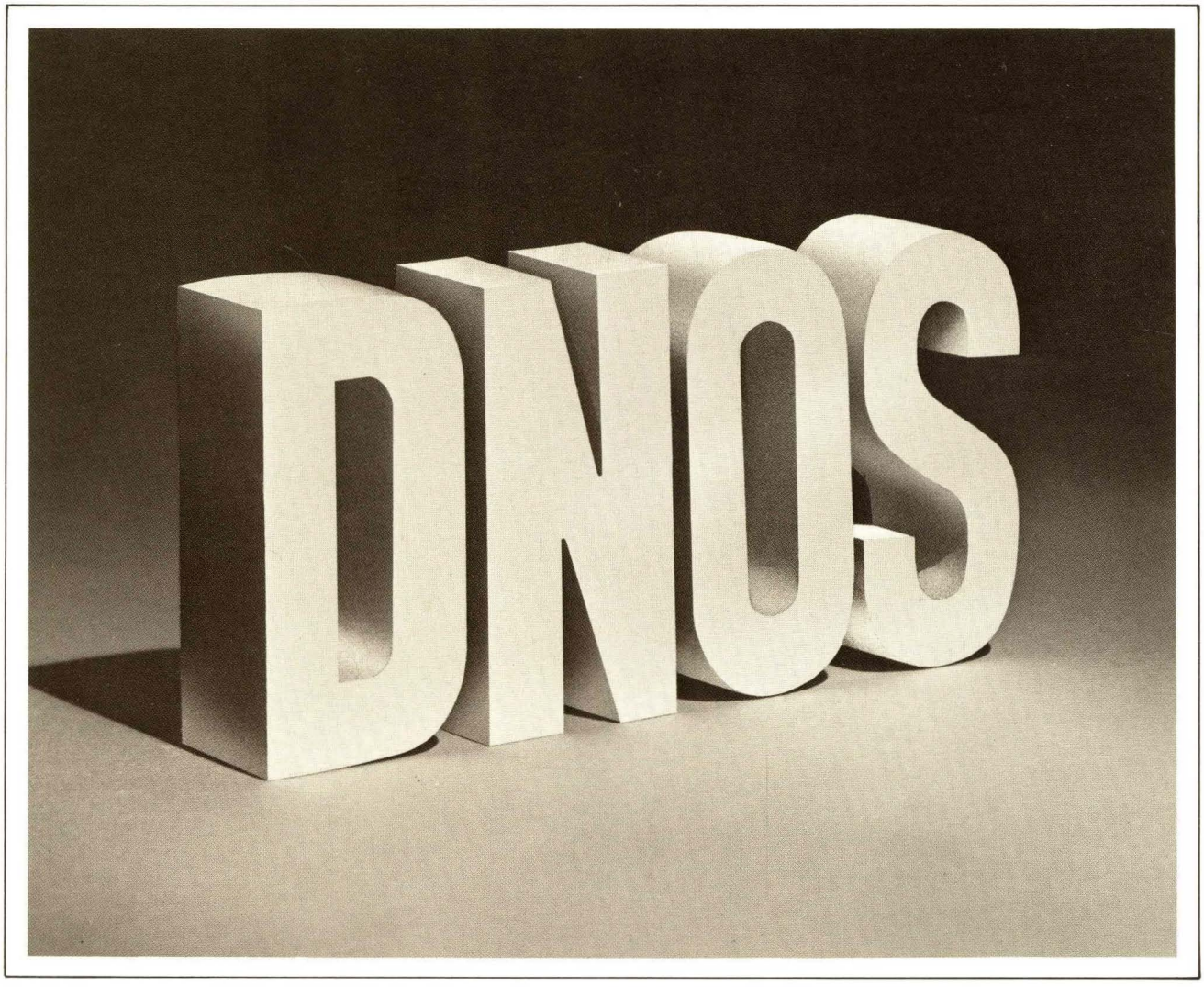
Model 990 Computer
DNOS Data Base Management System
Programmer's Guide



Part No. 2272058-9701 *A
15 July 1982

TEXAS INSTRUMENTS
INCORPORATED

MANUAL REVISION HISTORY

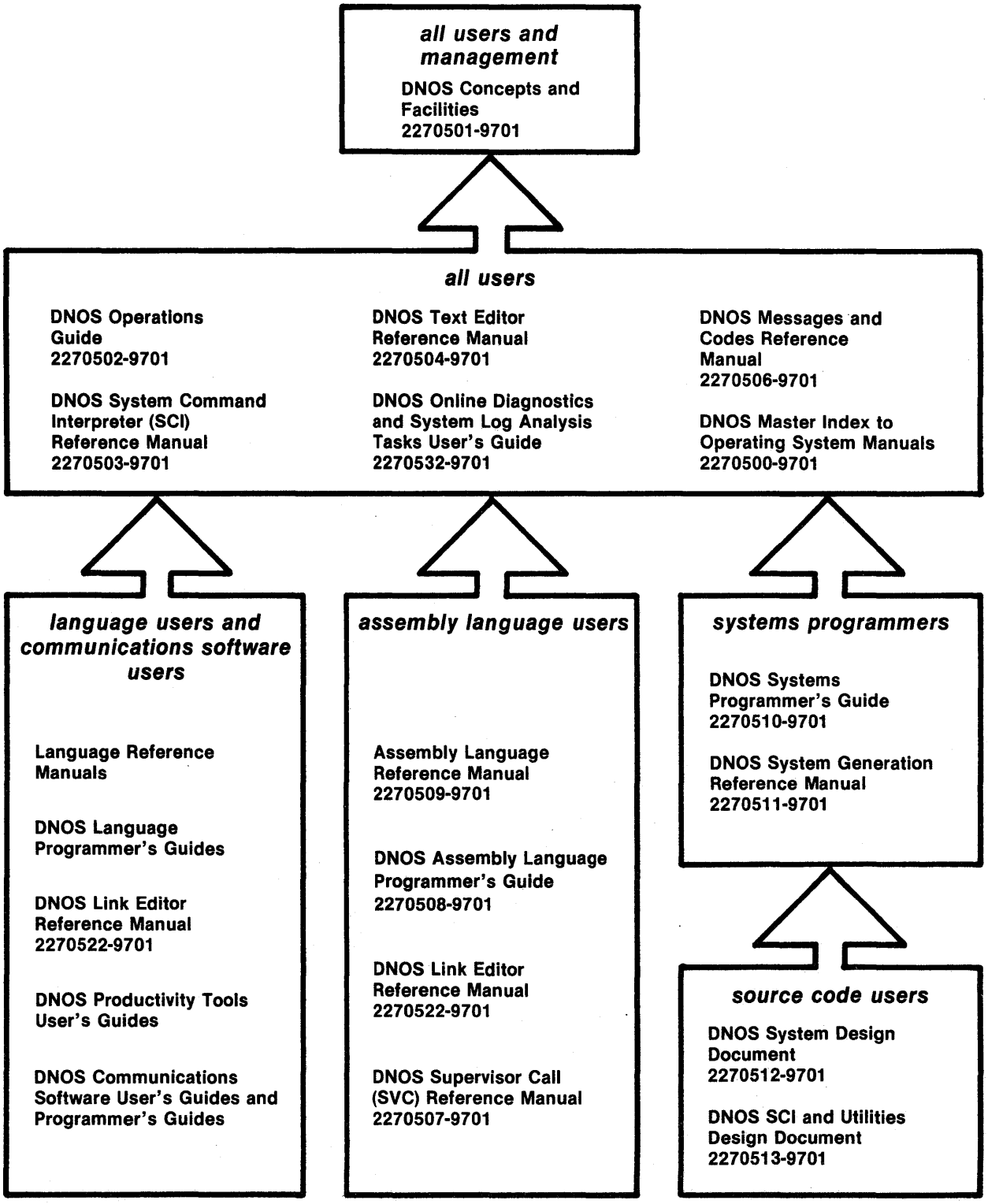
Model 990 Computer DNOS Data Base Management System Programmer's Guide (2272058-9701)

Original Issue . . . . . 1 August 1981
Revision . . . . . 15 July 1982

The total number of pages in this publication is 172.

DNOS
Distributed Network Operating System Software Manuals

The manuals supporting DNOS are arranged in this diagram according to the type of user. The manuals most beneficial to your needs are those contained in the block identified as your user group and in all the blocks above that set.



2280078

DNOS

Distributed Network Operating System Software Manuals Summary

Concepts and Facilities
Presents an overview of DNOS with topics grouped into functions of the operating system. All new users (or evaluators) of DNOS should read this manual.

Operations Guide
Provides the information necessary to perform daily tasks at a TI 990 Computer installation using DNOS. Step-by-step procedures are presented for such tasks as operating peripherals, initializing the system, backing up the system, and manipulating disk files.

System Command Interpreter (SCI) Reference Manual
Describes how to use SCI in both interactive and batch jobs. Command procedures and primitives are described, followed by a detailed presentation of all SCI commands in alphabetical order for easy reference.

Text Editor Reference Manual
Shows how to use the Text Editor interactively on DNOS and includes a detailed description of each of the editing commands and function keys.

Messages and Codes Reference Manual
Lists the error messages, informative messages, and error codes reported by DNOS.

Online Diagnostics and System Log Analysis Tasks User's Guide
Provides the information necessary to execute the online diagnostic tasks and the system log analysis tasks and to interpret the results.

Master Index to Operating System Manuals
Contains a composite index to topics in the DNOS operating system manuals.

Programmer's Guides and Reference Manuals for Languages
Each programmer's guide describes one of the languages supported by DNOS (for example, assembly language, Pascal, COBOL). Each guide covers operating system information relevant to the use of that language in the DNOS environment. The details of the language itself, including language syntax and programming considerations, are in the language reference manual.

Link Editor Reference Manual
Describes how to use the Link Editor on DNOS to combine separately generated object modules to form a single linked output.

User's Guides for Productivity Tools
Each user's guide describes one of the productivity tools (for example, TIFORM, Query-990, DBMS-990, Sort/Merge) supported by DNOS. Each guide explains the function of the processor, its features, and its interface requirements.

User's Guides and Programmer's Guides for Communications Software
Describe the features, functions, and use of the communications software available for execution under DNOS. For example, there is a user's guide for the DNOS 3780/2780 Emulator software package.

Supervisor Call (SVC) Reference Manual
Presents detailed information about each DNOS supervisor call and general information about DNOS services.

Systems Programmer's Guide
Discusses the DNOS nucleus and subsystems at a conceptual and functional level and describes how to modify the system for a specific application environment.

System Generation Reference Manual
Contains the information needed to perform system generation, including pregeneration requirements, generation procedures, and information about postgeneration results.

System Design Document
Contains the information needed to understand the functioning of the system when using a source kit. This includes descriptions of the subsystems in detail, naming and coding conventions, module cross-references, data structure details, and information not found in other manuals.

SCI and Utilities Design Document
Presents design information about SCI and the DNOS utilities.

Preface

This manual is intended for the programmer and the data base user. It provides information for defining data elements and for writing and testing programs for use with the Texas Instruments data base management system, DBMS-990. The data base administrator (DBA) oversees the data base operation, maintains the system, manages the security assignments and the security system, and assists in the design and operation of the data base. For further details, consult the *Model 990 Computer DNOS Data Base Administrator User's Guide.*

Certain symbols appear in the instruction definitions in this manual. Brackets ([ ]) signify that you can omit an item. Angle brackets (< >) indicate that the appropriate user-defined item is required. Braces ({ }) indicate that you must choose one of the enclosed items.

This manual is organized into the following sections and appendixes:

Section

1 General Description — Provides a general description of DBMS-990.

2 Data Base Elements — Describes the elements of the data hierarchy.

3 Data Definition Language (DDL) — Describes the statements in a DDL declaration and presents example DDL procedures, and error messages.

4 Data Manipulation Language (DML) — Defines each call parameter and each DML function code.

5 Security — Discusses passwords and access authorization.

6 Primitive Query — Describes primitive query functions and presents example queries and error messages.

7 Execution of Application Programs — Discusses preliminary procedures, program techniques, compiling and linking, and program testing.

Appendix

A DBMS Exception Reporting — Explains DBMS-990 status exception codes.

B Example DBMS Programs — Provides an example DBMS application written in COBOL, Pascal, and FORTRAN.

In addition to the DNOS manuals shown on the frontispiece, the following documents contain information related to this manual:

Title	Part Number
Model 990 Computer DNOS COBOL Programmer's Guide	2270516-9701
Model 990 Computer DNOS TI Pascal Programmer's Guide	2270517-9701
Model 990 Computer FORTRAN Programmer's Reference Manual	946260-9701
Model 990 Computer DNOS Data Base Administrator User's Guide	2272059-9701
Model 990 Computer DNOS Query-990 User's Guide	2276554-9701

Contents

4 — Data Manipulation Language (DML)

5 — Security

6 — Primitive Query

7 — Execution of Application Programs

Appendixes

Illustrations

Tables

1

General Description

1.1 INTRODUCTION

DBMS-990 is the data base management system (DBMS) that operates under the DNOS operating system on the Texas Instruments Model 990 Computer. Although DBMS-990 is a general-purpose data manager, you establish the content, grouping, relationship, and security of all data elements. DBMS-990 is easy to use and provides you with a logical viewpoint of the data. Normal physical constraints such as access method, record size, block size, and relative field position should not concern you.

The primary purpose of DBMS-990 is to provide a mechanism for organizing, storing, updating, and retrieving data through mass-storage devices. The 990 computer's mass-storage media is disk, which facilitates the use of random-access techniques.

This section provides an overview of the data base elements, data definition language (DDL), data manipulation language (DML), security, primitive query, file-access checking, and backup logging.

Figure 1-1 shows the various components of DBMS-990. The nucleus performs the actual manipulation of data. Security and logging of updated data are optional features of DBMS-990. The interface module is the communication link between user programs and the nucleus. The DML is embedded in each user program to facilitate this communication. The DDL and the DDL compiler enable you to define the data base file(s). Utilities communicate with the interface module to perform various maintenance tasks, including copying and restoring data base files. Primitive query also communicates with the interface module to enable you to inquire and display data in a limited manner. The various components make DBMS-990 an integrated system. The components that communicate with the interface module work indirectly with the nucleus. If security and backup logging are installed in your system, you must consider their effects on DBMS-990 and the data base. The DBMS-990 application programming languages are COBOL, FORTRAN, and Pascal. These programs contain the DML functions.

1.2 DATA BASE ELEMENTS

Data base elements consist of the data hierarchy and keys. The data hierarchy contains the logical data elements; keys allow access to the data.

1.2.1 Data Hierarchy

The DBMS-990 data hierarchy is oriented toward business documents such as invoices, purchase orders, and sales orders. A document is the basic means of initiating, executing, and recording business transactions. Accordingly, the document concept facilitates the automation of business activities and the development of the required computer software.

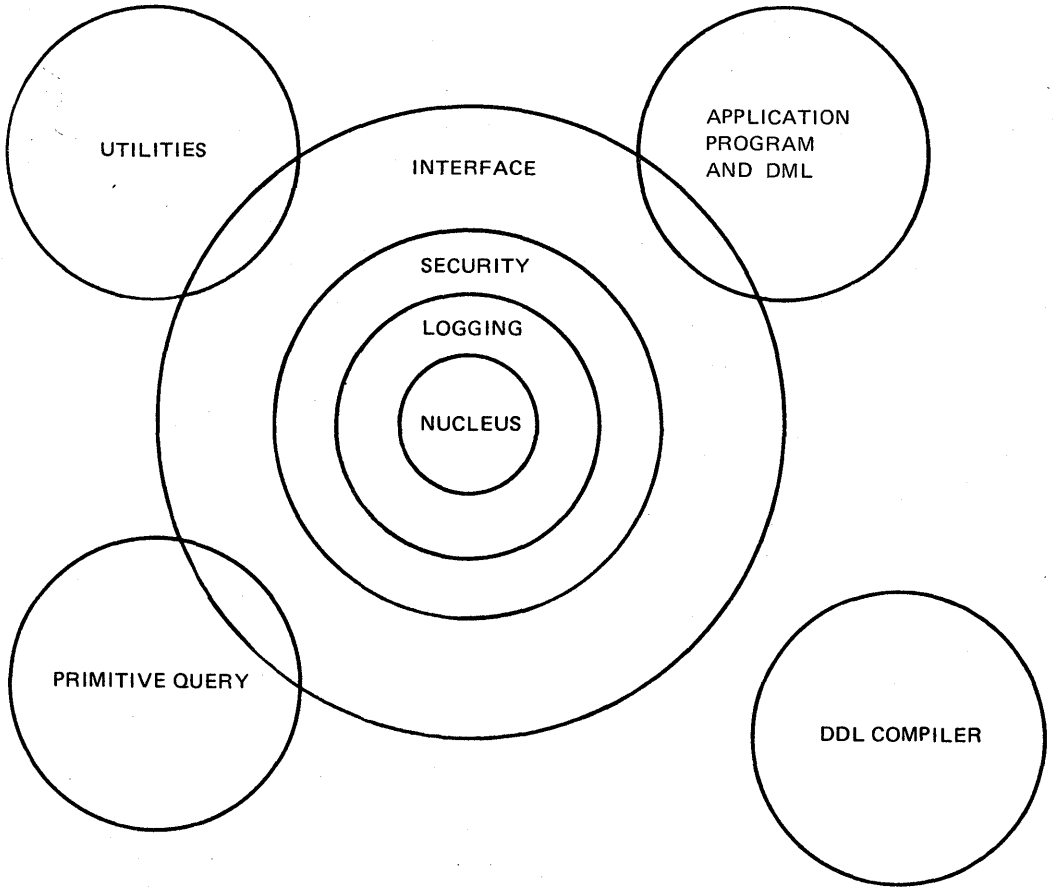


Figure 1-1. DBMS-990 Components

As Figure 1-2 shows, a DBMS-990 data file corresponds to a file contained in a filing cabinet, and a data record corresponds to the document that first contained the information. A document contains lines of information that describe the business transaction. The line in a DBMS-990 data file corresponds to a line in a document. Thus, the order of the hierarchy is as follows: a file contains records, records contain lines, and lines contain groups of fields and/or individual fields.

1.2.2 Keys

A key identifies a certain data element to facilitate rapid access. The two types of keys in DBMS-990 are primary and secondary. Primary keys identify records, while secondary keys identify lines. Figure 1-3 shows how a primary key (in this case, the invoice number) fits into a data base structure.

1.3 DATA RETRIEVAL METHODS

DBMS-990 provides both random and sequential methods of storing and retrieving data. When defining a key, the user declares its storage method as either random or sequential. With the sequential method, DBMS orders the key values sequentially and then maintains that order. Thus, you can declare a key as sequential, add and delete data, and then retrieve the data in sorted order.

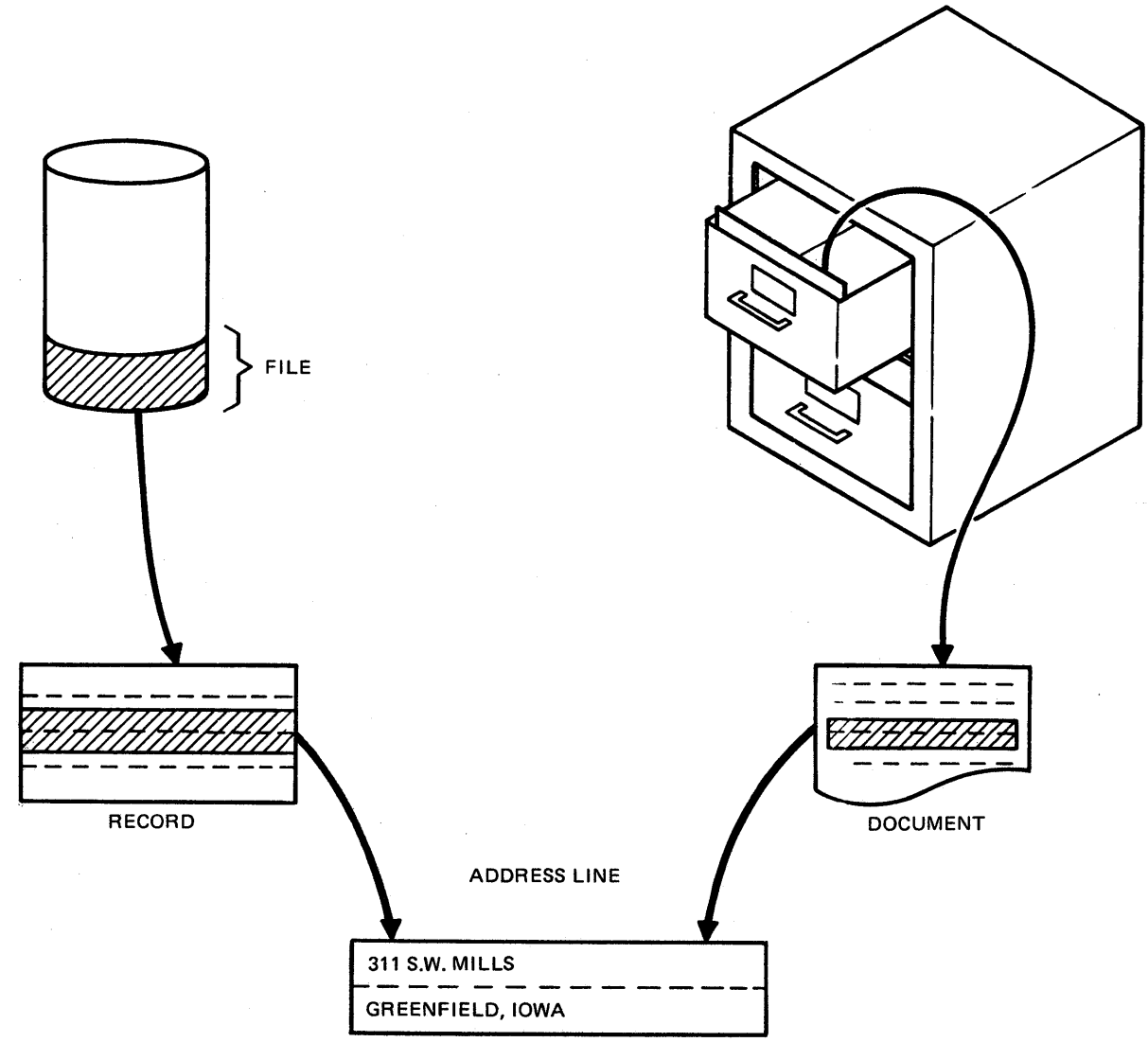
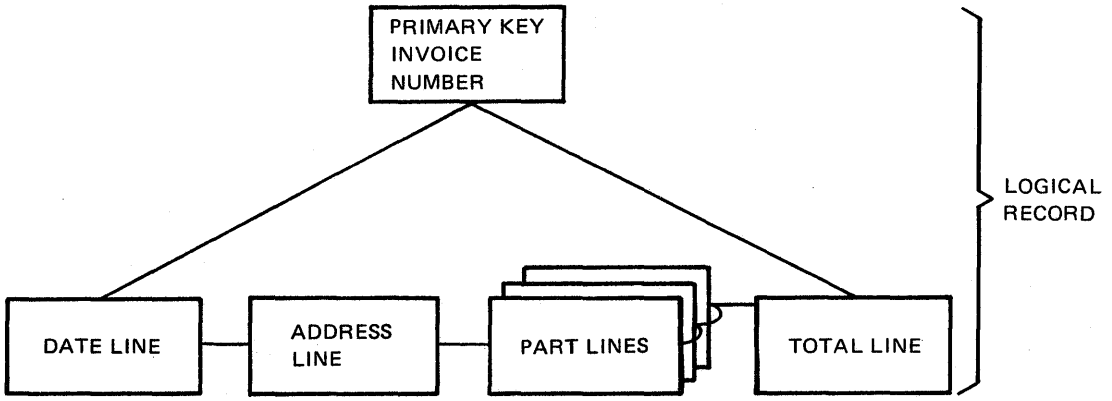


Figure 1-2. Example of File, Document, and Line Correlation

1.4 DATA DEFINITION LANGUAGE (DDL)

The DDL enables you to define or assign the data elements within the data base. Before any processing can take place on the data base, you must perform a DDL operation. Use the DDL to define file IDs; key IDs; retrieval methods; and the number of keys, lines, groups, and fields. When programming, you need be only minimally concerned with file boundaries and the position of data; most of these considerations are handled at the data definition level. Knowledge of the data position within the file is not required since data requests are made by field or group ID. As long as the field ID, format, and length do not change, the application programs do not need modification.



2277676

Figure 1-3. Example of a Data Structure

1.5 DATA MANIPULATION LANGUAGE (DML)

The DML enables you to read, replace, add, and delete data. The DML is not a complete language; it is a set of function codes passed to DBMS-990 through calls embedded in an application language.

When using the DML, specify only the field and/or groups to be transferred. You need not know the exact position of the data, only the declared ID and line type.

1.6 SECURITY

Security is an optional feature of DBMS-990. You can choose to include security during installation of DBMS-990. Security limits unauthorized use of the data base but requires a certain amount of overhead. The actual overhead involved depends on the degree of protection assigned to the data elements in the data base.

To retrieve data from a file, the password provided in the request to DBMS-990 must be associated with the file and/or data base. Each password is associated with one or more files. Each file associated with a password requires an access authorization code. You can assign access authorization to lines, groups, and fields. Lower-level data elements, such as lines and fields, assume the authorization of the next higher data element. For example, a line that does not have an authorization code assumes the code of the record. However, you can assign less authorization to the line. The same applies to groups and fields.

1.7 PRIMITIVE QUERY

Primitive query enables you to retrieve and display data base information without writing a program. Three functions are provided for limited retrieval and data display: read forward, read backward, and read serial. The display follows the format defined in the DDL.

1.8 FILE-ACCESS CHECKING

File-access checking is an optional feature of DBMS-990 and is installed at generation time. File-access checking involves three types of access:

- Shared access — All users have all access privileges to the file.
- Exclusive access — Only the current user has all access privileges.
- Read-only exclusive access — All users have only read access privileges to the file. File-access checking monitors the current status to the requested access, checking for incompatibilities, and returns appropriate error conditions when incompatibility exists.

1.9 BACKUP LOGGING

Backup logging is an optional feature of DBMS-990; it automatically records successful updates in an interactive environment. Updates include adds, replaces, and deletes.

1.10 TRANSACTION-LEVEL INTEGRITY

Transaction-level integrity is an optional feature that may be selected by the DBA at Data Base Generation (DBGEN) time. It allows you to define a series of operations as a transaction. By utilizing the transaction-level integrity feature, a programmer is able to require that all operations within the defined boundaries of the transaction be performed successfully or, if one operation cannot be performed, that the data base be restored to its pretransaction state.

In the event of a system crash, all transactions in progress are rolled back, restoring the data base to its pretransaction state. This relieves you of the need to manually examine the contents of the file records in order to verify where processing was interrupted. Automatic rollback of all transactions in progress at the time of a system crash is a feature of transaction-level integrity.

1.10.1 How Transaction-Level Integrity Protects Your Data Base

The advantage of transaction-level integrity in preventing erroneous changes to the data base is illustrated in the following example.

A bank customer has a savings account and a checking account and wishes to make a transfer of $1000 from savings to checking. Two data base operations must be performed to complete the transfer, as follows:

1. Subtract $1000 from savings.
2. Add $1000 to checking.

In this sequence, steps 1 and 2 comprise a transaction. That is, unless both operations are completed, you do not want either operation applied to the data base.

Without transaction-level integrity, each operation stands alone. If a system crash were to occur after step 1 and before step 2, the data base would contain an inaccurate entry. In this simple example, you could possibly recover by examining the contents of the data base and making the necessary changes. However, in more complicated transactions, the recovery may be considerably more complicated.

With the transaction-level integrity option, the fact that the two operations are defined as a transaction ensures that neither operation is applied or both are applied. This ensures that the data base never contains erroneous data.

1.10.2 Locking Protocol

In order to prevent two users from accessing the same data base line during the course of two separate transactions, transaction-level integrity employs a system of locks. A line becomes locked whenever a read or write operation involving that line occurs in a transaction. Whenever an add or delete is performed, the entire record is locked.

There are two levels of locking: shared lock and exclusive lock. Under shared lock, the data is available for reading but cannot be modified. Under exclusive lock, the data can neither be read nor modified by transactions other than the transaction with the exclusive lock. Table 1-1 diagrams the locking protocol.

When a lock is changed from a shared lock to an exclusive lock, it is said to be upgraded. The function for upgrading is hold line (HL), as shown in Figure 1-4.

Upgrading of a user's lock (from shared to exclusive) can only occur when no other user has a shared lock on that line. If a transaction cannot be upgraded because another user also has a shared lock on the line, a delay occurs in the transaction requesting the upgrade until one of the following conditions occurs:

- The competing transaction releases its shared lock on the line, at which point the delayed transaction is allowed to proceed.
- The second transaction also requests an exclusive lock, and a deadlock occurs.

In the second case, the system identifies the deadlock and gives the exclusive lock to the transaction that first requested the exclusive lock. The second transaction is denied access and rolled back. This automatic resolution of deadlock prevents two transactions from waiting on each other indefinitely.

Table 1-1. Locking Protocol

Type of Lock	First User Access	Other User Access
Shared	Read Only	Read Only
Exclusive	Read/Write	Delayed

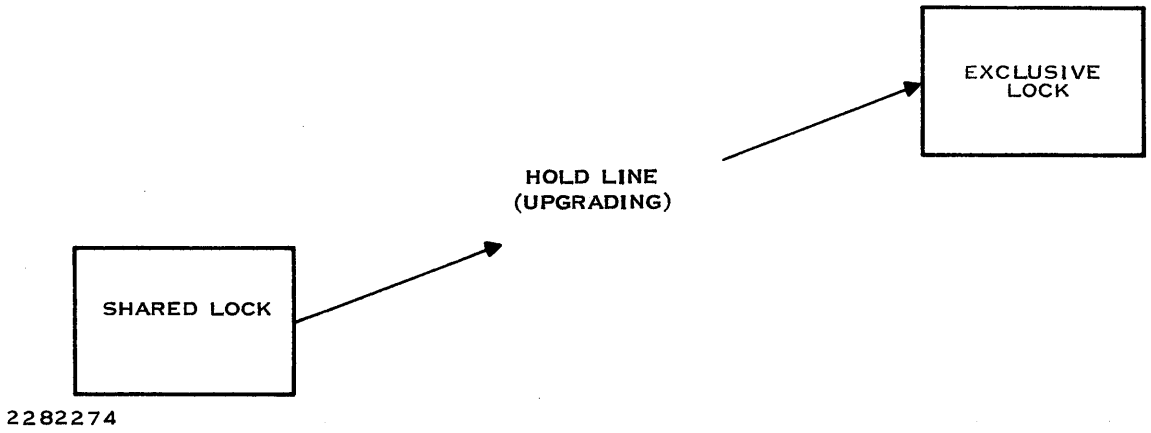


Figure 1-4. Upgrading a Lock

1.10.3 Deadlock

The importance of the deadlock resolution feature is illustrated in the following example of its use in a multiuser environment. Table 1-2 summarizes the steps in the example.

In the example, two travel agents are each servicing a customer. Both customers desire three seats on the same airline flight. They initially want to know if three seats are available. The procedures that each agent executes are as follows:

1. Each agent starts a transaction by making an inquiry to the data base. The application program would perform a start transaction (TS) and a read with release on the same record for each agent. Both agents have access to the same record. The record, therefore, is in a shared lock state. Neither agent can write to the record while the record is in this state.

2. The customer working with Agent A decides that he wants the three seats that are available. Agent A issues a reservation request. The application program attempts to perform a hold line function to upgrade the record lock from shared to exclusive.

3. The system delays Agent A's transaction due to the conflict with Agent B's shared lock on the record. Agent A's transaction is waiting on Agent B's transaction.

4. The customer working with Agent B decides that he wants the three seats also. Agent B issues a reservation request. The DML program attempts to perform a hold line function to upgrade the record lock from shared to exclusive. The two transactions are now waiting on each other. A's transaction is delayed waiting for B's transaction to release the shared lock. At the same time, B is waiting for A's transaction to release its lock. This condition is called deadlock.

5. The system identifies the deadlock and resolves it as follows:
   a. Rolls back Agent B's transaction, thereby releasing Agent B's shared lock
   b. Returns a deadlock status message to Agent B's transaction
   c. Grants Agent A's transaction the exclusive lock it needs
6. Agent A's transaction is committed to the data base.

The deadlock message returned to Agent B's transaction can be used by the programmer to restart Agent B's transaction or output a message to Agent B's terminal informing him of the conflict and instructing him to restart the transaction.

Table 1-2. Illustration of Deadlock

	Agent A		Agent B		
	User Action	**Program Function**	**User Action**	**Program Function**	**System Status**
1.	Makes inquiry	Start transaction (TS) Read with release	Makes inquiry	Start transaction (TS) Read with release	Shared lock on record
2.	Requests 3 seats	Hold line (HL)			Conflict (A is waiting on B; system cannot grant exclusive lock)
3.	Waits	Task delayed			
4.			Requests 3 seats	Hold line (HL)	Deadlock (B is waiting on A; A is waiting on B)
5.				Rollback transaction (TR) DL status returned	System: a. Rolls back B's transaction b. Returns DL status to B's transaction c. Grants A's transaction exclusive lock
6.	Receives 3 seats	Commit transaction (TC)			

1.10.4 Advantages of Transaction-Level Integrity

Transaction-level integrity allows the programmer to define a series of data base operations as a transaction. In doing so, the programmer can require that the system execute all operations within the transaction successfully or, if one or more operations cannot be performed, roll back any data base modifications made as part of the transaction.

DBMS-990 allows transaction nesting up to a maximum level of 10. The level of nesting permitted is specified at DBGEN.

The transaction-level integrity feature simplifies the programmer's work in that the system performs the following functions without programmer effort:

- Ensures that all conditions necessary for the success of a defined transaction are met before making a permanent change to the data base
- Returns an indication of a deadlock that the programmer can use to cause a restart of the transaction or abort the transaction and display a message to the user indicating a problem with the transaction
- Locks all data lines involved in a transaction until the transaction is either committed to the data base or rolled back
- Ensures that the outcome of any transaction operating in a concurrent environment is identical to that obtained by running the transaction by itself

Remember that there is a performance cost associated with the use of the transaction-level integrity option. System response time is sacrificed and additional memory space is required in return for a more secure data base. It is recommended that you specify small parameter values when first using the transaction-level integrity feature and increase the values if experience demonstrates that deadlock occurs too frequently.

1.10.5 Troubleshooting Deadlock

Any one of the following conditions can cause a deadlock:

- Two or more transactions request a lock on the same record.
- The system lock table is full. Too many locks are in use for the lock table to accommodate them all.
- A transaction contains too many updates.
- The primary keys of one or more users are longer than the maximum length specified at DBGEN time.
- More users are on the system than were specified at DBGEN time.

If frequent occurrence of deadlock is a problem, you should determine the principle reason for deadlock. If the Data Base Statistics (DBSTAT) option is enabled, you can do this by executing the DBSTAT command and examining the statistics. Refer to the *Model 990 Computer DX10 Data Base Administrator User's Guide* for an explanation of the DBSTAT command.

2

Data Base Elements

2.1 INTRODUCTION

The major components of a data base are the data hierarchy and the keys. The following paragraphs describe these elements.

2.2 DATA HIERARCHY

The data hierarchy consists of data elements from highest to lowest ranking. These data elements are as follows:

- File
- Record
- Line
- Group
- Field

The basis for the DBMS-990 data element is the document concept. The document is the primary means of initiating, executing, and recording business transactions. Since DBMS-990 allows a direct correlation between the document and the data base (Figure 2-1), the document concept aids development of commercial software systems.

2.2.1 File

A file is the highest level in the logical construct of the DBMS-990 data hierarchy. In terms of the document concept, a file represents a collection of documents of a specific type. A document might contain several pages of information; these pages are easily transferred to the file. Also, the data base can contain one or more files. The design and relationship of the files depend on the application and methods used in the application program.

2.2.2 Record

A record is a collection of one or more data lines. It represents one logical occurrence of a document in a DBMS-990 file. Since a file represents only one type of document, only one record type is represented in a file. However, a file contains many individual data records of the same type.

A primary key uniquely identifies each data record. This key allows rapid access to records, facilitating data access and modification. The purpose of the record, therefore, is to group lines under one primary key.

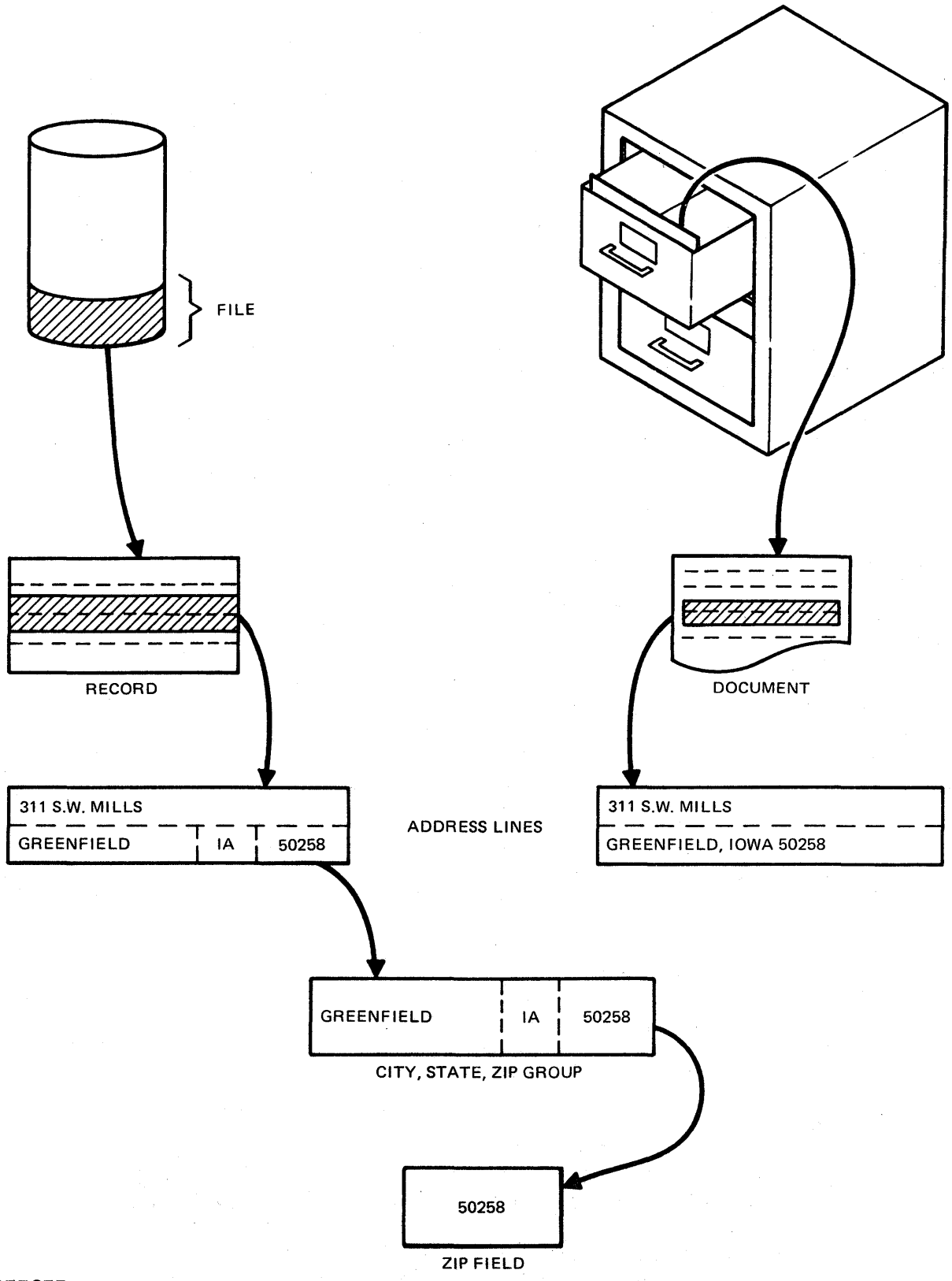


Figure 2-1. Relationship of Data Elements to a Document

A record has no meaning in an input/output (I/O) sense (as commonly associated with file management systems). Concepts such as variable-length or fixed-length records are meaningless in DBMS-990. A data record is the logical construct that contains all of the document lines for a specific transaction. Data transfers occur on a line basis unless the entire record is one line. In the latter case, a transaction resembling a record transfer occurs whenever that one line is requested.

2.2.3 Line

A line is a collection of data fields and/or groups of fields. Usually, each data line corresponds to a line on a business document. However, data from several lines in a document can form a single data line in a record. A data line is the highest-level data element that can be transferred. Examples of lines include address lines, name lines, part lines, and total lines.

Included within the line concept is the notion of line types, or levels. An alphanumeric value in the range of 01 through ZZ indicates the line type. A file description can contain only one definition of a particular line type. Each data record can contain only one 01 data line. Data lines of any other type can appear any number of times. A line type distinguishes between the different lines of a document or record. The DDL declaration defines each line type. Often, each line type has many individual data lines. The maximum size of a line is 512 bytes minus the primary key length, minus an additional 10 bytes, minus 8 times the number of secondary keys in the line.

A line can contain a secondary key that permits access to that line type. For example, by using part number as a secondary key, you can determine all customers that purchased a certain part. Only customer records that contain the specified part number (secondary key value) are read.

2.2.4 Group

A group associates several fields within a line. The group identification consists of a unique four-character ID. During the data definition stage, one or more fields are associated with the group. The fields within the group always occur within the same line and cannot be defined in more than one group. An access request for the group ID retrieves all member fields of the group. The purpose of a group is to allow you to access several fields by using a single name and to facilitate user access to the data during processing of the file. The maximum number of groups/fields per file is 200 minus the sum of the number of keys and the number of line types.

2.2.5 Field

A field is the most elementary member in the logical data base structure. It is the smallest unit that you can access by name during data manipulation. The data definition for a field includes a four-character ID, a data format, and a field length. Data is stored and retrieved by field or group ID within a line. Therefore, a programmer must know the field or group ID and the appropriate line type when accessing data from a DBMS-990 file. The maximum field size is determined by the data type of the field. The maximum number of groups/fields per file is 200 minus the sum of the number of keys and the number of line types.

2.3 KEYS

The two types of keys in DBMS-990 are primary and secondary. Keys facilitate data access and clarify file relationships. The primary key identifies the entire record, whereas a secondary key identifies a particular line within a record. The maximum length for any key is 40 bytes. Figure 2-2 shows how keys fit into a data base structure and how the source document relates to that structure.

2.3.1 Primary Keys

Accessing a file by primary key gives you immediate access to a specific record without requiring you to read the entire file. To access a record, transmit the appropriate key value to DBMS-990. Each record in a file must contain a unique primary key. Duplicate primary key values are not allowed within the same file.

2.3.2 Secondary Keys

A secondary key is a group or a field in one of the defined line types that allows access to a data line without requiring you to use the primary key. While the primary key allows keyed access to the record level, the secondary key permits keyed access directly to the line level. Any line type can contain a secondary key, but secondary keys are not required. Duplicate secondary key values are allowed. A secondary key value can be duplicated between records as well as within a record. However, secondary key values can also be unique. If you plan to use unique secondary key values, impose this restriction in the application program. You can use a maximum of 13 secondary keys for a given file.

An example of a secondary key is one that identifies all customers who purchased a certain product. Using this key, you could determine geographic trends and then use that information to make decisions about shipping, product inventory, and related matters. Figure 2-2 shows the data structure that serves as the basis for this application. Line type 05 contains the secondary key (part number) that facilitates this search.

Although multiple occurrences of line type 05 are possible within a record, a particular product appears only once within a record. Also, for each qualifying line 05, you can retrieve the primary key, name, and address information. This data then transfers to a separate file, which can be sorted and processed to produce a report by name and geographic location.

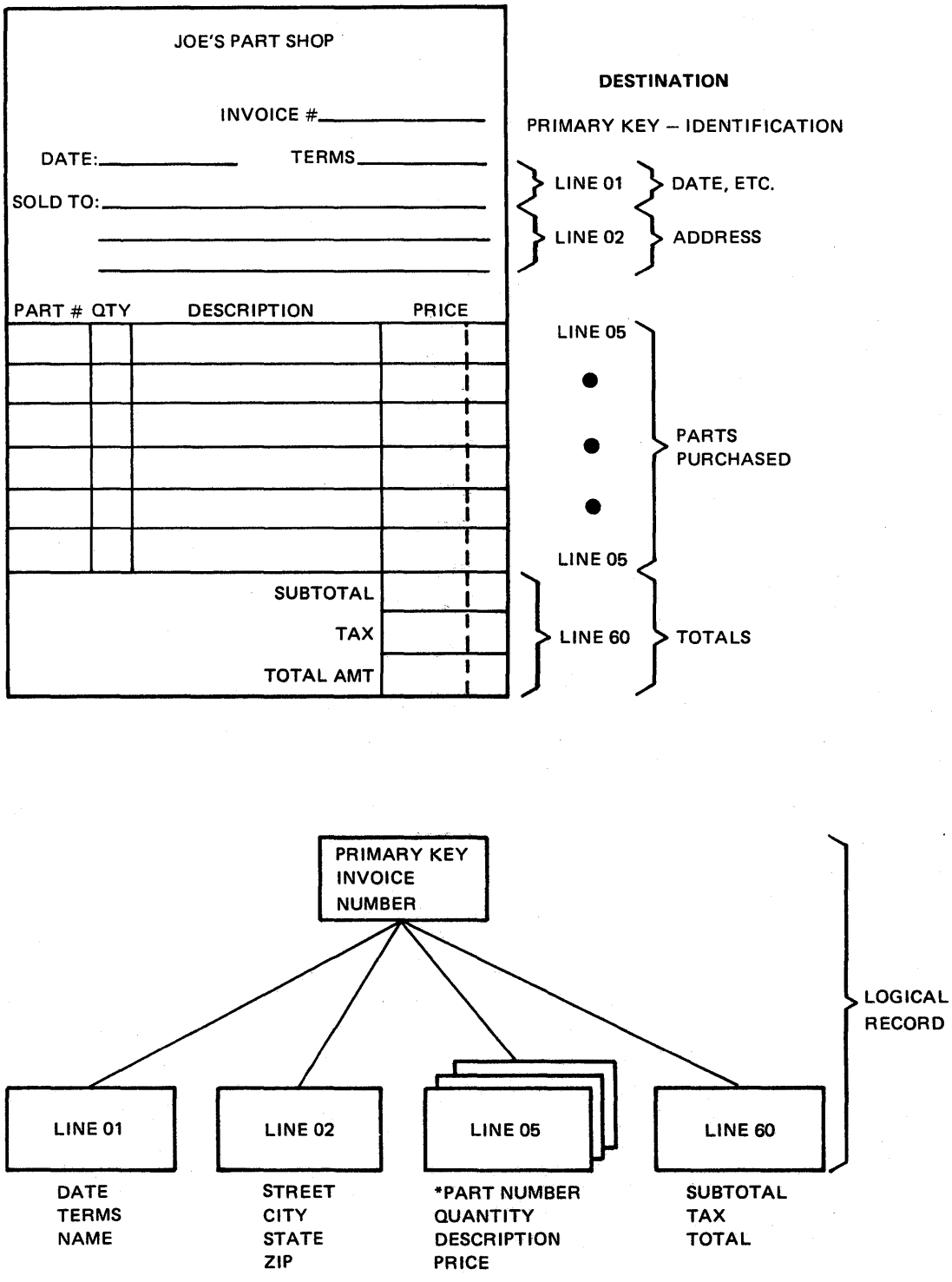


Figure 2-2. Relationship of a Source Document to a DBMS-990 Data Structure

3

Data Definition Language (DDL)

3.1 INTRODUCTION

The DDL describes the logical structure of the files that you manipulate. The DDL permits the complete description of a file and its associated record ID (primary key), lines, groups, and fields. Also, the DDL defines any secondary key IDs.

The output of the DDL compiler is stored with the file on the disk. DBMS-990 uses this output during subsequent file operations. You must describe a file by using the DDL before DBMS-990 can manipulate the file.

Programmers need be only minimally concerned with file boundaries or the position of data, since the DDL usually handles these considerations. As a result, program changes are less likely when the relative position of the data changes. As long as the data names, format, and length do not change, an application program does not need modification.

3.2 STANDARD DDL IDs

IDs in a data base structure apply to data names, keys, files, groups, and fields. The standard DDL ID is fixed length and consists of four characters. The first character must be alphabetic; the remaining characters can be numeric (0 through 9) or alphabetic (A through Z). An ID can contain trailing blanks but not embedded blanks. Whenever defining or accessing an ID, you must include all four characters. For example, define the ID Z1 with two trailing blanks (Z1␣␣).

3.3 DDL DECLARATION

The DDL declaration consists of a series of statements that DBMS-990 compiles. The overall organization of a DDL declaration consists of the following:

- FILE statement
- File description
- Secondary key description
- End File statement

You can include comments at the end of any statement. Placing an asterisk (*) in column one reserves the entire line for a comment. DDL statements can start in any column.

3.3.1 FILE Statement

The FILE statement identifies the file by a user-supplied DDL name and defines the maximum size of the file. The format is as follows:

FILE = <ID>, LINES = <file size> [<comments>]

The <ID> is the standard DDL ID for the file and must be used when accessing the file.

The <file size> is an estimate of the maximum number of data lines allowed in the file. The DDL allocates and reserves sufficient disk space for the file being defined. Refer to the *Model 990 Computer DNOS Data Base Administrator User's Guide* for details about estimating file size.

3.3.2 File Description

The actual file description consists of a series of statements that describe the primary key, lines, groups, and fields. The following statements describe the file:

- Record Identification (ID) statement
- LINE statement
- GROUP statement
- FIELD statement
- End Group (ENDG) statement
- End Line (ENDL) statement

3.3.2.1 Record Identification (ID) Statement. The Record Identification statement defines the primary key ID and the maximum number of primary keys allowed for the file. The following is the format:

ID = <ID> = [<data format>|GROUP],VOL = <records>[,ACCESS = {RANDOM|SEQUENTIAL}[/n]]

The <ID> is a standard DDL or group ID for the primary key and must be used when accessing the file through the primary key.

The <data format> is a standard DDL format for describing the data type of the primary key. It can be signed and can contain assumed decimal places. Refer to paragraph 3.4 for valid data formats. The maximum length of any key is 40 characters.

The <records> specifies the maximum number of primary keys that can exist in the file at any time. This number is also the maximum number of records that can exist in a file, since the primary key is unique for each data record.

The optional ACCESS clause specifies the data retrieval routine for the primary key. The choices are sequential ordering (specified as SEQUENTIAL/1), or random access (specified as RANDOM/1). The default is RANDOM/1. The /1 designator is optional because only one random routine and one sequential routine are currently available.

3.3.2.2 LINE Statement. The LINE statement defines the line type to be associated with the subsequent GROUP and/or FIELD statement. The following is the format:

LINE = <line type> ␣[<comments>]

The <line type> specifies a two-character value referring to the kind of line. Use combinations of letters or numbers in the ranges of 01 through 99 and AA through ZZ. Line type 01 is not required, but you must define at least one line. Although no specific line type is required, each line type must be unique within a particular DDL compile. The limit on the number of groups/fields (200 minus the sum of the number of keys and the number of line types) determines the limit on line types. With the exception of line 01, you determine the order of the line type definitions. When line 01 is present, you must specify it first. The maximum size of a line is 512 bytes minus the primary key length, minus an additional 10 bytes, minus eight times the number of secondary keys in the line. The size of a line is the sum of all field/group sizes in the line.

3.3.2.3 GROUP Statement. The GROUP statement defines the ID of a group; this ID applies to all subsequent field and group statements until an End Group (ENDG) statement is encountered. The format is as follows:

GROUP = <group ID> ␣[<comments>]

The <group ID> specifies a standard DDL ID for the group. The name must be unique within the file definition and can be declared as a secondary key.

3.3.2.4 FIELD Statement. The FIELD statement defines the field ID and data format. The following is the format:

FIELD = <field ID> = <data format> ␣[<comments>]

The <field ID> is a standard DDL ID. The ID must be unique within the file definition and can be declared as a secondary key.

The <data format> is the standard DDL format for describing the data type for the field.

The maximum field size is unique to each data type.

3.3.2.5 End Group (ENDG) Statement. The End Group statement defines the end of a group. Succeeding group definitions are allowed. Any succeeding FIELD or GROUP statements are not part of the last group definition. The format is as follows:

ENDG␣[<comments>]

3.3.2.6 End Line (ENDL) Statement. The End Line statement signifies the end of a line specification. Subsequent statements can be other LINE statements, secondary references, or an End File statement. The format is as follows:

ENDL␣[<comments>]

3.3.3 Secondary Key Description

The secondary key description consists of a SECONDARY-REFERENCES statement followed by one or more secondary key statements.

3.3.3.1 SECONDARY-REFERENCES Statement. The SECONDARY-REFERENCES statement signifies the beginning of the secondary key description. The following is the format:

```
SECONDARY-REFERENCES
```

3.3.3.2 Secondary Key Statement. The Secondary Key statement defines a previously defined group or field as a secondary key. A maximum of 13 Secondary Key statements is allowed per file. Any field or group defined as a secondary key can have a maximum of 40 characters. The format of the statement is as follows:

```
{<group ID>|<field ID>} = VOL = <keys> [,ACCESS = {RANDOM|SEQUENTIAL} [/N]]
```

The <group ID> or <field ID> is the standard DDL ID for the previously defined group or field that is specified as a secondary key.

The <keys> defines the maximum number of unique secondary key values that can exist in the file at any time for a specific secondary key. Duplicate values are allowed for secondary keys; these values are linked together. The size of the file determines the maximum number of duplicate values.

The optional ACCESS clause specifies the data retrieval routine for the secondary key. The choices are sequential ordering (specified as SEQUENTIAL/1), and random access (specified as RANDOM/1). The default is RANDOM/1. The /1 designator is optional because only one random routine and one sequential routine are currently available.

3.3.4 End File Statement

The End File statement signifies the end of the DDL declaration. The format is as follows:

```
END.␣<comments>
```

3.4 DATA FORMATS

The primary purpose of the data format is to allow the use of a query language, since this language needs to know the format of the data it displays. The data format is also a handy user reference. You are responsible for validating the format of data sent to the data base via DBMS-990. DBMS-990 does not validate the data it receives against the data format defined by the DDL.

The syntax for the data formats specified in the DDL is one of the following:

Type one: <data type>

or

Type two: <data type>/n

or

Type three: <data type>/n.d

where:

<data type> is a two-character code representing the overall characteristics of the data.

n is the total field length in bytes; you can omit n in formats where a default exists.

d is the number of places to the right of the decimal; use 0 if no decimal values are needed; d is not required for some formats (see Table 3-1).

Table 3-1 lists the various data types and provides a description and example of each. The data format is the last parameter in the FIELD statement and the second parameter in the Record Identification statement. Format types are specified according to the definition in the syntax. Type one format specifies only the format code; type two specifies the format code and field length; and type three specifies the format code, field length, and number of decimal places.

A single DDL can contain all of the data formats for the different languages. However, ensure that each language accesses only those elements whose data formats that particular language can use. Otherwise, unpredictable results can occur.

Table 3-1. DDL Data Types

Code	Description	Example Formats
AN	Arithmetic without sign. Decimal places are allowed. Use zero for no decimal places. Use type three format.	AN/8.2 COBOL: PIC 9(6)V9(2) COMP. FORTRAN: <none> Pascal: <none>
AS	Arithmetic signed. Length (n) must include sign, and decimal places are allowed. Use zero for no decimal places. Use type three format.	AS/8.2 COBOL: PIC S9(5)V9(2) COMP. FORTRAN: <none> Pascal: <none>
CH	Character string. Length includes total characters. Decimal places not allowed. Use type two format.	CH/20 COBOL: PIC X(20). FORTRAN: <A format> Pascal: PACKED ARRAY [1..20] of CHAR
CN	Character numeric. Decimal places are allowed. Use zero for no decimal places. Use type three format.	CN/6.2 COBOL: PIC 9(4)V9(2). FORTRAN: <none> Pascal: <none>
CS	Character numeric signed. Length (n) must include the sign. Decimal places are allowed. Use zero for no decimal places. Use type three format.	CS/8.5 COBOL: PIC S9(2)V9(5) FORTRAN: <none> Pascal: <none>
CX	Complex variable. Length (n) default is 8; if specified, it must be 8. Use type one or two format.	CX/8 COBOL: <none> FORTRAN: COMPLEX Pascal: <none>
FX	Scaled integer FORTRAN. Length (n) default is 2; if specified, it must be 2. (d) is the number of bits to the right of the binary point: the default is 0. Use type three format.	FX/2.4 COBOL: <none> FORTRAN: FIXED(4) Pascal: FIXED(16.4)
IS	Single-precision integer. Contained in one 16-bit word. Length (n) default is 2; if specified, it must be 2. Field may contain a sign. Use type one or two format.	IS/2 COBOL: PIC 9(5) COMP-1. FORTRAN: INTEGER*2 Pascal: INTEGER

Table 3-1. DDL Data Types (Continued)

Code	Description	Example Formats
ID	Double-precision integer. Contained in two 16-bit words and may be signed. Length (n) default is 4; if specified, it must be 4. Use type one or two format.	ID/4 COBOL: <none> FORTRAN: INTEGER*4 Pascal: LONGINT
LG	Logical variable. Length (n) default is 2; if specified, it must be 2. Use type one or two format.	LG/2 COBOL: <none> FORTRAN: LOGICAL Pascal: BOOLEAN
PK	Packed decimal. Digit length (n) must be even and includes the sign. Decimal places are allowed, and zero indicates no decimal places. Contained in n/2 bytes. Use type three format.	PK/6.2 COBOL: PIC S9(3)V9(2) COMP-3. FORTRAN: <none> Pascal: <none>
RS	Single-precision real. Contained in two 16-bit words and may be signed. Length (n) default is 4; if specified, it must be 4. Use type one or two format.	RS/4 COBOL: <none> FORTRAN: REAL *4 Pascal: REAL
RD	Double-precision real. Contained in four 16-bit words and may be signed. Length (n) default is 8; if specified, it must be 8. Use type one or two format.	RD/8 COBOL: <none> FORTRAN: REAL *8 Pascal: REAL

3.5 DDL EXAMPLES

Figures 3-1 and 3-2 are DDL examples. Figure 3-1 shows a DDL using a group primary key, nesting of groups, and indentation to improve readability. Figure 3-2 shows a DDL not using a group primary key. The next few paragraphs discuss designing a DBMS-990 file, followed by an explanation of the DDL.

```
 DBMS-990     <L. V. R>              DDL TRANSLATOR      MM/DD/YY   HH: MM: SS

FILE=EMPL, LINES=266
*
ID=NUMB=CN/6. 2, VOL=30, ACCESS=RANDOM/1
*
LINE=01
  FIELD=NAME=CH/20
  GROUP=ADDR
    FIELD=STRE=CH/20
    FIELD=CITY=CH/15
    FIELD=STAT=CH/2
    FIELD=ZIPC=CN/5. 0
    ENDG
  FIELD=SSN =CN/9. 0
ENDL
*
LINE=HI
  FIELD=JOB =CH/10
  GROUP=LOCA
    FIELD=LCTY=CH/10
    FIELD=LSTA=CH/15
    ENDG
  GROUP=RANK
    FIELD=GRAD=CH/2
    FIELD=RATE=CN/6. 2
    FIELD=EXPT=LG/2
    ENDG
ENDL
*
SECONDARY-REFERENCES
SSN =VOL=30, ACCESS=RANDOM/1
*
END.

          TOTAL PAGES REQUIRED - 108
           LINE LENGTH (BYTES) - 96
       TOTAL DESCRIPTION PAGES - 1
                TOTAL KEY PAGES - 6

  LINE 01 --   BASE = 16 , DATA = 71 , LINKAGE = 8  , TOTAL = 95
  LINE HI --   BASE = 16 , DATA = 45 , LINKAGE = 0  , TOTAL = 61

U DBMS-0112  **  NEW DATA BASE FILE CREATED  **
```

Figure 3-1. DDL Example Without a Group Primary Key

```
 DBMS-990     <L.V.R>              DDL TRANSLATOR       MM/DD/YY   HH:MM:SS

FILE=EMPL,LINES=266
*
ID=ENUM=GROUP,VOL=30,ACCESS=SEQUENTIAL/1
  FIELD=DEPT=CH/2
  FIELD=SSN =CH/9
  ENDG
*
LINE=02
  FIELD=NAME=CH/30
ENDL
*
LINE=03
  GROUP=ADDR
    FIELD=STRT=CH/20
    FIELD=CITY=CH/20
    FIELD=STAT=CH/2
    FIELD=ZIPC=CN/5.0
    ENDG
ENDL
*
END.

          TOTAL PAGES REQUIRED - 81
           LINE LENGTH (BYTES) - 68
       TOTAL DESCRIPTION PAGES - 1
                TOTAL KEY PAGES - 8

  LINE 02 --  BASE = 21 , DATA = 30 , LINKAGE = 0  , TOTAL = 51
  LINE 03 --  BASE = 21 , DATA = 47 , LINKAGE = 0  , TOTAL = 68

U DBMS-0112  **  NEW DATA BASE FILE CREATED  **
```

Figure 3-2. DDL Example with a Group Primary Key

3.6 DDL PROCEDURES

The basic user design considerations are those of organizing and defining a DBMS-990 file. Then, to create the file, you must assemble a DDL to define the file and compile the DDL. The compiled file definition (DDL) resides on the disk within the file it defines. This file definition is used not only in creating the file but also in maintaining it.

3.6.1 User Design Considerations

You should work closely with the data base administrator (DBA) in designing the data base and/or files. More detailed information is available in the *Model 990 Computer DNOS Data Base Administrator User's Guide* .

3.6.1.1 Lines and Fields. The organization of lines and fields can influence both the overall efficiency of programs that access the file and the effective use of disk storage. Line lengths for various line types should be as close as possible to the same length. However, if a particular line is contained no more than once per data record, the overhead factor is usually insignificant.

The number of line types defined for a record affects access efficiency. As a result of combining several document lines with related information into one larger line type, fewer reads are necessary and access efficiency increases. Therefore, balance line lengths and combine the elements of the document lines into fewer line types.

In choosing a data format and/or data type for a field, choose one that best describes the use of the field. Use complex data formats only when necessary. The query language uses this format for display purposes. If the wrong data format is defined in the DDL for a particular field, an incorrect display results, requiring a change in the data format.

3.6.1.2 Secondary Keys. A secondary key is a field that identifies a line for processing purposes. You should limit the number of secondary keys as much as possible. Also, relate the justification for a key to specific processing needs. You can use groups for secondary keys, but the maximum size of a secondary key is 40 characters. The maximum number of secondary keys per file is 13. Line 01 should not contain a secondary key if that key must be updated because a line type 01 cannot be deleted until all other lines are deleted.

3.6.2 Creating a DDL File

You can use the Text Editor to make or correct a DDL file for input to the DDL compiler. For details about the Text Editor, refer to the *Model 990 Computer DNOS Text Editor Reference Manual*. To display or print DDL files, enter a Show File (SF) command or a Print File (PF) command.

3.6.3 Format DDL (DDL) Command

The SCI command DDL creates a data base file according to the DDL statements in the input control file. The DDL listing is placed in the output list file specified, and the data base file is created with the pathname specified. The following shows the format and prompts of the DDL command:

```
FORMAT DDL
          INPUT ACCESS NAME:
      LISTING ACCESS NAME:
                PAGE SIZE:  256
        DB FILE PATHNAME:
```

Respond to the DDL prompts as follows:

INPUT ACCESS NAME — Enter a pathname that identifies the file containing the input to the DDL compilation.

LISTING ACCESS NAME — Enter the pathname of the file to which the summary list of the DDL compilation is written.

PAGE SIZE — Enter 256 or 288. The size of the page is a blocking factor, specifying the unit of bytes read or written during an I/O operation.

DB FILE PATHNAME — Enter the pathname of the section in which the defined file is allocated; you can use DUMY as the filename to check the DDL syntax.

The following example shows how to enter a DDL command to define a file:

```
FORMAT DDL
          INPUT ACCESS NAME:  .SAMPLE.DDL
      LISTING ACCESS NAME:  .SAMPLE.LSTDDL
                PAGE SIZE:  256
        DB FILE PATHNAME:  .DBMS.DBFILE
```

This DDL execution creates the file definition specified in the file .SAMPLE.DDL and places the DDL listing on file .SAMPLE.LSTDDL. The data base file is created under the pathname .DBMS.DBFILE.

3.6.4 DDL Listing

After creating the DDL file, you must compile it. Information specific to the data definition is printed after the compile ends, whether it ends normally or abnormally. In case of an abnormal termination, the approriate error messages appear.

The DDL in Figure 3-2 defines the DBMS-990 file EMPL. The primary-key ID is NUMB (employee number) and is 6 characters in length, with two decimal places. The maximum number of primary keys that the file can contain is 30 (VOL = 30); this number is also the maximum number of data records that the file can contain (since each data record must have a primary key). The primary key values are ordered randomly.

The listing next defines the line types of the file. Each line type includes an identifying number. The definition then specifies the fields in the line. Each line type definition ends with an End Line statement. Each field type defined is given an ID and a data format. To group fields, define them between a GROUP statement and an End Group statement. A particular field cannot belong to more than one group. After defining all of the line types, the listing specifies any secondary references. Then, an End File statement terminates the DDL definition.

The information given at the end of a successful DDL run starts with the total number of pages required by the file. Next, the line length, in bytes, of each line to be stored in the file appears. This length, which is justified to a word boundary, equals the length of the longest line defined provided the latter is an even number; if it is an odd number, 1 is added to it.

Next, the total number of pages used for description information and key storage appears. A page is the unit of bytes that DBMS-990 uses for I/O, that is, the basic unit that the data base actually reads or writes. The page size can be either 256 or 288 bytes. Choose the page size based on the sector size of the disk.

Line type summary information is printed last, before the **NEW FILE CREATED** message. This information is as follows:

- BASE information — Refers to the number of bytes DBMS-990 uses to store primary key data in each data line; the number of bytes equals ten plus the length of the primary key.
- DATA information — Refers to the number of bytes used in a data line to store the data for that particular line type regardless of whether all fields are valued. Reserve space in a data line for any fields that are not valued initially.
- LINKAGE information — Refers to the number of bytes used in a data line for each secondary reference defined for that line type.
- TOTAL information — Shows the total number of bytes this particular line type uses when it occurs as a data line in the DBMS-990 file.

Note that the number of bytes that each line type reserves for data is different. This difference can result in a significant amount of unused space in a file when the lengths of the line types are highly dissimilar and all line types, other than line 01, occur as data lines with a high frequency.

3.7 DDL ERRORS

Appendix A lists and explains the various DDL errors.

4

Data Manipulation Language (DML)

4.1 INTRODUCTION

The data manipulation language (DML) consists of specific function codes that allow you to manipulate data. The DML functions consist of reading or writing data lines, reading a key, deleting data lines, opening or closing files, and starting, committing, or rolling back transactions. Ensure that the appropriate file is open before any processing begins.

It is not possible to read or write an entire data record unless that particular record consists of only one data line. Writing to a file consists of creating new data records (adding new lines to the file), updating data records (changing data within a file), or deleting data records (removing lines from the file).

To access a data base file, include the appropriate DML commands in an application program, which is sometimes referred to as a *host language program*. Use the host language to contruct a call to DBMS-990 according to the requirements of the DML. The host language DML call uses a parameter list to inform DBMS-990 of the type of function (access) to be performed and the elements to be manipulated by the function. DBMS-990 processes the request and returns the results to the host language program in a data area provided in the DML call parameter list. Formulate the DML call parameter list according to the specifications in this section.

The DML calls occur on a line basis. However, the data access can occur on a field and/or group basis. When initiating a DML call, you must provide the following information:

- Password
- File ID
- Key ID
- Key value
- Line type
- Field/group ID(s)
- Name of user data area

The password information is required only if you installed the security feature at generation time. Otherwise, the password is ignored. In either case, you must allocate the password area of the call parameters.

You can transfer all or part of a data line by specifying the fields and/or group for which data is to be transferred. You need not know the specific position of the data in the file, only the logical name (the field or group ID), the data format, and the line type. Thus, the application program does not change as long as the element's logical name, line type, and data format do not change.

4.2 CALL PARAMETERS

A DML call to DBMS-990 consists of six parameters. To call DBMS-990, use the procedures specified for calling external subroutines in the COBOL, FORTRAN, and Pascal manuals (see Preface). The following is the COBOL syntax for a DML call:

```
CALL "DBMSYS" USING     <control block>
                        <end of control block>
                        <line list>
                        <end of line list>
                        <data area>
                        <end of data area>
```

All parameters are required. The actual data item representing each parameter must begin on a word boundary. In COBOL, the best way to ensure word alignment of the control block, line list, and data area parameters is to define each as a 01 data item. In Pascal, use an unpacked record structure to define the control block, line list, and data area parameters. If you pack a Pascal record structure the call parameters might not align on a word boundary. Specify an extra element (word) at the end of the control block, line list, and data area for Pascal. This element signifies the end of the control block, line list, and data area to DBMS. In FORTRAN, define the control block, line list, and data area parameters as DIMENSION arrays that will be aligned on a word boundary. If you use any other methods, the parameters might not align on a word boundary.

4.2.1 Control Block

To define this parameter in COBOL, specify the name of the 01 data item that starts the control block; in FORTRAN, specify the first location in the dimensioned array; in Pascal, specify the variable name of the control block record structure. The control block contains the following information:

Description	Positions	Length in Bytes	Set By
Password	1-4	04	User
Function code	5-6	02	User
Status (exception code)	7-8	02	DBMS-990
File ID	9-12	04	User
Location pointer 1 (loc1)	13-16	04	Both
Location pointer 2 (loc2)	17-20	04	Both
Key ID	21-24	04	Both
Key value	25-N	1-40	Both

In FORTRAN, you must dimension an extra word after the key value.

In Pascal, you must define an extra unpacked element after the key value. The extra element marks the end of the control block for DBMS. The password is optional, depending on whether security was invoked during the installation of DBMS-990. However, you must always allocate the password positions. The function code defines the type of data manipulation requested.

When an exception occurs, an appropriate exception code is returned in the status field. Check the status bytes after every operation. If they do not contain asterisks, an exception has occurred. Refer to Appendix A for a description of exception conditions.

CAUTION

Do not modify the location pointers (loc1 and loc2) to any value other than asterisks (****) unless a DML function explicitly calls for such modification. Inappropriate modifications can seriously damage the file or produce unpredictable results. The only exceptions are when you place the location pointer results of one DML call into the control block for another DML call or when you swap the contents of loc1 and loc2.

For most DML calls, you must set the key ID area before performing the call. The key ID is the four-character name for the primary or secondary key defined at data definition time. Use a key value that already exists, if possible. If the value does not have a corresponding key in the file for the key ID specified, an exception condition results. The length of a key value may vary from key to key; the maximum is 40 bytes. If the defined key value field is larger than the length of the key, the key value is returned left justified.

CAUTION

If the defined key value field is shorter than the actual length of the key, the next field is overlaid with a portion of the key value.

4.2.2 End of Control Block

The end of control block parameter tells DBMS-990 where the control block ends. In COBOL, this parameter is the name of the 01 data item that immediately follows the key value. In Pascal or FORTRAN, it is the name of the last (i.e., extra) element in the control block.

4.2.3 Line List

The line list parameter defines the line(s) and field(s) used in the operation. It also specifies the disposition of the returned information. In COBOL, this parameter is the name of the 01 data item that starts the line list. In Pascal, it is the variable name of the line list record structure. In FORTRAN, it is the first location in the dimensioned array for the line list.

The following is the format for a single line type in the line list parameter:

Description	Length (Bytes)	Set By	
Line identification	07	User	(Must be present
Return indicator	01	Both	for each line
Field IDs	Variable (4 characters each)	User	type specified.)
Disposition	08	User	(Must be last entry of list.)

In FORTRAN, dimension an extra word at the end of the line list, after the disposition.

In Pascal, define an extra unpacked element at the end of the line list, after the disposition.

Line identification consists of a seven-character line type specification, as follows:

LINE = xx

where:

xx is the line type.

The return indicator signifies the line type from which a data line is retrieved. If the return code contains an asterisk, the data fields for that particular line type are returned in the data area. In a line list containing multiple line types, the return indicator of the accessed line type is set to an asterisk on return from the call. Each of the other return indicators is set to a comma. On adds, replaces, and deletes, set the return indicator of the appropriate line type to an asterisk. Set each of the other return indicators to a comma or to something other than an asterisk.

Field IDs are the four-character IDs of the fields in a specified line type. They may be individual four-character definitions, they may be strung together, or they may be defined by an array.

Disposition specifies whether to release (RLSE) or hold (HOLD) a retrieved data line. Release implies that the retrieved information is for inquiry purposes. Hold retains the line for update or deletion. The following is the format for disposition:

Position	Value
1-4	****
5-8	RLSE or HOLD

With a RLSE disposition, the data line is available to other users immediately upon completion of the DML call. The HOLD disposition retains a data line until the line is used in a write or delete DML function or until the execution of another read with hold on a different data line. If the held data line is not released, it is retained until the program terminates; no other program has update capabilities on that line until such termination. Also, read access is not allowed on a held line. A program can put on hold only one data line at a time, regardless of the number of files open. HOLD has no meaning in any of the update functions.

4.2.4 End of Line List

The end of line list parameter tells DBMS-990 where that line list ends. This parameter must be the name of the next item after disposition of the line list. In a COBOL program, use the level 01 data item that follows the line list. In Pascal, use the name of the last (or extra) element in the line list record structure.

4.2.5 Data Area

The data area parameter specifies the data name of one of the following:

- The program area that contains the data to be transmitted
- The program area that will contain retrieved data

The data area should be large enough to contain the data for the fields specified in the line list. You can specify either a single definition for the data area or a definition that is subdivided into definitions for all of the fields specified in the line list. If the defined data area is larger than the combined length of the data fields requested, the retrieved data is returned left justified. Data is placed in the data area in the same order as that of the field IDs of the line list.

CAUTION

If the defined data area is shorter than the combined length of the data fields requested, the next field is overlaid with a portion of the retrieved data.

4.2.6 End of Data Area

The end of data area parameter tells DBMS-990 where the data area ends. This parameter must be the name of the data structure that follows the last data item in the data area. In a COBOL program, use the next level 01 data item following the data area. In Pascal, use the name of the extra or last element in the data read area record structure. In FORTRAN, use the name of the extra array element (word) that is dimensioned at the end of the data area.

4.2.7 Parameter List Examples

Figure 4-1 is an example of a COBOL definition of the parameters and the corresponding CALL to DBMS-990. Note that the end parameters are one-byte separators. As a result, the first two end parameters and their corresponding data items can be eliminated; they are then replaced by the data line list and the data area, as in Figure 4-2. Figure 4-2 shows the resulting COBOL call and its definitions.

Figure 4-3 is an example of a Pascal definition of the DML parameters and the corresponding call to DBMS-990. The end parameters are the name of the extra (last) unpacked element that must be defined in the control block, line list, and data area parameters. A multiple line list parameter is created by using a Pascal record structure large enough to contain the necessary data. This structure indicates the desired lines plus the extra (last) unpacked element at the end of each record structure.

4.2.7.1 COBOL Call with Dummy Parameters. When a function code does not need a line list or data area, you can use dummy parameters. For example, you can use a single asterisk as the open function (OF) line list and data area parameters. The following is the COBOL call with ASK as the dummy parameter; if ASK is a 01 data item following CONTROL-BLOCK, ASK can be used as the END-CONTROL parameter.

```
CALL "DBMSYS" USING     CONTROL-BLOCK
                        END-CONTROL,
                        ASK,
                        ASK,
                        ASK,
                        ASK,
```

```
                DBMSYS(CB, CB. TERM, CB. TERM::LINELIST, CB. TERM,
                CB. TERM::DATAAREA, CB. TERM);

WORKING-STORAGE SECTION

01      CONTROL-BLOCK.
          02 FILLER          PIC X(4)      VALUE SPACES.
          02 FUNC            PIC XX        VALUE "RF".
          02 STAT            PIC XX        VALUE "**".
          02 FILEX           PIC X(4)      VALUE "SOFL".
          02 LOC1            PIC X(4)      VALUE "****".
          02 LOC2            PIC X(4)      VALUE "****".
          02 KEYN            PIC X(4)      VALUE "SONM".
          02 KEYX            PIC X(6).

01      END-CONTROL-BLOCK     PIC X         VALUE "*".

01      LINE-LIST.
          02 FILLER          PIC X(7)      VALUE "LINE=01".
          02 TST-1           PIC X         VALUE "*".
          02 FILLER          PIC X(4)      VALUE "VEN1".
          02 FILLER          PIC X(7)      VALUE "LINE=02".
          02 TST-2           PIC X         VALUE ",".
          02 FILLER          PIC X(4)      VALUE "VEN2".
          02 FILLER          PIC X(7)      VALUE "LINE=03".
          02 TST-3           PIC X         VALUE ",".
          02 FILLER          PIC X(8)      VALUE "ITMMQTYX".
          02 FILLER          PIC X(8)      VALUE "****HOLD".

01      END-LINE-LIST         PIC X         VALUE "*".
01      DATA-AREA             PIC X(40).
01      END-DATA-AREA         PIC X         VALUE "*".

PROCEDURE DIVISION.

CALL "DBMSYS" USING     CONTROL-BLOCK,

                        END-CONTROL-BLOCK,

                        LINE-LIST,

                        END-LINE-LIST,

                        DATA-AREA,

                        END-DATA-AREA.
```

Figure 4-1. Example One of COBOL DML Parameters

```
WORKING-STORAGE SECTION.

01      CONTROL-BLOCK.
          02 FILLER          PIC X(4)      VALUE SPACES.
          02 FUNC            PIC XX        VALUE "RF".
          02 STAT            PIC XX        VALUE "**".
          02 FILEX           PIC X(4)      VALUE "SOFL".
          02 LOC1            PIC X(4)      VALUE "****".
          02 LOC2            PIC X(4)      VALUE "****".
          02 KEYN            PIC X(4)      VALUE "SONM".
          02 KEYX            PIC X(6).
01      LINE-LIST.
          02 FILLER          PIC X(7)      VALUE "LINE=01".
          02 TST-1           PIC X         VALUE "*".
          02 FILLER          PIC X(4)      VALUE "VEN1".
          02 FILLER          PIC X(7)      VALUE "LINE=2".
          02 TST-2           PIC X         VALUE ",".
          02 FILLER          PIC X(4)      VALUE "VEN2".
          02 FILLER          PIC X(7)      VALUE "LINE=3".
          02 TST-3           PIC X         VALUE ",".
          02 FILLER          PIC X(8)      VALUE "ITMMQTYX".
          02 FILLER          PIC X(8)      VALUE "****HOLD".
01      DATA-AREA           PIC X(40).
01      END-DATA-AREA       PIC X         VALUE "*".

PROCEDURE DIVISION.

CALL "DBMSYS" USING     CONTROL-BLOCK,

                        LINE-LIST,

                        LINE-LIST,

                        DATA-AREA,

                        DATA-AREA,

                        END-DATA-AREA.
```

Figure 4-2. Example Two of COBOL DML Parameters

4.2.7.2 FORTRAN Call with Dummy Parameters. Figure 4-3 is an example of a FORTRAN definition of the DML parameters and the corresponding call to DBMS-990. The end parameters are the name and subscript of the extra last word that must be dimensioned in the arrays for the control block, line list, and data area parameters. The multiple line list parameter dimensioned is large enough to contain the data that indicates the desired lines plus the extra word at the end of the array. The necessary constants can be easily loaded by using the FORTRAN data statements. All FORTRAN examples assume IMPLICIT INTEGER (A-Z).

```
      DBMSYS(CB, CB. TERM, CB. TERM::LINELIST, CB. TERM,
                CB. TERM::DATAAREA, CB. TERM);


            .
            .
             DIMENSION ITEMCB(15), ITEMLL(13), ITEMDA(14)
            .
            .
            .
      C*  INITIALIZE ARRAYS
            DATA ITEMCB/'    RF  ITEM********ITMN      '/
            DATA ITEMLL/'LINE=01, DESCUPRC****RLSE  '/
            .
            .
            .
      C*  CALL DATA BASE
            CALL DBMSYS (ITEMCB(1), ITEMCB(15), ITEMLL(1),
       1       ITEMLL(13), ITEMDA(1), ITEMDA(14))
            .
            .
            .
```

Figure 4-3. Example of FORTRAN DML Parameters

4.2.7.3 Pascal Call with Dummy Parameters. Figure 4-4 illustrates an example of a Pascal call using dummy parameters, the end of the control block is used for five of the parameters.

```
(*    DEFINE DATA TYPES *)
TYPE
     C2   = PACKED ARRAY [1. .2 ] OF CHAR;
     C4   = PACKED ARRAY [1. .4 ] OF CHAR;
     C6   = PACKED ARRAY [1. .6 ] OF CHAR;
     C8   = PACKED ARRAY [1. .8 ] OF CHAR;
     C20 = PACKED ARRAY [1. .20] OF CHAR;
     DA_TYPE = (SOF2, SOF3, CUST, ITEM);
(*    DEFINE RECORD AREAS *)
DATAREA = RECORD
     CASE  DA_TYPE OF
     SOF2    :   (SHIP    :   C6);
     SOF3    :   (QUAN  :   C4;
                  SONM  :   C6);
     CUST    :   (NAME  :   C20);
     ITEM    :   (DESC   :   C20; UPRC : C6);
END;
LINELIST = RECORD
     LL : PACKED ARRAY [1. .24] OF CHAR;
     TERM : INTEGER;      (* EXTRA ELEMENT MARKING END OF LINELIST *)
END;
CONTROLBLOCK = RECORD
     PSWD   :   C4;
     FUNC   :   C2;
     STAT   :   C2;
     DBFILE :   C4;
     LOC1   :   C4;
     LOC2   :   C4;
     KEYN   :   C4;
     KEYV   :   C6;
     TERM   :   INTEGER;   (* EXTRA ELEMENT MARKING END OF CONTROL BLOCK *)
END;
(*    VARIABLE DEFINITIONS *)
VAR
     CUST_LL : LINELIST;
     DA : RECORD
          DATA : DATAREA;
          TERM : INTEGER;      (* EXTRA ELEMENT MARKING END OF DATA AREA *)
     END;
     CB : CONTROLBLOCK;
(*    DEFINE EXTERNAL PROCEDURE TO CALL DBMS *)
PROCEDURE DBMSYS (VAR CB  : CONTROLBLOCK; VAR CBE  : INTEGER;
                        VAR LL   : LINELIST; VAR LLE  : INTEGER);
                        VAR DA  : DATAREA; VAR DAE  : INTEGER);
                        EXTERNAL FORTRAN;
     .
     .
     .
      DBMSYS (CB, CB. TERM, CUST_LL, CUST_LL.  TERM, DA.DATA, DA.TERM);
```

Figure 4-4. Example of Pascal DML Parameters

4.3 DML FUNCTIONS

The DML functions are of four types: file, read, update, and transaction. Using the appropriate function codes, you can manipulate, alter, and delete the data base records. However, data transfers occur on a line basis. The field IDs specified in the line list parameter of the DML call determine which data fields the application program uses.

4.3.1 File Functions

The following paragraphs describe file-access checking and the open and close file functions. Although the open and close file functions are not required when file-access checking is not installed in DBMS-990, it is recommended that they be included in all applications so that the programs will still execute if file-access checking is later installed. File functions are ignored by DBMS-990 when file-access checking is not installed.

4.3.1.1 File-Access Checking. An application program must issue open and close file functions for each file when file-access checking is installed. Issue an open file before attempting to access the file and a close file when finished with the file.

Three types of file access are allowed with DBMS-990: shared, exclusive, and read-only exclusive. The most common type is shared. With shared access, all users can open the file and perform all functions on the file. Exclusive access means that the current user has all access privileges and is the only user with access to the file. Read-only exclusive access means that the current user has only read privileges and that other users can read the file if they open with the same file access.

File-access checking monitors the current file status against requested accesses, checks compatibility, and returns appropriate error conditions when incompatibility exists. For example, a file-access request of exclusive is allowed only when that file is free from any other file access. If the file is not free and a user requests exclusive access, file-access checking returns the appropriate error condition. Table 4-1 lists all of the conditions under which DBMS-990 must resolve file-access requests and the corresponding results.

Table 4-1. File Access Resolutions

Requested Access	Current Status*			
	None	Shared	Exclusive	Read-Only Exclusive
Shared	Y	Y	N	N
Exclusive	Y	N	N	N
Read-Only Exclusive	Y	N	N	Y

Note:

* Y indicates access granted; N indicates access refused.

4.3.1.2 Open File (OF). The open file function (DML function code OF) opens the file specified in the file ID area of the control block. Specify the type of access needed in the key name area of the control block. Use one of the following access types:

- EXCL — for exclusive access; no other user can access the file; all read and write functions are permitted.
- ROEX — for read-only exclusive access; other users can read the file if they open ROEX; only read functions are permitted.
- SHRD — for shared access; other users can access the file; all read and write functions are permitted.

Figure 4-5 is an example of a COBOL control block parameter and DML call for performing file functions.

```
01      CONTROL BLOCK.
                  02      PSWD        PIC X(4)        VALUE SPACES.
                  02      FUNC        PIC XX          VALUE "OF".
                  02      STAT        PIC XX          VALUE "**".
                  02      FLID        PIC X(4)        VALUE "PYRL".
                  02      FILLER      PIC X(8)        VALUE "********".
                  02      KYID        PIC X(4)        VALUE "EXCL".
                  02      KEYV        PIC X(40)       VALUE "EXCL".
01      ASK                         PIC X           VALUE "*".
.
CALL "DBMSYS" USING CONTROL-BLOCK, ASK, ASK, ASK, ASK, ASK.
```

Figure 4-5. COBOL File Function

Figure 4-6 is an example of a FORTRAN control block parameter and DML call for performing file functions.

```
  DIMENSION OFCB (13)
  DATA OFCB/   OF**SOFL********ROEX  '/
  CALL DBMSYS (OFCB (1), OFCB (13), OFCB (13), OFCB (13),
$OFCB (13), OFCB(13))
```

Figure 4-6. FORTRAN File Function

All call parameters must be specified even though the line list and data area parameters are not used. The example in Figure 4-2 can be used for file functions. Parameters not needed can be defined as one-byte fields but must be on a word boundary.

Even if security is included in the system, security checking is not performed on file functions. Also, you must specify access type and check the status bytes for errors on returning from the DML call. Refer to Appendix A for a list of error codes.

4.3.1.3 Close File (CF). The close file (CF) function closes the file specified in the file ID area of the control block. The CF function uses the same format as the OF function. The function and file ID must be valid. The access type in the key ID parameter must be the same as that specified in the OF function. Check the status field for errors. If another program or task is using the file, the file is not closed until all programs accessing the file have closed it. When the status is indicated by a series of asterisks, the application program file has successfully closed.

You can specify the other parameters of the CF function in the same manner as in the OF function. Define parameters that are not required as one-byte fields.

4.3.2 Read Functions

The DML read functions return a data line in a file based on the selection criteria provided in the DML call parameters. The specified data is removed from the data line and returned to the calling program via the call parameters. Only one data line is processed per DML call. Check the status bytes of each read function after the DML has completed to ensure that the function has terminated properly; the status code should contain a series of asterisks. If an error condition occurs, refer to Appendix A. DBMS-990 sets the status bytes to asterisks before it executes the function.

Figure 4-7 is an example of a Pascal control block parameter and DML call for performing file functions.

```
(*    DEFINE DATA TYPES *)
TYPE
     C2   = PACKED ARRAY [1. .2 ] OF CHAR;
     C4   = PACKED ARRAY [1. .4 ] OF CHAR;
     C6   = PACKED ARRAY [1. .6 ] OF CHAR;
     C8   = PACKED ARRAY [1. .8 ] OF CHAR;
     C20 = PACKED ARRAY [1. .20] OF CHAR;
     DA_TYPE = (SOF2, SOF3, CUST, ITEM);
(*    DEFINE RECORD AREAS *)
DATAREA = RECORD
     CASE  DA_TYPE OF
     SOF2    :    (SHIP    :    C6);
     SOF3    :    (QUAN    :    C4;
                   SONM    :    C6);
     CUST    :    (NAME    :    C20);
     ITEM    :    (DESC    :    C20; UPRC : C6);
END;
LINELIST = RECORD
     LL : PACKED ARRAY [1. .24] OF CHAR;
     TERM : INTEGER;
END;
CONTROLBLOCK = RECORD
     PSWD    :    C4;
     FUNC    :    C2;
     STAT    :    C2;
     DBFILE :    C4;
     LOC1    :    C4;
     LOC2    :    C4;
     KYID    :    C4;
     KEYV    :    C6;
     TERM    :    INTEGER;
END;
(*    VARIABLE DEFINITIONS *)
VAR
     CUST_LL : LINELIST;
     DA : RECORD
          DATA : DATAREA;
          TERM : INTEGER;
     END;
     CB : CONTROLBLOCK;
(*    DEFINE EXTERNAL PROCEDURE TO CALL DBMS *)
PROCEDURE DBMSYS (VAR CB : CONTROL BLOCK; VAR CBE : INTEGER;
                  VAR LL : LINELIST; VAR LLE : INTEGER);
                  VAR DA : DATAREA; VAR DAE : INTEGER);
                  EXTERNAL FORTRAN;
     .
     .
     .
      CB.DBFILE : = 'PYRL';
      CB.PSWD : = 'BOSS';
      CB.FUNC : = 'OF';
      CB.KYID : = 'EXCL';
      DBMSYS (CB, CB. TERM, CUST_LL, CUST_LL.TERM, DA, DA. TERM);
```

Figure 4-7. Pascal File Function

Read functions are normally performed using the RLSE disposition option. The data line read with this option is always released after the DML call terminates. When a read function is performed using the HOLD disposition option, the data line read is held until execution of a write or delete function. If no subsequent write or delete function is performed on the held data line, that line can be released by using the release line (RL) function or by executing a read with hold on another data line. The new data line is then held. Any held data line is released when a program terminates.

To obtain the primary key value associated with a data line during any read function, specify the primary key ID in the field ID portion of a line list. When using this feature, allocate space for the primary key value in the data area.

CAUTION

Do not modify the location pointers (loc1 and loc2) to any value other than asterisks (****) unless a DML function explicitly calls for such modification. Inappropriate modifications can seriously damage the file or produce unpredictable results. The only exceptions are when you place the location pointer results of one DML call into the control block for another DML call or when you swap the contents of loc1 and loc2.

4.3.2.1 Read Forward (RF). The read forward (RF) function reads the next data line of the line type associated with the key ID and key value specified in the control block parameter. The primary purpose of the RF function is to retrieve data fields from a specified data line. You can use either a primary or a secondary key to perform an RF function.

The RF function locates a data line for the line type specified in the line list and scans that data line for the specified fields or groups. The requested data items are then returned via the data area parameter.

When multiple line types are specified in the line list parameter, an asterisk is returned (instead of a comma) in the return indicator field of the line type to indicate the particular data line that the call is returning. A comma is returned in all other return indicators for all other line types specified.

Each DML call returns only one data line, even if the line list parameter specifies multiple line types. The data line returned is the first data line encountered whose line type matches one of the line types specified.

The following specifications apply to the location pointers at call initiation:

- Loc1 must contain either asterisks or a valid address from a previous call.
- If loc1 contains asterisks, the data lines associated with the specified key are searched from first to last until the line type specified is encountered. Otherwise, the data lines are read beginning with the address specified in loc2 until the line type specified is encountered.

The following specifications apply to the location pointers at call termination:

- If both loc1 and loc2 contain asterisks, either the data line of the specified line type does not exist or no more data lines of the type specified exist. If the value does not exist, an NK exception code is returned; otherwise, the status code contains asterisks.
- Loc1 normally contains the address of the data line read.
- Loc2 normally contains the address of the next data line.
- When only loc2 contains asterisks, the data line returned is the last data line for the specified key or record.

When changing primary or secondary keys, set loc1 to asterisks prior to reading the desired data line. This ensures that the first data line is read and helps prevent further processing problems.

4.3.2.2 Read Backward (RB). The read backward (RB) function reads the preceding data line of the type associated with the key value in the control block parameter. The primary purpose of the RB function is to retrieve data fields from a specific data line. An RB function can use either a primary or a secondary key.

The RB function scans the data line located for the line type specified in the line list, looking for the desired fields or groups. The requested data items are then returned in the data area parameter.

When multiple line types are specified in the line list parameter, an asterisk is returned (instead of a comma) in the return indicator field of the line type to indicate the data line that the call is returning. A comma is returned in all other return indicators for all other line types specified.

Each DML call returns only one data line, even if the line list parameter specifies multiple line types. The data line returned is the first data line encountered whose line type matches one of the line types specified.

When changing primary or secondary keys, set loc1 to asterisks prior to reading the desired data line. This ensures that the correct data line is read and helps prevent further processing problems.

The following specifications apply to the location pointers at call initiation:

- Loc1 must contain either asterisks or a valid address from a previous call.

- If loc1 contains asterisks, the call searches the data lines associated with the specified key from last to first until the line type specified is encountered. Otherwise, the call reads the data lines backwards, starting with the address specified in loc2, until the line type specified is encountered.

The following specifications apply to the location pointers at call termination:

- If both loc1 and loc2 contain asterisks, either a data line of the specified line type does not exist for the specified key or no more data lines exist for the type specified.
- Loc1 normally contains the address of the data line read.
- Loc2 normally contains the address of the next data line; it points to the data line preceding the one just read.
- When only loc2 contains asterisks, the call returns the first data line for the specified key or record (analagous to the last in the RF function).

4.3.2.3 Read Serial (RS). The read serial (RS) function reads the next data line in the file of the line type specified in the call parameter list. This function reads the data lines in the order in which they are found, ignoring the logical structure of data records. The RS ignores the key value field, but a valid key ID must be specified.

The RS function is valid for all line types. This function scans the first data line encountered for the line type specified in the line list parameter, looking for the desired fields or groups. To find this data line, the function searches the file serially from the current location until it finds a data line of the specified type or until it encounters the end of the data area. An RS may read the entire file without finding a data line of the type specified. The call returns the requested data items via the data area call parameter. The value of the primary key associated with the data line read is returned in the key value field of the control block.

Each RS call returns only one data line, even if multiple line types are specified in the line list parameter. An asterisk is returned in the return indicator field of the line list (instead of a comma) to indicate the data line being returned. The data line returned is the first data line encountered whose line type matches one of the line types specified.

The following specifications apply to the location pointers at call initiation:

- Loc1 must contain either asterisks or a valid address from a previous call.
- If loc1 contains asterisks, the call searches the appropriate data lines in physical storage order, without regard to logical record associations, in a first to last sequence until the specified line type is encountered.
- If loc1 contains a valid address, the call searches the data lines, starting with the address in loc1, until a data line of the type specified is encountered.

The following specifications apply to the location pointers at call termination:

- Loc1 normally contains the address of the data line read. When loc1 contains asterisks, either a data line of the specified line type does not exist or no more data lines exist for the requested line type.
- Loc2 points to the end of the data area.

NOTE

The value of the primary key for the data line found is returned in the key value field of the control block parameter. The key value field must be large enough to contain the key; otherwise, the key value is truncated to fit into the field specified.

4.3.2.4 Read Ascending (RA). The RA function reads key values from the key storage area and returns the key values in ascending order. RA does not transfer data other than key values. (For key types that are not sequential, RA returns key values in a duplicatable random order.) In addition to retrieving keys in sorted order, RA can start from any point in the key area. The key value specified in the control block defines the starting point (starting positions apply only to sequential keys). By manipulating the location pointers (loc1 and loc2), you can describe any of the following starting positions:

- Start at key equal to the specified value
- Start at key greater than the specified value
- Start at key greater than or equal to the specified value
- Start at the lowest key value

Table 4-2 shows the use of location pointers to specify the starting point of the RA operation. When the RA function terminates, loc1 points to the key just read and loc2 points to the first data line of the key. The retrieved key value is placed in the key value area of the control block. Consequently, you can perform successive RA operations without resetting the control blocks. (The key value just read becomes the initial key value for the next RA operation.)

Table 4-2. RA Starting Location Pointers

Loc1	Loc2	Starting Point
****	****	Lowest key value
****	xxxx	Equal key value
xxxx	xxxx	Greater than key value
xxxx	****	Greater than or equal to key value

Note:

**** indicates that the location pointer is set to asterisks, and xxxx indicates that the pointer is set to some valid address or binary zeros.

The most convenient method of using the RA function is to provide one control block for retrieving keys and another for retrieving data lines. You can set the location pointer to binary zeros by equating it to a double precision (4-byte) integer constant with the value of zero.

4.3.2.5 Read Descending (RD). The RD function reads key values from the key storage area and returns the key values in descending order. RD does not transfer data other than key values. (For key types that are not sequential, RD returns key values in a duplicatable random order that is the same as the RA function.) In addition to retrieving keys in sorted order, RD can start from any point in the key area. The key value specified in the control block defines the starting point (starting positions apply only to sequential keys.) By manipulating the location pointers (loc1 and loc2), you can describe any of the following starting positions:

- Start at key equal to the specified value
- Start at key less than the specified value
- Start at key less than or equal to the specified value
- Start at the highest key value

Table 4-3 shows the use of location pointers to specify the starting point of the RD operation. When the RD function terminates, loc1 points to the key just read and loc2 points to the first data line of the key. The retrieved key value is placed in the key value area of the control block. Consequently, you can perform successive RD operations without resetting the control blocks. (The key value just read becomes the initial key value for the next RD operation.)

Table 4-3. RD Starting Location Pointers

Loc1	Loc2	Starting Point
****	****	Highest key value
****	xxxx	Equal key value
xxxx	xxxx	Less than key value
xxxx	****	Less than or equal to key value

Note:

**** indicates that the location pointer is set to asterisks; xxxx indicates that the pointer is set to some valid address or binary zeros.

The most convenient method of using the RD function is to provide one control block for retrieving keys and another for retrieving data lines. A location pointer can be set to binary zeros by equating it to a double precision (4-byte) integer constant with the value of zero.

4.3.2.6 Partial Key Search. The sequential key capability allows a partial key search. If you specify the beginning portion of a key value, DBMS-990 will find the first key that fits the description. For example, suppose the DDL for an employee file describes a name field as a 40-character field that is a secondary key. Suppose further that you want to change an employee's address but can remember only that the employee's name begins with CH. Perform the following steps to retrieve the first employee whose name field begins with CH:

1. Initialize the key value area of the control block to binary zeros.
2. Move the characters CH into that key value area.
3. Perform an RA function to retrieve the first name field beginning with CH.
4. Perform subsequent RA functions with the same control block to retrieve the next sequential name fields beginning with CH.

4.3.2.7 Hold Line (HL). The hold line function places a hold on the data line to which the location address in loc1 refers. Using this function, you need not reread a data line by using the HOLD disposition in the line list in order to write (WT) or delete (DL) a data line. The function does not replace or eliminate the HOLD disposition. Since this function does not read or transfer data, the line list and data area call parameters are not used; you can replace them with dummy call parameters in the call statement. The DML function code is HL.

The data line to which loc1 refers is verified as existing; if it does not exist, it is assumed to have been deleted (by another task) and a status of invalid key (IK) is returned.

The location pointers at call initiation are as follows:

- Loc1 must contain a valid address from a previous call; it cannot contain asterisks. If it does contain asterisks, a status of lock line (LL) is returned.
- Loc2 must contain either asterisks or a valid address from a previous call.

The location pointers at call termination are as follows:

- Loc1 is unchanged by the HL function.
- Loc2 is unchanged by the HL function.

CAUTION

Use the hold line (HL) function carefully in an interactive environment. If more than one program is accessing a particular line in the data base at the same time, the data in the line may change between the time the user executes a read and the time the HL function is executed.

4.3.2.8 Release Line (RL). The release line (RL) function releases any held data line associated with a task. Since only one line can be held by a task at one time, the location pointers need not contain the address of the held line. If the task is currently holding no line, the function terminates with a status code of asterisks (no exception). Since this function does not read or transfer any lines, the line list and data area call parameters are not used; you can replace them with dummy parameters in the call statement. The DML function code is RL.

Using this function, you need not reread a data line by using the RLSE (release) disposition in order to release the hold on a data line. A hold on a data line can still be released by rereading the line with a RLSE disposition, by reading a different line using the HOLD disposition, or by writing to (WT) or deleting (DL) the line.

The location pointers at call initiation are as follows:

- Loc1 must contain either asterisks or a valid address from a previous call.
- Loc2 must contain either asterisks or a valid address from a previous call.

The location pointers at call termination are as follows:

- Loc1 is unchanged by the RL function.
- Loc2 is unchanged by the RL function.

4.3.3 Update Functions

Provided they terminate successfully, the DML update functions always change the data content or logical structure of the file. The data in the call parameters either replaces an existing data line or adds a new data line. To delete data lines, use the delete function. The DML update functions can manipulate only one data line at a time, except for the delete record (DR) function.

Check the status of each update function after the DML call has completed to ensure that the function has terminated properly. The status code should contain asterisks. (The DR function returns asterisks as the status code even if no record existed for the specified key.) If an error condition occurs, refer to Appendix A.

CAUTION

Do not modify the location pointers (loc1 and loc2) to any value other than asterisks (****) unless a DML function explicitly calls for such modification. Inappropriate modifications can seriously damage the file or produce unpredictable results. The only exceptions are when you place the location pointer results of one DML call into the control block for another DML call or when you swap the contents of loc1 and loc2.

All update functions require the RLSE disposition option. These functions ignore the HOLD option. If HOLD is specified, release is performed. Perform a read with HOLD option before performing a write or delete function. If you execute these functions without a prior read with HOLD option, an exception status code other than asterisks is returned. Refer to Appendix A.

4.3.3.1 Add After (AA). The add after (AA) function inserts a data line of the line type specified in the line list call parameter into the file after the data line to which loc1 refers. The primary use of this function is to add lines to a file. The DML function code is AA. Each AA call can add only one data line.

The AA function adds data lines only to primary keys. If no data lines currently exist for the specified primary key, a new data record is created using the key value of the control block and the line list and data area parameters. Secondary key values are added automatically when the added data line contains secondary keys.

To insert the data line after a specific data line, perform a read function prior to the add; as a result, the specified data line is found and its location pointer is placed in loc1. To add data lines in a series, do not reload loc1 with asterisks.

If an AA function is performed with multiple line types specified in the line list, an asterisk must appear in the return indicator field of one of the line types to indicate which line type is to be added. If no asterisk appears, the first line type specified is added. When more than one return indicator contains an asterisk, the first line type that contains an asterisk in its return indicator is added.

NOTE

When adding a line type 01, always set loc1 to asterisks; otherwise, an error occurs. A line type 01 need not be present or defined within a file. However, when it is defined, it must be the first line type added for each record.

The following specifications apply to the location pointers at call initiation:

- Loc1 must contain either asterisks or a valid address from a previous call.
- If loc1 contains asterisks, the data line is added after the last data line associated with the specified key. Otherwise, the data line is added after the data line whose address is specified in loc1.

The following specifications apply to the location pointers at call termination:

- Loc1 contains the address of the added data line.
- Loc2 normally contains the address of the next data line (the one after the added data line).
- If loc2 contains asterisks, no data line exists after the added data line.

4.3.3.2 Add Before (AB). The add before (AB) function inserts a data line of the line type specified in the line list call parameter into a file before the data line to which loc1 refers. The primary use of the function is to add data lines to a file. The DML function code is AB. An AB call can add only one data line.

The AB function adds data lines only to primary keys. If no data lines currently exist for the key, a new data record is created by using the key value of the control block and the line list and data area parameters. Secondary key values are added automatically when the added data line contains secondary keys.

To insert a data line before a specific data line, perform a read function prior to the add; as a result, the specified data line is found and its location pointer is placed in loc1. To add data lines in a series, do not reload loc1 with asterisks.

If an AB function is performed with multiple line types specified in the line list, an asterisk must appear in the return indicator field of one of the line types to indicate which line type to add. If the asterisk does not appear the first line type specified is added. When more than one return indicator contains an asterisk, the first line type that contains an asterisk in its return indicator is added.

The following specifications apply to the location pointers at call initiation:

- Loc1 must contain either asterisks or a valid address from a previous call.
- If loc1 contains asterisks, the data line is inserted before the first data line of the record associated with the specified primary key. Otherwise, the data line is inserted before the data line whose address is specified in loc1.

The following specifications apply to the location pointers at call termination:

- Loc1 contains the address of the added data line.
- Loc2 normally contains the address of the next data line (the one after the added data line).
- Loc2 contains asterisks when no data line exists after the added data line.

4.3.3.3 Write (WT). The write (WT) function replaces the data line to which loc1 refers with the data line specified in the call parameters. The primary use of this function is to update the field contents within existing data lines.

Perform a read with HOLD disposition option prior to a WT function. The held data line is used to rewrite an updated data line. Only the fields specified in the line list are replaced. Any other fields in the held data line are not changed. The held data line is released upon termination of the WT function.

If a WT function is performed with multiple line types specified in the line list, an asterisk must appear in the return indicator field of one of the line types to indicate which line type to replace. The asterisk should appear in the appropriate return indicator as a result of the read with HOLD option. When more than one return indicator contains an asterisk, the first line type that contains an asterisk in its return indicator is used. If an asterisk does not appear in the line list, an exception condition is reported in the status field of the control block.

The WT function cannot modify secondary key values. You can delete the data line containing the secondary key value and then add the line to the file with the new key value by using either the AA or the AB function. However, you can update a line without changing its secondary key.

At call initiation, loc1 must contain the address of the data line to be updated. This is obtained from the read with hold operation prior to the WT function.

The following specifications apply to the location pointers at call termination:

- Loc1 is unchanged by the WT function.
- Loc2 is unchanged by the WT function.

4.3.3.4 Delete (DL). The delete (DL) function deletes the data line whose address is in loc1. The primary use of this function is to delete data lines from the data base files. The DML function code is DL.

Perform a read with HOLD disposition option before a DL operation. The held data line is released upon termination of the DL function. When the data line to be deleted is the last data line for a record, the data record and the primary key are also deleted.

If a DL function is performed with multiple line types specified in the line list, an asterisk must appear in the return indicator field of one of the line types to indicate which line type to delete. The asterisk should appear as a result of the read with HOLD option. When more than one return indicator contains an asterisk, the first line type that contains an asterisk in its return indicator is used. If an asterisk does not exist in the line list, an exception condition is reported in the status field of the control block. As a result of the deletion, DBMS-990 can reuse the line space.

At call initiation, loc1 must contain the address of the data line to be deleted. This is accomplished by performing the read with HOLD operation prior to the DL operation.

The following specifications apply to the location pointers at call termination:

- Loc1 normally contains the address of the data line that preceded the deleted data line. Loc1 contains asterisks when no data line precedes the deleted data line.
- Loc2 is unchanged by the DL function.

4.3.3.5 Delete Record (DR). The delete record (DR) function deletes a record by deleting all of the data lines associated with the primary key value specified in the control block. The key ID and key value specified in the control block apply to a primary key. This function allows you to easily delete entire records without executing both a read (with hold) and delete function for each data line in the record. When the last data line is deleted for the record, the primary key value specified is also deleted. The DR function returns asterisks as the status code even if no record existed for the specified key.

The location pointers at call initiation are as follows:

- Loc1 must contain asterisks.
- Loc2 must contain asterisks.

The location pointers at call termination are as follows:

- Loc1 is unchanged by the DR function.
- Loc2 is unchanged by the DR function.

4.3.4 Transaction Functions

Three DML functions used in defining transactions are available for DML applications if the transaction-level integrity feature has been chosen at the time of data base generation. A transaction is a series of operations or updates to a data base that logically belong together, such as the individual operations performed to transfer funds from one account to another.

The transaction functions mark the beginning and end of the operations comprising a transaction. Within a transaction, all of the updates must be performed successfully, or the entire transaction can be rolled back leaving the data base in its pretransaction state.

DBMS-990 allows transaction nesting up to a maximum level of 10. The actual level on your system is specified at DBGEN. If you exceed this level, the system returns an invalid transaction (IT) status. With nested transactions, all internal Rollback Transactions (TRs) are actually carried out. However, only the outermost TC actually commits the updates to the data base files.

Remember that there is a system performance cost associated with the use of the transaction-level integrity feature. Consequently, you should use the transaction functions only to group operations that logically belong together. Also, you are cautioned to keep the number of operations within a transaction to a minimum to reduce both the amount of memory and the execution time needed to support this feature.

The transaction functions are as follows:

- Start transaction (TS)
- Commit transaction (TC)
- Rollback transaction (TR)

4.3.4.1 Start Transaction (TS). The start transaction (TS) function marks the beginning of a transaction. All operations that follow, up to the occurrence of either a TC or TR function, are defined as belonging to the transaction.

4.3.4.2 Commit Transaction (TC). The commit transaction (TC) function marks the end of a transaction and causes all operations occurring since the last TS function to be applied to the data base. In the case of nested transactions, only the outermost TC causes the updates to be applied to the data base.

4.3.4.3 Rollback Transaction (TR). The rollback transaction (TR) function causes all operations occurring since the last TS function to be nullified. The data base is returned to its original pre-transaction state.

The system performs the TR function in three situations, as follows:

- When two transactions are deadlocked
- When the computer memory workspace is not large enough to accommodate the number of locks requested
- When the system crashes

The system returns a deadlock status (DL) or lock tables (LB) code in the first two cases.

NOTE

It is the programmer's responsibility to check for the DL status code and take appropriate action to ensure that the application program user is aware that the transaction was not committed. It is advisable to check for a DL status after every TC.

The following example illustrates a control block for a transaction function:

```
01      CONTROL BLOCK.
                  02      PSWD        PIC X(4)        VALUE "DBMS".
                  02      FUNC        PIC XX          VALUE "TS".
                  02      STAT        PIC XX          VALUE "**".
                  02      FLID        PIC X(4)        VALUE "PYRL".
                  02      FILLER      PIC X(8)        VALUE "********".
                  02      KYID        PIC X(4)        VALUE "EXCL".
                  02      KEYV        PIC X(40)       VALUE "EXCL".
01      ASK                             PIC X         VALUE "*".
.
CALL "DBMSYS" USING CONTROL-BLOCK, ASK, ASK, ASK, ASK, ASK.
```

In case of deadlock, the following example illustrates the most efficient method of structuring a restart:

```
LABEL:      START TRANSACTION
            .
            .
            .
            .
            COMMIT TRANSACTION
            IF   DB_STAT = 'DL' GO TO LABEL
```

The programmer can also initiate the TR function. A program can perform a series of DML operations, test the result, and commit the transaction only if certain conditions are met. Using this technique, the programmer can conditionally specify that the preceding operations are rolled back.

NOTE

With transaction-level integrity, a pre-image log is maintained on the system. This pre-image log records data prior to updates. If an error occurs during a write or read from the pre-image log, the following message is written to the system log: PREIMAGE FILE ERROR.

5

Security

5.1 INTRODUCTION

Security is an optional feature of DBMS-990 that can be included during installation of DBMS-990. Security uses passwords to limit unauthorized use of the data base. Although the security system cannot eliminate all violations, it does aid in controlling access to a data base.

Security requires a certain amount of overhead. The amount required depends on the degree of protection assigned to the data elements in the data base. While the degree of protection affects performance, the number of passwords affects storage requirements.

5.2 PASSWORDS

You can access a file or data base only if your password is associated with that file or data base. Each password is associated with one or more files; a single data base might have one password that applies to all of its files, or it might have several passwords distributed between the related files of the data base. The primary purpose of passwords is to assign file access. For example, employees in a personnel department might be assigned access only to personnel information while those in payroll have access only to payroll information.

Various SCI commands assist in maintaining password entries and initiating security. (Refer to the *Model 990 Computer DNOS Data Base Administrator User's Guide*.)

5.3 ACCESS AUTHORIZATION

Access authorization defines the type of access allowed to the data elements of a file for a particular password and/or user. Authorization must be assigned to each file associated with a password. The following access types are available for each file:

- Read
- Write (replace)
- Add
- Delete

These access types are combined to form an authorization code. For example, one password may specify read access to the data within a file. This same password may contain authorization to read, write, and add data in another file so that the user of this password can only read the first file but can read, write, and add data to the second file. Almost any combination of read, write, add, and delete is permissible. However, any authorization code that includes write or delete must also contain read.

Authorization codes can apply to all levels of data. In the absence of assigned authorization codes, lower-level data elements assume the authorization of the next higher data element. For example, a line assumes the authorization code of the file, and a field assumes the authorization code of the line. To avoid this, you can usually assign less access authorization (including no access) to a line or field. However, if a line has delete authorization all fields on that line must also have delete authorization.

In security checking, no distinction is made between the access authorization of a group and one of its fields. The DBA (or whoever assigns passwords) resolves any conflicts in access authorization between a field and its group.

6

Primitive Query

6.1 INTRODUCTION

This section contains the information necessary to operate primitive query in order to display data base information in a limited manner. The following paragraphs discuss commands, examples of queries, and error messages.

6.2 PRIMITIVE QUERY (PQUERY) COMMAND

The Primitive Query (PQUERY) command provides you with a limited capability to retrieve and display information stored in a data base without writing a program. PQUERY is primarily a debugging tool, used to view the contents of the data base during application program development. PQUERY provides the following functions: read forward (RF), read backward (RB), and read serial (RS).

6.2.1 PQUERY User Interface

You must resolve two screens in order to use PQUERY. The first screen is used only once per session, but the second screen is repetitive.

The first screen for the PQUERY command is as follows:

```
PRIMITIVE QUERY
                    PASSWORD:
          LISTING ACCESS NAME:
```

In response to the prompt LISTING ACCESS NAME, enter the pathname to which the output will be sent. Pressing the RETURN key in response to this prompt displays the output at the VDT. To change the output access pathname, terminate the current PQUERY session, reexecute the PQUERY command, and enter a new pathname in response to the prompt. In response to the prompt PASSWORD, enter the appropriate user password. The password must include at least read authorization for any desired group or field. This prompt is displayed only when security is part of DBMS-990.

The second screen for the PQUERY command is as follows:

```
PRIMITIVE QUERY
                      FUNCTION:
                     DB FILE ID:
                         KEY ID:
                      KEY VALUE:
                      FIELD IDS:
            NO. OF OUTPUT LINES:
                     TERMINATE:  YES
```

This PQUERY screen is repetitive and will be reissued to request new responses until you reply YES to the prompt TERMINATE.

In response to the prompt FUNCTION, enter RF for read forward, RB for read backward, or RS for read serial.

In response to the prompt FIELD IDS, enter a list of field and group IDs. When a group ID is specified, the data is displayed in hexadecimal format. All fields and groups specified must belong to the same line type.

In response to the prompts KEY ID and KEY VALUE, enter the ID of the key field and the key value from which you will select lines for output. The key ID is required for all functions. The key value is required for the RF or RB functions. For the RS function, the primary key value is returned for the given key ID and is displayed as the first field of output. The key value response is not required for an RS function (press RETURN).

For an RF or RB function, the response to the prompt NO. OF OUTPUT LINES determines the maximum number of lines to be retrieved and displayed for the specified primary key. If less than the maximum number of lines are found, only that many lines are displayed. For an RS function, this response determines the number of data lines to be retrieved and displayed at one time for the field and/or group ID(s) specified.

6.2.2 PQUERY Output

If the response to the prompt TERMINATE is YES, the PQUERY command terminates following the display of the data. Current positioning information is lost. If the response is NO, the command does not terminate and positioning information is not lost. The second screen reappears. If you change no prompt responses other than the response to NO. OF OUTPUT LINES, reexecution of an RF or RB function retrieves and displays the next data line of the specified key record. An RS function always retrieves the next line. When all data lines containing a specified field or group type are exhausted, the message END OF DATA LINE(S) appears.

The output report consists of three parts. The first part contains information extracted from the second screen prompt responses. The second part identifies the line number associated with the data and provides the field headings in the order that the field IDs were listed. The third part consists of the data listed under the field name headings.

When the number of characters in the output line exceeds 80, data carries over to the next output line. Field headings are displayed for the portion that carries over. Fields that are not in ASCII format appear in hexadecimal format.

6.3 EXAMPLE QUERIES

The following is an example of an RF function, showing the user prompts and responses, along with the associated output:

```
==================================================
PRIMITIVE QUERY
            FUNCTION: RF
             FILE ID: CUST
              KEY ID: CUSN
           KEY VALUE: D0001
           FIELD IDS: NAMESTRTCITYSTATZIPCCRED
NO. OF OUTPUT LINES: 50
           TERMINATE: YES
==========================================
     LINE TYPE IS: 01

CUSN      NAME                    STRT                    CITY
STAT    ZIPC  CRED
D0001   HOLE EARTH DIST.          1234 MOUNTAIN LN.       LITTLE HILL
TX      78123  A1
```

The following is an example of an RS function and illustrates the hexadecimal output obtained for a data format other than CH (see the ITEM file DDL for the example program in Appendix B). The user prompts and responses, along with the associated output are displayed.

```
================================================================
PRIMITIVE QUERY
            FUNCTION: RS
              FILE ID: ITEM
                KEY ID: ITMN
           KEY VALUE:
           FIELD IDS: DESCUPRCQTYOQTYH
NO. OF OUTPUT LINES: 50
           TERMINATE: YES
================================================
     LINE TYPE IS: 01

ITMN  DESC                    UPRC          QTYO      QTYH
A001  ARMADILLOS              313030313233  30303031  30313233
B002  BLACK HOLES             303230323334  30303032  30323334
C003  CLAY                    303033333435  30303033  30333435
D004  DIPS                    303030343536  30303034  30343536
E005  ERECTORS                303035353637  30303035  30353637
F006  FREEBIES                303630363738  30303036  30363738
G007  GOOBERS                 373030373839  30303037  30373839
H008  HERBS                   303830383930  30303038  30383930
I009  IDIOMS                  303039393030  30303039  30393030
J010  JUMPS                   303031303030  30303130  31303030
K011  KILNS                   303131313030  30303131  31313030
L012  LONE STARS              313230303030  30303132  31323030
N014  NIBBLES                 303431343030  30303134  31343030
S019  SHOVELS                 313031393031  39303139  31393030
T020  TALES                   303939303939  30313030  34303030
Y025  YARNS                   303235303030  30303235  32353030
Z026  ZEBRAS                  303032363030  30303236  32363030
=======  END OF DATA LINES =======
```

6.4 ERROR MESSAGES

PQUERY provides two types of error messages. The following messages are displayed in response to errors in procedure specifications:

1. INVALID PASSWORD.
2. INVALID KEY VALUE.
3. NO FIELD OR GROUP NAME SPECIFIED.
4. UNABLE TO OPEN OUTPUT FILE.
5. STATUS EXCEPTION FROM DBMS, STATUS = XX. (refer to Appendix A for meanings of DBMS status codes)
6. FIELD XXXX IS UNDEFINED OR NOT IN SAME LINE-TYPE.
7. ILLEGAL FUNCTION, XX, MUST BE RS, RF OR RB.
8. FIELDS ARE ON DIFFFERENT LINES, BAD FIELD = XXXX.
9. BAD FIELD NAME XXXX.

Errors that the operating system discovers are reported by an error message of the following form:

OPERATING SYSTEM ERROR XXXX

where:

XXXX is the four-character system error code.

7

Execution of Application Programs

7.1 INTRODUCTION

This section discusses the steps required to produce a running application program using DBMS-990 and outlines the procedure necessary to run the data base system. It is assumed that DBMS-990 has been generated.

Data base users fall into two categories: those with a data base administrator (DBA) and those without a DBA. The DBA designs the data structures, assigns security passwords, and maintains the system. The *Model 990 Computer DNOS Data Base Administrator User's Guide* assists the DBA in performing these functions. If a DBA is not assigned, the programmer should read the DBA manual and perform the DBA's duties.

The following paragraphs describe preliminary procedures, programming considerations, and operation of DBMS-990. It is assumed that the data structures have been designed.

7.2 PRELIMINARY PROCEDURES

The preliminary procedures involve two steps: creating files and security considerations.

7.2.1 File Creation

Data base files are created using the DDL compiler and the DDL for the appropriate file. Once the design of the data structures is accomplished, file structures are translated into a series of DDL statements. Use the Text Editor to enter these DDL statements into the computer, and to make corrections. To initiate the DDL compiler, use the DDL command, as outlined in Section 3.

You must be aware of the structure and definition of the data in order to write effective programs. The line types defined, the secondary keys, and the field IDs are required in most applications. Usually, the DBA provides this information.

7.2.2 Security

Security is an optional feature of DBMS-990. When security is generated into the system, every access to a data base file must include a valid password. Once the DDL compiler creates the data base file, the DBA must assign passwords to the file. The *Model 990 Computer DNOS Data Base Administrator User's Guide* specifies the SCI commands for maintaining the password files.

7.3 COMMON PROGRAM CONSIDERATIONS

Program considerations consist of the coding of DML parameters (or blocks), call techniques to DBMS-990, and exception processing and optimization. The COBOL language is used in the examples included in this section. Appendix B contains example programs in Pascal and COBOL.

7.3.1 Coding of DML Parameters

In a COBOL program, the DML call parameters are coded in the working-storage section of the data division. The three main parameters for the call are the control block, the line list, and the data area. The other parameters are merely end indicators for the main parameters.

NOTE

DBMS-990 modifies each DML parameter to optimize future calls. The control block and line list parameters are coded as described in Section 4.

7.3.1.1 Control Block. To minimize storage requirements, define a limited number of control blocks for each file. In most control blocks, only the function code need be changed from one call to the next for the same file.

7.3.1.2 Line List. In coding line lists, avoid changing any of the fields except the return indicator and disposition fields. DBMS-990 modifies the field IDs portion of the line list on the initial call and each time that area changes. Frequent user modification of the field IDs portion of the line list slows execution of the application program.

Figure 7-1 shows an example of a coded line list parameter. If one retrieval is for FLD1, one is for FLD3, and another is for FLD2, retrieve all three fields in one call, as shown. Even if you have to retrieve additional fields with a line type, this method saves time when compared to the time required to make individual calls to DBMS-990.

```
01      LINE-DML.
        05 LINE-ID          PIC X(7)     VALUE "LINE=01".
        05 RETURN-IND       PIC X        VALUE ",".
        05 FIELDS           PIC X(12)    VALUE "FLD1FLD2FLD3".
        05 DISPOSITION      PIC X(8)     VALUE "****RLSE".
```

Figure 7-1. Line List Example

The use of multiple line types in a line list deserves some consideration. Generally, if accessing more than one line type, a line list that specifies multiple line types requires the fewest DML calls. The first encounter of any specified line type sets the return indicator for that line type to *.

For some applications, it might be convenient to include one line list that specifies a single line type and another that specifies multiple line types. Use the line list best suited for each data access.

7.3.1.3 Data Area. Data to be sent or received is contained in the data area. This area can be defined as a single area or as multiple areas with multiple definitions. The only restriction is that the actual parameter must be defined on a word boundary, as with a COBOL 01-level definition. When memory considerations are important, use a single data area for most of the calls to DBMS-990. In this case, redefinitions are necessary so that all line types and their data elements may be processed. Figure 7-2 shows an example of a single data area. Figure 7-3 shows an example of multiple data areas. Note that with multiple data areas the DML calls use different data areas rather than the same data area.

7.3.2 Call Techniques to DBMS-990

When security is installed, you must provide the appropriate file password to DBMS-990. The password can be hard-coded into the application program, solicited from you or obtained through an input parameter. When file-access checking is installed in DBMS-990, files must be opened with the appropriate file access specified before any DML functions can be executed.

```
01      DATA-AREA.
            05 FILLER                  PIC X(40)        VALUE SPACES.
01      DA-LINE-01 REDEFINES DATA-AREA.
            05 FLD1                    PIC X(10).
            05 FLD2                    PIC X(15).
            05 FLD3                    PIC X(15).
01      DA-LINE-02 REDEFINES DATA-AREA.
            05 ACCOUNT                 PIC 9(10).
            05 DESCRIP                 PIC X(10).
            05 AMOUNT                  PIC 9(8)V99.
            05 FILLER                  PIC X(10).
```

Figure 7-2. Example of Single Data Area

```
01      DATA-AREA-01.
        05 FLD1                PIC X(10).
        05 FLD2                PIC X(15).
        05 FLD3                PIC X(15).
01      DATA-AREA-02.
        05 ACCOUNT             PIC 9(10).
        05 DESCRIP             PIC X(10).
        05 AMOUNT              PIC 9(8)V99.
```

Figure 7-3. Example of Multiple Data Areas

Prior to calling DBMS-990, initialize the control block and line list parameters. Appendix B contains examples of complete calls to DBMS-990. To conserve memory, use common call subroutines wherever possible. Figure 7-4 contains a common DBMS-990 call routine. Since the same or common control block, line list, and data area are utilized, a PERFORM is used instead of individual calls to DBMS-990. The only portion that needs to be altered prior to the PERFORM is the function code. This application updates all current lines of a line type or adds new data lines. This is a special application, used for illustrative purposes only.

7.3.3 Exception Processing and Optimization

A status return code of something other than asterisks does not always signify an error condition. Figure 7-5 contains two examples of status code checking. Case A is primarily designated for adding, while case B is best suited for updating or writing. Case B is more efficient if most of the updates are changes; case A is best if most of the record transactions are additions.

7.3.4 Holding Lines

Another consideration in program design is the method of holding lines for update (write) or delete purposes. With DBMS-990, you can hold only one line at a time for update. In an interactive system with more than one user, it is important to design update procedures that do not lock out other users for excessive periods of time. Figure 7-6 illustrates the use of the HOLD disposition.

In case A of Figure 7-6, the user holds the part information from the first retrieval. All other users are locked out of that part information while the first user decides what to do. The first user must determine whether the part number is the correct one before entering an order. Any delay in entering the order delays the other users, since they cannot access the line being held.

In case B of Figure 7-6, more DBMS-990 calls are required. However, the part information is not held on the first retrieval. As a result, another user might obtain the part before the first user enters the order. Only one party can receive the parts when the quantity on hand is limited, and the first order entered should receive the parts. In effect, case B holds the part for the time it takes the computer, not a human, to make a decision. Thus, case B is a better interactive system than case A.

7.3.5 Transaction Bracketing

You should use the transaction commands (TS, TC or TR) to bracket updates that logically belong together, such as the steps required in a transfer of funds between two accounts. Including too many DML calls within a single transaction results in a deadlock. Therefore, you should limit the size of transactions. Appendix B illustrates a program that uses transactions.

```
01      CONTROL-BLOCK.
          05 PASSWORD          PIC X(4).
          05 FUNCTION          PIC XX.
          05 STATUS            PIC XX.
          05 FILENAME          PIC X(4)      VALUE "POFL".
          05 LOC1              PIC X(4).
          05 LOC2              PIC X(4).
          05 KEY-NAME          PIC X(4)      VALUE "ORDN"
          05 KEY-VALUE         PIC 9(9).
             .
             .
             .
01        INPUT-PARAM           PIC X         VALUE SPACE.
01        LINE-IND              PIC X         VALUE SPACE.
          88 LINE-FOUND                        VALUE "Y".
01        EOF-IND               PIC X         VALUE SPACE.
          88 EOF-FOUND                         VALUE "*".

PROCEDURE DIVISION
MAIN-LINE.
          ACCEPT INPUT-PARAM.
          PERFORM READ-TRANS.
          PERFORM UPDATE-ROUTINE UNTIL EOF-FOUND.
             .
             .
             .
UPDATE-ROUTINE.
          MOVE "RF" TO FUNCTION.
          PERFORM COMMON-CALL.
          IF LINE-FOUND and FLD1 = INPUT-PARAM
             .
             .
             (move in new data)
             .
             .
             MOVE "WT" TO FUNCTION,
             PERFORM COMMON-CALL,
```

Figure 7-4. Example of Common DBMS-990 Call Routine (Sheet 1 of 2)

```
        ELSE,
            MOVE "AA" TO FUNCTION,
            PERFORM COMMON-CALL.
        PERFORM READ-TRANS.
END-UPDATE-ROUTINE.
                .
                .
                .

                .
                .
COMMON-CALL.
        CALL "DBMSYS" USING CONTROL-BLOCK,
                                        END-CONTROL-BLOCK,
                                        LINE-LIST,
                                        END-LINE-LIST,
                                        DATA-AREA,
                                        END-DATA-AREA.

IF STATUS NOT = "**"
        MOVE "N" TO LINE-IND,
                .
                .
        (Error logic)
                .
                .
ELSE,
        IF LOC1 = "****" AND LOC2 = "****"
            MOVE "N" TO LINE-IND,
        ELSE,
            MOVE "Y" TO LINE-IND.
END-COMMON.
                .
                .
                .
READ-TRANS.
        READ TRANS-FILE AT END
                .               MOVE "*" TO EOF-IND.
                .
                .
                .
```

Figure 7-4. Example of Common DBMS-990 Call Routine (Sheet 2 of 2)

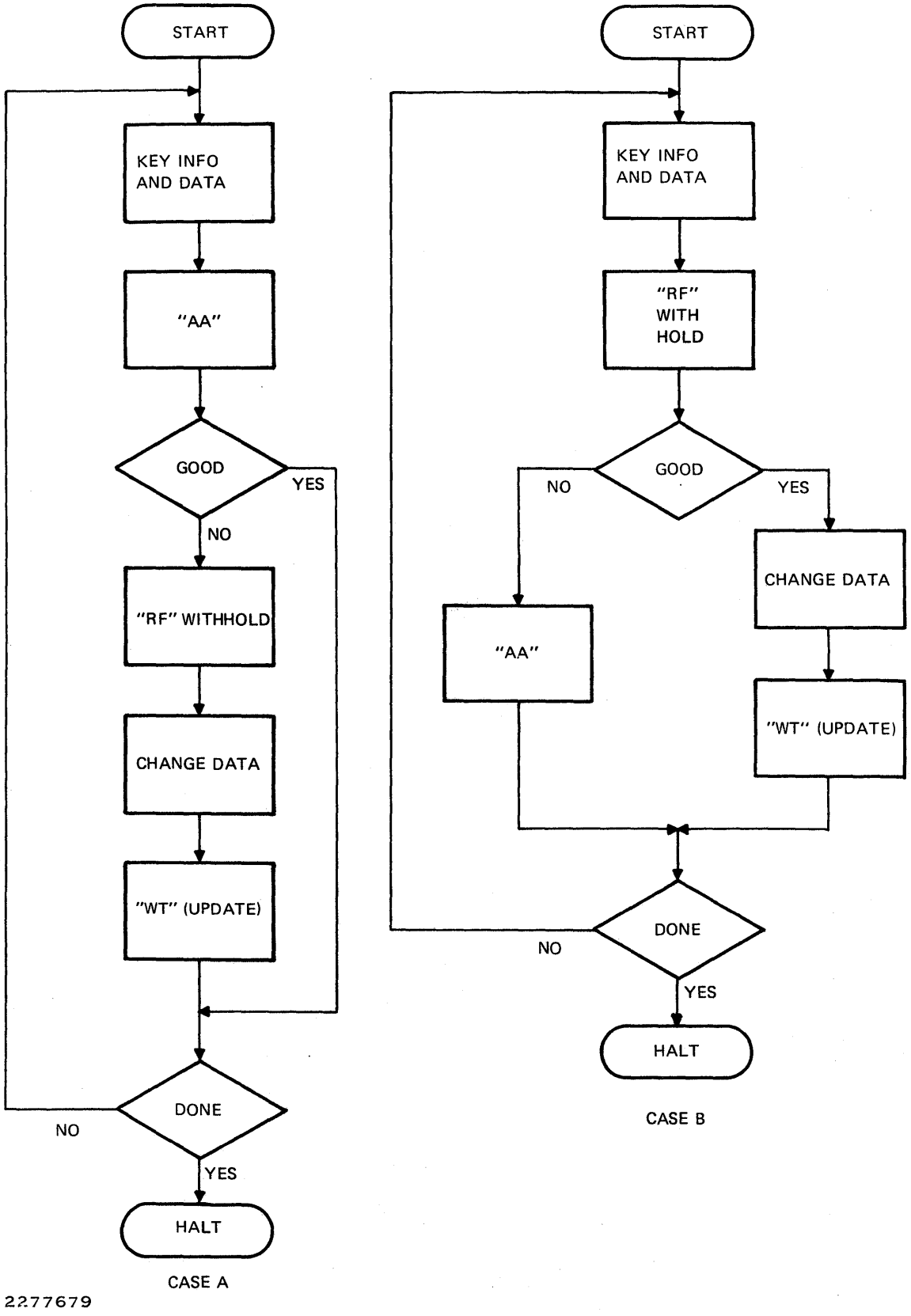


Figure 7-5. Adding and Updating

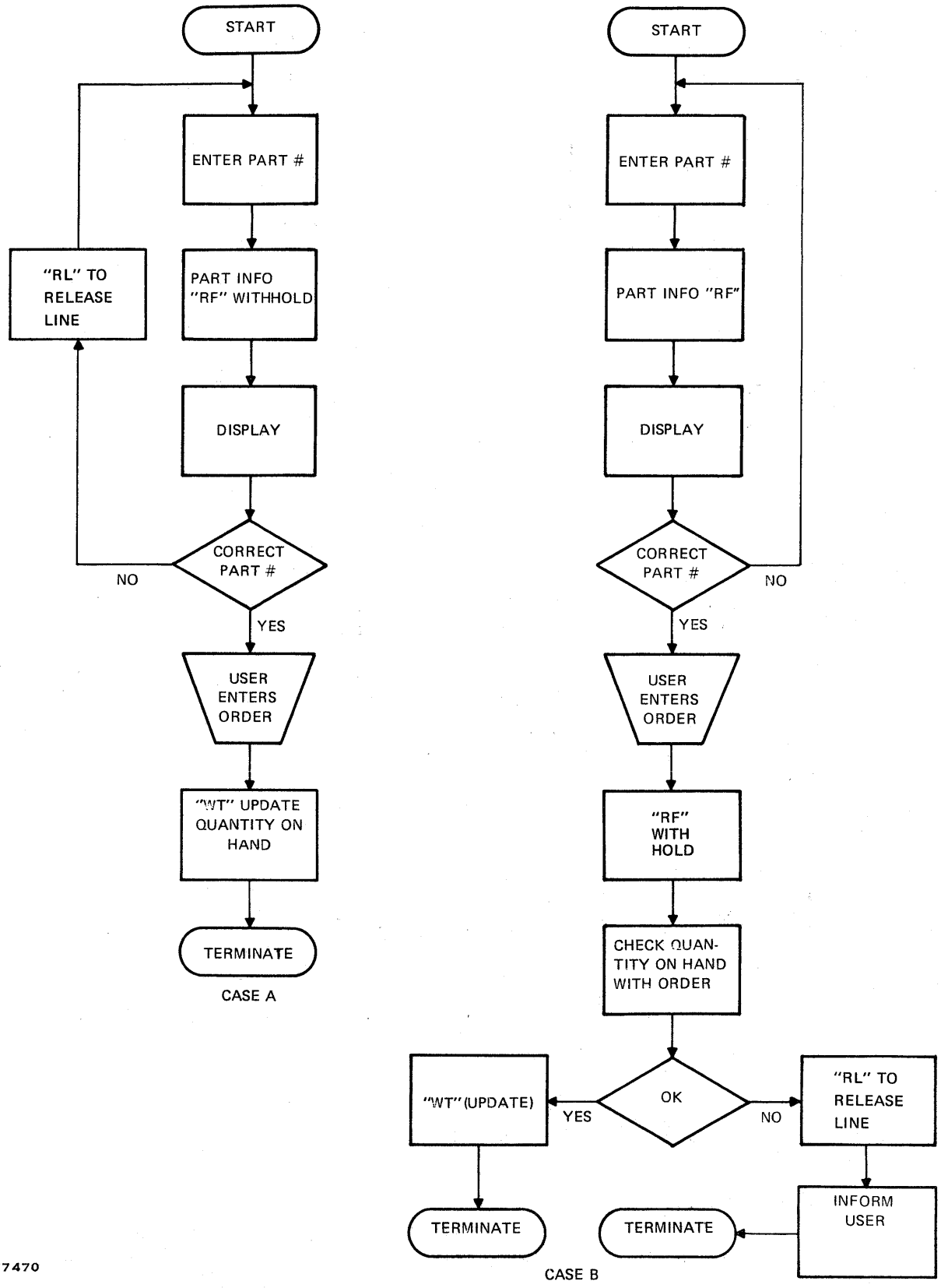


Figure 7-6. Use of HOLD Disposition

7.4 COMPILING AND LINKING COBOL

After writing the application program, the user must compile and link the program. The *Model 990 Computer DNOS COBOL Programmer's Guide* includes instructions for using the COBOL compiler. The *Model 990 Computer DNOS Link Editor Reference Manual* includes instructions for link editing. Before linking the program, have available a program file in which to place the output of the Link Editor.

To link COBOL, create the necessary link control file structure by using the Text Editor. The resulting control file is then input to the Link Editor. All entries are required except for the optional entries enclosed in brackets ([ ]); angle brackets (< >) indicate user-supplied information. Refer to Section 3 of the *Model 990 Computer DNOS Data Base Administrator User's Guide*.

In Figure 7-7, the TASKNAME option of the TASK command defines the saved name for the program file. The MAIN PROGRAM option is the user application program.

```
FORMAT IMAGE ,REPLACE
PROC RTCOBOL
INCLUDE .S$SYSLIB.RCBPRC
TASK <TASKNAME>
INCLUDE .S$SYSLIB.RCBTSK
INCLUDE .S$SYSLIB.RCBMPD
INCLUDE <MAIN PROGRAM>
INCLUDE .S$DBMS.SNDMSG
INCLUDE .S$DBMS.COBINT
[INCLUDE <USER SUBROUTINES>]
END
```

Figure 7-7. Link Control File for COBOL and DBMS

7.5 COMPILING AND LINKING PASCAL

After writing the Pascal application program, compile and link the program. See the *Model 990 Computer DNOS TI Pascal Programmer's Guide* for detailed instructions on using the Pascal compiler. See the *Model 990 Computer DNOS Link Editor Reference Manual* for detailed instructions on link editing. Before linking the program, have available a program file in which to place the output of the Link Editor.

The link control file shown in Figure 7-8 is created using the Text Editor. The necessary control file is then input to the Link Editor. All entries are required. Angle brackets (< >) indicate user-supplied information.

The TASK NAME of the TASK command supplies the saved name for the program file. The MAIN PROGRAM is the application name.

For additional information on Pascal, see Appendix B.

```
NOSYMT
FORMAT IMAGE,REPLACE
LIBRARY S$TIP.OBJ
PROC DBINFACE
DUMMY
INCLUDE .S$DBMS.DBINFACE
TASK <TASK NAME>
INCLUDE S$TIP.OBJ.MAIN
INCLUDE <MAIN PROGRAM>
INCLUDE .S$DBMS.SNDMSG
INCLUDE .S$DBMS.FRGMY
INCLUDE <USER SUBROUTINES>
END
```

Figure 7-8. Link Control File for Pascal and DBMS

7.6 COMPILING AND LINKING FORTRAN

After completing a FORTRAN application program, you must compile and link the program. The *Model 990 Computer FORTRAN DNOS Programmer's Reference Manual* contains detailed instructions for using the FORTRAN compiler. The *Model 990 Computer Link Editor Reference Manual* contains instructions for link editing. Before linking the program, have available a program file in which to place the output of the Link Editor.

Since the FORTRAN run time cannot be linked with the DBMS interface, only the DBMS interface module can be in procedure segment one (P1). Figure 7-9 illustrates an example of how to link a FORTRAN task with DBMS-990.

The link control file shown in Figure 7-9 is created using the Text Editor. The desired control file is then input to the Link Editor. All entries are required. Angle brackets (< >) indicate user-supplied information.

The TASK NAME of the TASK command supplies the saved name for the program file. The MAIN PROGRAM is the application program name.

For additional information on FORTRAN, see Appendix B.

```
NOSYMT
FORMAT IMAGE,REPLACE
LIBRARY.FORT78.OSLOBJ
LIBRARY.FORT78.STLOBJ
TASK <TASK NAME>
INCLUDE <MAIN PROGRAM>
INCLUDE .S$DBMS.SNDMSG
INCLUDE <USER SUBROUTINES>
END
```

Figure 7-9. Link Control File for FORTRAN and DBMS

7.7 PROGRAM TESTING WITH DBMS-990

Program testing consists of three major steps: start up, execution, and termination. The following paragraphs discuss these topics.

7.7.1 Start Up

Before executing the program, you must start DBMS-990 (provided it has not been started already). Use the command SDBMS to activate DBMS-990. (See the *Model 990 Computer DNOS Data Base Administrator User's Guide*.)

7.7.2 Execution

Once DBMS-990 is running, execution of the application program can proceed after it is compiled and link edited. Two commands are available for COBOL execution: Execute COBOL Task (XCT) and Execute COBOL Task Foreground (XCTF). The COBOL debugger is also available to help find problems. During testing, use small files if possible. However, use any amount of data necessary for adequately testing the program. For FORTRAN programs, the Execute FORTRAN Task (XFT) or Execute FORTRAN Task Foreground (XFTF) commands are available. For Pascal programs use the Execute Pascal Task (XPT) command. To select background or foreground mode, respond B or F to the MODE prompt.

7.7.3 Termination

Once you have completed the execution phase of testing, you can terminate DBMS-990 by using the EDBMS command. This protects the system from being damaged by a system or hardware crash.

7.8 SUMMARY OF DBMS-990 OPERATION

The following steps are required to develop and run an application on DBMS-990:

1. Design data structures.
2. Partition the structures into DBMS files.
3. Run the DDL compiler to create files. Use the DDL command.
4. Assign security to files, lines, groups, and fields. This is optional (applies only when security is installed in the system).
5. Code the application program by using the DML.
6. Compile and link edit the application program.
7. If necessary, start DBMS-990 by using the SDBMS command. The SDBMS command automatically opens the security and alias files and opens the log when these features are installed in the system.
8. Assign the required files using the ADBF command.
9. Execute the NADB utility and/or test the application program.
10. If desired, use the CLLOG command to terminate the log and close the log file, and the RDBF command to release the assigned files.
11. If desired, end DBMS-990 with the EDBMS command. The EDBMS command automatically releases any assigned files, terminates the security and alias features, and closes the backup log file.

Appendix A

DBMS Exception Reporting

A.1 INTRODUCTION

DBMS-990 errors consist of the following:

- DML errors
- Utility errors
- DBMS errors

The status code of the control block parameter reports DML errors. Utility errors are those that DBMS-990 utilities encounter.

A.2 DML ERRORS

Check the status parameter of the DML call control block after each call. If the status parameter contains anything other than asterisks, an error condition has occurred.

Table A-1 lists and explains the error codes that appear in the status area; Table A-2 lists the utility error codes and explanations.

Table A-1. Error Codes

Code	Type of Error	Probable Cause
AC	Access error	An attempt has been made to access a file using an improper access type (i.e., an update attempt to a file opened with ROEX access), to close a file that has not been opened, or to open a file with an undefined access type.
AE	Address error	The user has supplied an invalid address in loc1 or loc2. This error is possible on all functions if the user has accidentally altered loc1 or loc2. However, it is most likely to occur on a read forward (RF) function. Check the program logic for possible modification to loc1 or loc2.

Table A-1. Error Codes (Continued)

Code	Type of Error	Probable Cause
AS	Add error	The user is attempting to execute an add command on a line type 01 with loc1 not set to asterisks.
BF	Bad file	The data base file has a bad internal pointer or address. This can result if a system failure occurred while the file was being modified and the file was not recovered. Refer to the *Data Base Administrator's Guide* for recovery procedures.
BP	Bad pathname	The pathname supplied for the log file access name was too long.
DA	Delete asterisks	A delete record (DR) function has been specified and loc1 does not contain asterisks.
DB	Data base error	The data base is not running.
DF	Duplicate file	An attempt has been made to assign a file ID that has already been assigned.
DL	Deadlock	The transaction has been rolled back.
DU	Duplicate	The user is attempting to add multiple type 01 lines to the same data record.
FA	Find asterisks	The asterisks at the end of the line list were not found. This can result if the asterisks do not start on a word boundary or if they are not the correct length.
FB	File buffers	No file buffers exist to open the file. If the error occurred during an ADBF command, the size specified in response to the prompt MAXIMUM ASSIGNED FILES in the SDBMS command was not large enough. If the error occurred during an OF function, the size specified in response to MAXIMUM OPEN FILES was not large enough. Wait until a file is closed, or restart DBMS-990 and specify a larger number of file buffers.
FE	Field error	The user has specified an invalid field or group ID(s) in the parameter list. DBMS-990 cannot find the field or group ID. Verify the spelling in the file definition and calling program. Verify that the correct line type is specified.

Table A-1. Error Codes (Continued)

Code	Type of Error	Probable Cause
FH	Full hold buffer	The internal buffer that DBMS-990 uses to register the held lines is full. This results when too many tasks are holding lines; it can also occur when a number of tasks terminate without releasing held lines, thus filling the table.
FL	Full	The area reserved for data lines is full. This error can occur during an add after (AA) or add before (AB) function. Delete any unnecessary records or create a larger file and copy the data to the new file by using CPYFIL/RLDFIL.
FN	Function error	An invalid function code was specified. The function passed to DBMS-990 in the control block is not defined.
FS	File reset	A UDBF was performed on a file that had previously been unlocked.
FU	File in use	The current file is in use and not available at this time.
GF	Good file	A UDBF was performed on a file that was not locked.
HL	Hold line error	A hold line (HL) function has been attempted with loc1 set to asterisks.
IE	Invalid entry ID	The key ID specified is not the primary or secondary key. For an add or delete function, the key ID must be the primary key.
IG	Invalid group	The group ID specified for a query group (QG) function is not a group.
II	Invalid item	The user is not authorized to perform the function against a data item specified in the call.
IK	Invalid key	Either the data line to which the loc1 points does not contain the same value as the key value given in the control block, or an attempt has been made to execute a read on a secondary key that does not exist in the line specified.
IL	Invalid line	The specified line type does not contain the specified field.

Table A-1. Error Codes (Continued)

Code	Type of Error	Probable Cause
IO	I/O error	The operating system encounters an I/O error that occurred during a read or write to the disk.
IT	Invalid transaction	The transactions are not properly bracketed by TS, TC and/or TR.
KF	Key area full	Key area for specified key ID is full for an add after (AA) or add before (AB) function.
KU	Key update error	An attempt is being made to alter the value of a primary or secondary key with the write (WT) function. To alter a key value, delete the line and then reenter it with the new value included.
LA	Line asterisk	A multiple line type specification has been passed to DBMS-990 but no asterisk was found for a write (WT) or delete (DL) function.
LB	Lock tables full	The lock tables are full due to keys that are too large.
LE	Line error	DBMS-990 has received an invalid line list. The "LINE=" syntax cannot be located. Thus the field or group IDs cannot be found.
LF	Log full	The log file is full. Use CLLOG to close the file, define a new file, and then use OPLOG to open the new file.
LL	Lock line error	An attempt was made to lock a line with LOC1 = *.
LO	Log error	A log input/output error has occurred.
L1	Line = 01 error	An attempt is being made to add a line type other than line 01 before a line type 01 has been added. Check the program logic. A line type 01 must exist before any other line type can be added to the record if a LINE = 01 is specified in the DDL.

Table A-1. Error Codes (Continued)

Code	Type of Error	Probable Cause
NB	No buffer	Not enough buffers are available to facilitate the required operation. This is a temporary condition. The size specified in response to the prompt MAXIMUM BUFFERS in the SDBMS command was not large enough. Either wait until a buffer is free, or stop DBMS-990, increase the maximum number of buffers, and then restart DBMS-990.
NF	No file	DBMS-990 cannot find the file ID specified in the file command. The file ID may be misspelled, or it may have been released. Verify that the file ID is assigned and is spelled correctly.
NH	No hold	An attempt is being made to delete a line that has not been held. Prior to deleting a line, the read with hold option must have been specified. The only line available for the delete (DL) is the last one read; with the hold option, the task can only hold one line at a time.
NK	No key found	DBMS-990 cannot find the primary or secondary key value for a read forward (RF) or a read backward (RB) function. Check the program logic and input data.
NL	No logging	DBMS-990 has back-up logging installed, but a log file has not been opened.
OA	Open assign LUNO error	An operating system error occurred while the user was trying to assign a LUNO to the file. Verify that the file exists.
OE	Open error	An operating system error occurred while trying to open the file. Note that DBMS-990 was successful in assigning a LUNO to the file but could not open it, implying that the file is already in use, or that the file was not created by the DDL translator.
OL	Open log error	The log must be either a magnetic tape, cassette, or sequential file. The type found is none of these, or the logical record length is too small.
ON	Open name	The file ID specified does not match the file ID internally stored in the file of the pathname specified.

Table A-1. Error Codes (Continued)

Code	Type of Error	Probable Cause
RL	Record Length	The file being assigned has a record length that is not a valid page size. This file was not created by the DDL translator.
R1	ROEX access error	An attempt has been made to open a file with SHRD or EXCL access while the file is already open with ROEX access, or the same task is trying to execute multiple opens with ROEX access.
SV	Security violation	The user has entered an invalid password and is not authorized to use a data item specified in the call.
PF	Preimage buffer full	The value given for the MAX LINE IMAGES at DBGEN has been exceeded. Reduce the size of the transaction.
S1	SHRD access error	An attempt has been made to open a file with EXCL or ROEX access while the file is already open with SHRD access, or the same task has already opened the file with SHRD access.
UF	Undefined field	The field name in the line list is not defined in the DDL.
UL	Undefined line	The DDL does not define the line type specified for this file.
UT	Undefined key	An attempt has been made to access a data base file with a key type that was not included in DBMS-990 during DBGEN.
WF	Wrong file	All or part of a line that was encoded by a previous call to DBMS-990 has been used in a subsequent call, but the file ID was changed and the new file specified does not contain one of the fields.
XX	Call error	The DML call parameter list is too large for the interface buffer. Possible causes include the following: the call has too many parameters; the wrong parameter list has been sent; or the buffer size allocated during DBGEN is not large enough for the parameter list.

Table A-1. Error Codes (Continued)

Code	Type of Error	Probable Cause
X1	EXCL error access	An attempt has been made to open a file with EXCL, SHRD, or ROEX access when the file is already open with EXCL access.
01	Line = 01 error	An attempt is being made to delete a line type 01 when other line types for the key still exist. A line type 01 cannot be deleted until all other line types for the key have been deleted.

Table A-2. Utility Error Codes

Error Code	Meaning
ABORT	RECOVR forced to abort (entered Q).
BADDAT	A log record has an invalid date and time stamp. If the log file is being used to recover from an operating system crash, BADDAT might signal the end of a log file that has no EOF, instead of signaling an error condition.
BADFIL	The DB FILE ID specified does not match the file ID of the DB FILE PATHNAME specified.
BADKEY	Primary key ID must be the same for the copy file and the DBMS file.
BADLOG	Log file pathname cannot be opened or contains inconsistent data.
BADPSW	Invalid password entry.
BADSFL	The specified copy file cannot be opened or is not of the proper type.
ERROR	A command function name is not valid.
FERROR	A disk I/O operation exceeded the bounds of the file, and all range checks passed. (An operating system error, indicated by U SVC-0331, occurred.)
FILNTF	The specified pathname does not exist. (An operating system error, either U SVC-0304 or U SVC-0315, occurred.)
FNAME	The file ID specified for RLDFIL does not match the copy file.
INVTYP	Invalid type or file change in RLDFIL.

Table A-2. Utility Error Codes (Continued)

Error Code	Meaning
LINTYP	The number of line types that CPYFIL found exceeds the maximum allowed.
LOGI/O	I/O error typing to read log file.
NO OUT	The output listing file cannot be opened.
WRPROT	The utility cannot write to the file. (An operating system error, U SVC-0214, occurred; the disk drive is write protected.)

In addition to the error codes listed in Table A-2, DBMS utilities return the following six character error codes:

CHAR	Contents
1 and 2	DML function code
3 and 4	DML status
5 and 6	Asterisks

A.3 DBMS Error Messages and Codes

DBMS error messages are in the following form:

nnnn <message>

Table A-3 lists the DBMS error messages and explanations. Table A-4 shows the internal error codes and the corresponding message numbers. This is useful on systems that do not contain message files. Use the internal code to find the message number. Then, look up the message number in Table A-3 to find the explanation.

Table A-3. DBMS Error Messages

U	DBMS-0001	ERROR ON OPEN OR CLOSE DBMS FILE, STATUS = ?1 Explanation: A file access error status was returned. This is a result of some other user having exclusive (X1 status) or read-only exclusive (R1 status) access to the file. User Action: Retry when file exclusive or read-only exclusive access privileges are released from the file.
U	DBMS-0002	FIELDS ARE ON DIFFERENT LINES, BAD FIELD = ?1 Explanation: All the output fields must be from the same line. The field identified as the bad field is contained in a different data line from the previous field(s). User Action: Retry excluding the bad field ID.
U	DBMS-0003	ILLEGAL FUNCTION "?1", MUST BE "RS","RF", OR "RB" Explanation: The function entered is not a legal function for the PQUERY utility. User Action: Retry using a valid function, must be 'RS', 'CF', or 'RB'.
U	DBMS-0004	INVALID KEY VALUE Explanation: The value entered for the primary key field does not exist. User Action: Retry with a key value that exists.
U	DBMS-0005	NO FIELD OR GROUP ID SPECIFIED Explanation: No field or group names were entered for the FIELD IDS prompt. User Action: Retry using valid field(s) and/or group(s) ID for prompt.
U	DBMS-0006	STATUS EXCEPTION FROM DBMS, STATUS = ?1 Explanation: The data base manager returned the error status defined in the return message when PQUERY tried to process the request.

Table A-3. DBMS Error Messages (Continued)

		User Action: Consult the error code table to determine the reason for the error status.
U	DBMS-0007	UNABLE TO OPEN FILE, DBMS STATUS = ?1 Explanation: PQUERY could not open the file with shared access. Another task has the file open with either exclusive or read-only exclusive. User Action: Retry operation when PQUERY can get access to the file.
U	DBMS-0008	UNABLE TO OPEN LISTING FILE Explanation: PQUERY utility could not open the listing file requested. User Action: Probably caused by an invalid pathname, does not exist or is a directory name. Another possibility is there is not enough disk space for the listing file.
U	DBMS-0009	UNDEFINED FIELD NAME "?1" Explanation: The field name given in the message is not defined for the file ID entered. User Action: Enter correct field name and retry.
U	DBMS-0010	UNABLE TO OPEN THE LISTING FILE, SVC ERROR ?1 Explanation: The data base utility could not open the requested listing file. The error could be caused by an invalid pathname, a full directory, or a full disk space. User Action: Refer to the SVC error code for the exact cause of the error and respond accordingly.
U	DBMS-0011	DBMS UTILITY ERROR: ?1 Explanation: The DBMS-990 utility detected an error during processing. User Action: See the table containing the utility error codes for further explanation.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0012	CANNOT CLOSE SECURITY FILE Explanation: An error was received from closing the security file. User Action: Possible hardware error.
U	DBMS-0013	CANNOT GET TCA FILE Explanation: An error occurred when trying to access the TCA region. User Action: Possible hardware error.
U	DBMS-0014	CANNOT GET OPEN SECURITY FILE Explanation: An error was received when opening the security file. User Action: Possible hardware error. Another possibility is that another task has the file open with access privileges that conflict.
U	DBMS-0015	ERROR IN SYNONYM ASSIGNMENT Explanation: An error was received when trying to assign a synonym User Action: Delete some synonyms and retry the operation.
U	DBMS-0016	ERROR WHEN ASSIGNING LUNO TO SECURITY FILE Explanation: An error was received when trying to assign a LUNO to the security file. User Action: Verify the security file exists.
U	DBMS-0017	ERROR WHEN READING SECURITY FILE Explanation: An error was received when trying to read the security file. User Action: Possible hardware error.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0018	ERROR WHEN RELEASING LUNO FOR SECURITY FILE Explanation: An error was received when trying to release a LUNO assigned to the security file. User Action: Possible hardware error.
U	DBMS-0019	INVALID MASTER PASSWORD Explanation: The password entered is not the correct master password. User Action: Retry with the correct master password.
U	DBMS-0020	INVALID DATA FORMAT CONVERSION FOR RLDFIL Explanation: RLDFIL does not support conversion of the data for format specified. User Action: Do not use RLDFIL for changing the format for data types that are not supported by RLDFIL.
U	DBMS-0021	DBMS FILE IS NOT EMPTY Explanation: The file into which the data is to be loaded is not empty. The reload does not occur. User Action: Format the file with the DDL translator before trying the reload.
U	DBMS-0022	INVALID FIELD CONVERSION IN RLDFIL Explanation: RLDFIL encountered a field conversion request that is invalid. User Action: Refer to the documentation for RLDFIL for the valid type changes. Compare the DDL for the file that CPYFIL copied to a sequential file against the DDL for the data base file that is being reloaded. Change the invalid conversion to be valid.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0023	FIELD DROPPED IN NEW DDL — ?1 Explanation: The field was found in the old DDL but is not in the new DDL. User Action: Verify that the field should not be in the new DDL.
U	DBMS-0024	?1 LINE TYPE MISSING IN NEW DDL Explanation: Line identifier is missing in new DDL. This message occurs if the old DDL contains a line type not in the new DDL. User Action: Verify that the line type should not be in the new DDL.
U	DBMS-0025	POSSIBLE LOSS OF SIGNIFICANCE IN FIELD — ?1 Explanation: This message results if the field in the old DDL has more significant digits than the field declared in the new DDL. User Action: Verify that loss of significance is valid.
U	DBMS-0026	TYPE TRANSFER FOR RPG DATA NOT IMPLEMENTED Explanation: RPG data types are not supported by the RLDFIL utility. User Action: None.
U	DBMS-0027	UNABLE TO OPEN THE INPUT FILE, SVC ERROR ?1 Explanation: The DDL translator could not open the requested input file. The error could be caused by an invalid pathname or by specifying a file that does not exist. User Action: Refer to the SVC error code for the exact cause of the error and respond accordingly.
U	DBMS-0028	CANNOT OBTAIN TCA FILE, SVC ERROR ?1 Explanation: The data base utility could not obtain the TCA file. User Action: Refer to the SVC error code for the exact cause of the error and respond accordingly.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0029	ERROR ON READ OF INPUT FILE, SVC ERROR ?1 Explanation: The DDL translator has detected an error on a read from the input file. User Action: Refer to the SVC error code for the exact cause of the error and respond accordingly.
U	DBMS-0030	DATABASE MANAGER ALREADY RUNNING Explanation: The data base manager is already running. User Action: An EDBMS command must be issued prior to reissuing the SDBMS command.
U	DBMS-0031	INVALID OR MISSING PARAMETER Explanation: A parameter required by the data base is either missing or invalid. User Action: Verify that the parameters used in the bid are correct.
U	DBMS-0032	UNABLE TO BID DATABASE MANAGER. SVC ERROR: 2B?1 Explanation: An SVC error occurred during the bid task. User Action: Refer to the SVC error code for the correct action.
U	DBMS-0033	DBMS UTILITY ERROR: ?1 Explanation: An error was detected during the execution of the DBMS-990 utility. The six-character code defines the type of error that occurred. User Action: Look up the six-character code in the Utility Error Codes table. Determine the cause of the error. Retry the operation.
U	DBMS-0034	INVALID FUNCTION Explanation: The SECFUNC task was bid with an invalid function code. User Action: Check to see that the SCI procedure has not been altered.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0035	INVALID PATHNAME Explanation: An invalid pathname was entered. User Action: Retry using a valid pathname.
U	DBMS-0036	INVALID STATUS CODE: ?1 Explanation: An error status code was returned by the data base manager. User Action: Refer to the DBMS-990 status codes for further information.
U	DBMS-0037	INVALID FILE Explanation: The file ID specified is longer than four characters. User Action: Retry using a maximum of four characters for a file ID.
U	DBMS-0038	ERROR ON WRITE TO LISTING FILE, SVC ERROR ?1 Explanation: The DDL translator has detected an error on a write to the listing file. User Action: Refer to the SVC error code for the exact cause of the error and respond accordingly.
U	DBMS-0039	ERRORS DETECTED DURING DDL TRANSLATION Explanation: The DDL translator has detected errors during translation. User Action: Check the listing file for the specific location of syntax errors and for any semantic errors. Make the appropriate revisions and resubmit the DDL.
U	DBMS-0040	STATUS CODE: ?1 Explanation: An invalid status was returned by the data base manager. User Action: Refer to DBMS-990 status codes for further information.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0041	CANNOT GET ACCESS Explanation: Another task has exclusive (or read-only exclusive) access to the file specified. User Action: Retry the operation when the file has been released from exclusive access.
U	DBMS-0042	CANNOT GET PARAMETER Explanation: An error occurred when trying to read bid parameter. User Action: The task was bid with incorrect number of parameters. Possible hardware error.
U	DBMS-0043	CANNOT OPEN FILES Explanation: The maximum number of open files has been reached or another task has the security or alias files open with access privileges that conflict. User Action: Restart the data base manager (SDBMS) with a larger maximum number of open files. If another task has the files open then retry later.
U	DBMS-0044	CODE CONFLICT Explanation: The entry to be added is not a subset of the authorization of the higher-level entry, or the item to be added does not have delete authority but the associated line does have delete authority. User Action: Resolve the conflict and retry.
U	DBMS-0045	DATABASE DOWN Explanation: The data base is not up; it has not been started. User Action: Start the data base with SDBMS command.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0046	DUPLICATE FILE Explanation: Tried to add a file that has already been assigned to the password. User Action: Verify access authorizations for the file assigned to the password.
U	DBMS-0047	DUPLICATE ITEM Explanation: Tried to add an item that has already been assigned to the password. User Action: Verify access authorizations for the item assigned to the password.
U	DBMS-0048	DUPLICATE LINE Explanation: Tried to add a line that has already been assigned to the password. User Action: Verify access authorization for the line assigned to the password.
U	DBMS-0049	DUPLICATE PASSWORD Explanation: Tried to add a password that has already been assigned. User Action: Verify password exists.
U	DBMS-0050	ENTRY AREA FULL Explanation: Attempted to add password/alias entry when the password/alias entry area is full. User Action: Expand the disk data area. Users should consult the DBA. The DBA should copy the security/alias file, depending on which file is full, using the CPYFIL utility. The security file ($SC1)/alias file ($AL1) is located in the SC1/AL1 node of the data base library directory. Redo the DBINS procedure specifying new security/alias file. Increase the number of security/alias entries and reload the copied data.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0051	FILE NOT FOUND Explanation: The file ID specified does not have authorization. The file ID specified for deletion does not exist for the password. User Action: Must first give authorization to the file. Retry deletion with the valid file ID for the password.
U	DBMS-0052	INVALID CODE Explanation: Error when entering the authorizations for the entry. User Action: Reenter the command.
U	DBMS-0053	INVALID LISTING ACCESS NAME Explanation: The listing access name is not valid. User Action: Retry using a valid listing access name. Check available space in directory and/or disk volume.
U	DBMS-0054	INVALID FUNCTION Explanation: The task was bid with a function that is not defined. User Action: Check to see that the SCI procedure has not been altered.
U	DBMS-0055	INVALID ITEM Explanation: Item ID is longer than four characters. User Action: Retry using four characters or less for the item name.
U	DBMS-0056	INVALID LINE Explanation: The line ID specified is longer than two characters. User Action: Retry using two characters for the line ID.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0057	INVALID MASTER Explanation: Master password is longer than four characters. User Action: Retry using four characters or less for master password.
U	DBMS-0058	INVALID MAX VALUE Explanation: When creating data base security, the MAX PASSWORDS or MAX ENTRIES prompt value was greater than 999999. User Action: Redo the DBINS process specifying values less than 999999 for the MAX PASSWORDS and/or MAX ENTRIES prompt.
U	DBMS-0059	INVALID PASSWORD Explanation: The password specified does not have access authorization or has not been assigned. Also, the password specified is longer than four characters. User Action: Verify the validity and/or authorization of the password.
U	DBMS-0060	INVALID TYPE Explanation: Type is not FILE, LINE or ITEM. User Action: Retry using a valid type.
U	DBMS-0061	ITEM NOT FOUND Explanation: The item ID specified does not exist. User Action: Retry with different item ID.
U	DBMS-0062	LINE NOT FOUND Explanation: The line ID specified does not have authorization. The line ID specified for deletion does not exist for password. User Action: Must first give the authorization to the line. Retry with a valid line ID for the password.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0063	NO BUFFERS Explanation: No open file buffers. Other tasks are already using the buffers. User Action: Wait until buffers are available.
U	DBMS-0064	PASSWORD AREA FULL Explanation: The maximum number of passwords specified in DBINS has been exceeded. User Action: Expand the disk data area. Users should consult the DBA. The DBA should copy the security file using the CPYFIL utility. The security file ($SC1) is located in the SC1 node of the data base library directory. Redo the DBINS procedure specifying new security file. Increase the number of passwords and reload the copied data.
U	DBMS-0065	PASSWORD NOT FOUND Explanation: The password used is not a valid password. User Action: Verify the correct password was used.
U	DBMS-0066	DB FILE NOT ASSIGNED Explanation: An attempt was made to access the security or alias files without the files being assigned. User Action: Stop the data base using the EDBMS command. Restart the data base using the SDBMS command.
U	DBMS-0067	ALTERNATE COLLATING SEQUENCE SPECIFIED AND S1 ROUTINES WERE NOT INCLUDED Explanation: A pathname for an alternate collating sequence was specified as a bid parameter to the data base manager that was generated without sequential keys. Alternate collating sequences are only valid for the ordering of sequential keys; thus a data base manager generated without sequential keys cannot use the alternate collating sequence file.

Table A-3. DBMS Error Messages (Continued)

		User Action: Remove the alternate collating sequence pathname from the bid parameter.
U	DBMS-0068	UNABLE TO ASSIGN LUNO TO ALTERNATE COLLATING SEQUENCE FILE Explanation: The pathname defined for the alternate collating sequence file is invalid. User Action: Change the pathname used for the alternate collating sequence to be valid or null.
U	DBMS-0069	UNABLE TO OPEN ALTERNATE COLLATING SEQUENCE FILE Explanation: An error occurred when the data base manager attempted to open the alternate collating sequence file. User Action: Verify the validity of the pathname in the bid parameter for the alternate collating sequence file.
U	DBMS-0070	UNABLE TO READ ALTERNATE COLLATING SEQUENCE FILE Explanation: An error occurred when the data base manager attempted to read the alternate collating sequence file. User Action: Verify the validity of the pathname in the bid parameter for the alternate collating sequence file.
U	DBMS-0071	UNABLE TO OPEN THE DATA BASE FILE, SVC ERROR ?1 Explanation: The DDL translator could not open the requested data base file. The error could be caused by an invalid pathname, a full directory, or a full disk space. User Action: Refer to the SVC code for the exact cause of the error and respond accordingly.
U	DBMS-0072	** AN EQUAL SIGN (' = ') WAS EXPECTED ** Explanation: The DDL translator expected an equal sign in the location indicated by the up arrow.

Table A-3. DBMS Error Messages (Continued)

		User Action: Correct the syntax and resubmit the DDL.
U	DBMS-0073	** INVALID LINE/VOLUME VALUE ** Explanation: A zero (0) has been entered for the value of a line or volume. User Action: Change the line or volume to a nonzero value and resubmit the DDL.
U	DBMS-0074	** INVALID CHARACTERS IN ID ** Explanation: A four character ID has been entered that contains an invalid character. The invalid character has been flagged with an up arrow. An ID may contain only alphanumeric characters, numeric characters, dollar signs ($), and blanks. An ID must start with a dollar sign or an alphanumeric character and may not contain embedded blanks. User Action: Correct the invalid character and resubmit the DDL.
U	DBMS-0075	** A COMMA (',') WAS EXPECTED ** Explanation: The DDL translator expected a comma in the location indicated by the up arrow. User Action: Correct the syntax and resubmit the DDL.
U	DBMS-0076	** KEYWORD EXPECTED ** Explanation: The DDL translator expected a keyword before the location indicated by the up arrow. User Action: Verify the entry is a valid keyword and resubmit the DDL.
U	DBMS-0077	** A NUMERIC VALUE WAS EXPECTED ** Explanation: The DDL translator expected a numeric value before the location indicated by the up arrow. User Action: Verify the entry is a valid numeric value and resubmit the DDL.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0078	** A FORMAT WAS EXPECTED AFTER THIS DATA TYPE ** Explanation: The DDL translator expected a format after the data type specified in a field definition line. User Action: Add a valid format to the data type and resubmit the DDL.
U	DBMS-0079	** INVALID DATA TYPE LENGTH ** Explanation: An invalid data type length has been entered. A CX or RD field must be 8 bytes long, an RS or ID field must be 4 bytes long, and an LG or IS field must be 2 bytes long. An FX field must have a length of 2 bytes with between 0 and 16 bits. A PK, AN, AS, CN, or CS data type must have a length of between 1 and 18 bytes. User Action: Verify the data length is valid and resubmit the DDL.
U	DBMS-0080	** A PERIOD ('.') WAS EXPECTED ** Explanation: The DDL translator expected a period in the location indicated by the up arrow. User Action: Correct the syntax and resubmit the DDL.
U	DBMS-0081	** MAXIMUM LINE/VOLUME COUNT EXCEEDED — ?1 ** Explanation: The number specified for the line/volume count is greater than that allowed by the DDL translator. User Action: Revise the number and resubmit the DDL.
U	DBMS-0082	CANNOT FORMAT DATABASE FILE WITH PATHNAME SPECIFIED Explanation: The file specified for the new data base file already exists and does not match the characteristics of the file to be created. The preexisting file must be a relative record file with the same physical and logical page size as the file to be created, and both files must have the same number of records.

Table A-3. DBMS Error Messages (Continued)

		User Action: Choose another pathname for the data base file or delete the file that currently resides at the pathname specified.
U	DBMS-0083	ALIAS AREA FULL Explanation: Attempted to add an alias when the alias area was full. User Action: Expand the disk data area. Users should consult the DBA. The DBA should copy the alias file using the CPYFIL utility. The alias file ($AL1) is located in the data base directory in the AL1 node. Redo the DBINS procedure specifying new alias file. Increase the number of alias entries and reload the copied data.
U	DBMS-0084	ALIAS NOT FOUND Explanation: The alias specified does not exist. User Action: Check for spelling error and try again.
U	DBMS-0085	DUPLICATE ENTRY FOR ALIAS Explanation: The specified alias name is already assigned. User Action: Select another alias name.
U	DBMS-0086	ERROR WHEN OPENING LISTING FILE Explanation: An error was received when trying to open the listing file. User Action: Verify the validity of the listing file access name. Check to ensure that the file can be created with access name specified.
U	DBMS-0087	ERROR WHEN WRITING TO LISTING Explanation: An error was received when writing the listing file. User Action: Check to ensure listing file device is ready. If listing access name is a disk then check that space is available for the file.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0088	FILE NOT DEFINED FOR ALIAS Explanation: The specified alias is not assigned to the specified file. User Action: Verify the correct alias and file and retry the operation.
U	DBMS-0089	ILLEGAL ALIAS NAME Explanation: An alias name must begin with an alphabetic character; be 20 characters or fewer in length; and consist of alphanumeric, dollar sign ($), dash (-), or underscore characters only. User Action: Modify the alias name to fit the description above.
U	DBMS-0090	ILLEGAL FIELD FOR ALIAS Explanation: The specified field does not have access authorization or has not been assigned. User Action: Verify the field ID and retry the operation.
U	DBMS-0091	LINE TYPE NOT DEFINED FOR ALIAS Explanation: The specified alias is not assigned to the specified line type. User Action: Verify the alias and line type and retry the operation.
U	DBMS-0092	** AN '01' LINE TYPE MAY ONLY FOLLOW THE PRIMARY KEY DEFINITION ** Explanation: The DDL translator has encountered an 01 line type after other line types have been defined. If a file contains an 01 line, it must precede all other line type definitions. User Action: Position the 01 definition to precede all others or change the ID of the line; then resubmit the DDL.
U	DBMS-0093	** DUPLICATE FIELD/GROUP NAME ENCOUNTERED — ?1 ** Explanation: The DDL translator has encountered a field or group ID that has been defined twice.

Table A-3. DBMS Error Messages (Continued)

		User Action: Change one of the occurrences of the duplicate ID and resubmit the DDL.
U	DBMS-0094	** INVALID ACCESS DECLARATION ** Explanation: The DDL translator has encountered an invalid access declaration in the optional ACCESS clause. One of the access keywords RANDOM or SEQUENTIAL is required. The /1 designator is optional. User Action: Correct the syntax error indicated by the location of the up arrow and resubmit the DDL.
U	DBMS-0095	** DUPLICATE LINE TYPE ENCOUNTERED ** Explanation: The DDL translator has encountered a line type ID that has been previously defined. User Action: Change the previous occurrence of the ID or the occurrence flagged and resubmit the DDL.
U	DBMS-0096	** INVALID DATA TYPE ** Explanation: The DDL translator has encountered an invalid data type. A list of valid data types is listed in the DBMS Programmer's Guide. User Action: Verify that a valid data type has been entered and resubmit the DDL.
U	DBMS-0097	** SYNTAX ERROR ** Explanation: The DDL translator has encountered a syntax error. Usually, this error is caused by a nonblank character at the end of a line. User Action: Correct the syntax error marked by the location of the up arrow and resubmit the DDL.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0098	** ERROR ON CONVERSION OF ASCII VALUE ** Explanation: The DDL translator could not convert a numeric ASCII string to internal format. The ASCII string probably contains a non-numeric character. User Action: Verify that the value flagged by the up arrow contains only numeric characters and resubmit the DDL.
U	DBMS-0099	** STATEMENT OUT OF ORDER ** Explanation: The DDL translator has encountered a statement that it did not expect. The statement is not valid in its present location. User Action: Relocate the statement in a valid location and resubmit the DDL.
U	DMBS-0100	** MAXIMUM LINE LENGTH EXCEEDED ** Explanation: The data line identified exceeds the maximum length allowable, 512 bytes. The maximum size of a data line is 512 bytes minus the primary key length, minus 10 bytes overhead, minus eight times the number of secondary keys in the line. User Action: Shorten the line and resubmit the DDL.
U	DBMS-0101	ERROR ON CLOSE OF THE DATA BASE FILE, SVC ERROR ?1 Explanation: The DDL translator has detected an error while trying to close the data base file. User Action: Refer to the SVC error code for the exact cause of the error and respond accordingly.
U	DBMS-0102	ERROR ON WRITING TO THE DATA BASE FILE, SVC ERROR ?1 Explanation: The DDL translator has detected an error while trying to write to the data base file. User Action: Refer to the SVC error code for the exact cause of the error and respond accordingly.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0103	END OF TAPE ENCOUNTERED BY CPYFIL Explanation: CPYFIL utility encountered an end of tape mark. User Action: Mount a new reel of tape. Press the "return" key when the tape unit becomes ready.
U	DBMS-0104	** FILE NOT PROCESSED DUE TO SYNTAX ERRORS OR 'DUMY' PATHNAME ** Explanation: No data base file has been created. Either the DDL translator detected syntax errors during parsing or the user specified DUMY in response to the DB FILE PATHNAME indicating no file should be created. User Action: Correct any syntax errors flagged in the listing and enter a valid file pathname in response to the DB FILE PATHNAME prompt.
U	DBMS-0105	END OF TAPE ENCOUNTERED BY RLDFIL Explanation: RLDFIL utility encountered an end of tape mark. User Action: Mount the next reel of tape from the multiple tapes produced by CPYFIL. Press "return" when the tape unit becomes ready.
U	DBMS-0106	END OF TAPE ENCOUNTERED BY RECOVR Explanation: RECOVR utility encountered an end of tape mark. User Action: Mount the next reel of tape from the multiple tapes containing the log file. Press "return" when the tape unit becomes ready.
U	DBMS-0107	?1: ?2 IS NOT A VALID KEYTYPE Explanation: The key type found in the file is not a valid DBMS-990 key type. DBMS-990 supports two key types, sequential (S1) and random (R1). The key type reported is not supported. User Action: None.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0109	NO VALID DATABASE FUNCTIONS NEEDED Explanation: No valid data base functions were entered in response to the FUNCTIONS prompt. User Action: Refer to the DBSTAT section of the DBA User's Manual for a list of legal functions or enter ALL for a report listing the statistics on all the functions.
U	DBMS-0110	** MAXIMUM GROUP LENGTH EXCEEDED ** Explanation: The group flagged by the DDL translator exceeds the maximum length allowable. User Action: Reduce the size of the group and resubmit the DDL.
U	DBMS-0111	** MAXIMUM NUMBER OF SECONDARY KEYS EXCEEDED ** Explanation: More than 13 secondary keys have been defined. User Action: Reduce the number of secondary keys and resubmit the DDL.
U	DBMS-0112	** NEW DATA BASE FILE CREATED ** Explanation: The data base file has been formatted and is ready to be accessed by DBMS-990. User Action: No user action required.
U	DBMS-0113	** NO FIELDS DEFINED FOR THIS LINE/GROUP ** Explanation: The line or group flagged by the DDL translator has no fields defined. User Action: Add a field definition to the line or group and resubmit the DDL.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0114	** NO LINES DEFINED FOR THIS FILE ** Explanation: No lines have been defined to the DDL translator for this file. User Action: Add a line definition to the file and resubmit the DDL.
U	DBMS-0115	ERROR ON WRITE TO LISTING FILE, SVC ERROR ?1 Explanation: The DBSTAT utility has detected an error during a write to the listing file. User Action: Refer to the SVC error code for the exact cause of the error and respond accordingly.
U	DBMS-0116	** KEY LENGTH EXCEEDS MAXIMUM ALLOWABLE ** Explanation: The length of the primary or secondary key flagged by the DDL translator is longer than the maximum allowable, 40 bytes. User Action: Reduce the size of the key field or group and resubmit the DDL.
U	DBMS-0117	** INVALID SECONDARY KEY NAME ** Explanation: The field/group name flagged by the DDL translator is invalid as a secondary key. Either the field/group has not been defined in a line, the field/group is defined as the primary key, or the field/group is already defined as a secondary key. User Action: Verify that the field/group is valid as a secondary key and resubmit the DDL.
U	DBMS-0118	ERROR RETURNED FROM DBMS, CODE ?1 Explanation: DBMS-990 has returned an error code to the DBSTAT utility. User Action: Refer to the DBMS Programmer's Guide for an explanation of the two character error code and respond accordingly.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0122	THE PAGE SIZE HAS TO BE EITHER 256 OR 288 Explanation: The page size for the DB file must be either 256 or 288. Usually, choice is made dependent on sector size of target disk for the DB file. User Action: Retry with valid page size, 256 or 288.
U	DBMS-0123	** DDL ENDS PREMATURELY ** Explanation: The DDL translator has reached an EOF before parsing the END. statement. User Action: Verify that all statements are in their correct order, add the END. statement and resubmit the DDL.
U	DBMS-0124	DATABASE MANAGER UNABLE TO OPEN PRE-IMAGE FILE Explanation: The system files have been damaged. User Action: Perform system generation at this time.
U	DBMS-0126	DATABASE MANAGER CANNOT GET REQUIRED MEMORY Explanation: An attempt was made to start the data base manager (SDBMS) with too many buffers. User Action: Retry the SDBMS command with smaller numbers for the parameters.
U	DBMS-0130	DBMS SUCCESSFULLY STARTED Explanation: The data base manager is running User Action: No user action required.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0131	SYSTEM FAILURE OCCURRED WHILE UPDATING FILE ?1 Explanation: The system crashed in the process of updating the file in question. The physical integrity of this file is now in question. User Action: Perform the Copy/Concatenate (CC) command on the backup for the file pathname and the RECOVR utility.
U	DBMS-0132	UNABLE TO ACCESS FILE ?1 Explanation: For some reason the file in question is no longer available. User Action: Check to see if the necessary volume is installed. If the file is no longer available, perform CDBL.
U	DBMS-0133	PATHNAME OF THE FILE IS ?1 Explanation: Gives the pathname of the affected file. User Action: None.
U	DBMS-0134	INTEGRITY ERROR ENCOUNTERED, PERFORM RECOVR Explanation: The system was interrupted in the process of updating a file. The physical integrity of the file is now in question. User Action: Perform the Copy/Concatenate command on the backup for the file pathname and perform the RECOVR utility.
U	DBMS-0135	UNABLE TO RESTART DBMS, CANNOT REOPEN FILES Explanation: The system is not able to open all the files that were assigned when the data base was last running. User Action: Check to see that the file(s) indicated is accessible, e.g., that the appropriate volume is installed, the file(s) has not been deleted. Perform the CDBL utility if the file is not accessible.

Table A-3. DBMS Error Messages (Continued)

U	DBMS-0136	RESTART FAILED Explanation: The system was unable to restart the data base manager. User Action: None.
U	DBMS-0137	FILES REOPENED, ROLLBACK FAILED Explanation: Either the system log has been damaged or one of the files affected by rollback does not match the form it was in when the data base was last running and an error has occurred during rollback. User Action: Perform CDBL.
U	DBMS-0138	DBMS OPEN ERROR IS ?1 Explanation: Returns error status from DBMS. User Action: Check Appendix A of the DBMS Programmer's Guide for an explanation of the two character code.

Table A-4. Internal Message Codes

Internal Message Code	DBMS Message Number
>0001	0001
>0002	0002
>0003	0003
>0004	0004
>0005	0005
>0006	0006
>0007	0007
>0008	0008
>0009	0009
>000A	0010
>000B	0011
>000C	0012
>000D	0013
>000E	0014
>000F	0015
>0010	0016
>0011	0017
>0012	0018
>0013	0019
>0014	0020
>0015	0021
>0016	0022
>0017	0023
>0018	0024
>0019	0025
>001A	0026
>001B	0027
>001C	0028
>001D	0029
>001E	0030
>001F	0031
>0020	0032
>0021	0033
>0022	0034
>0023	0035
>0024	0036
>0025	0037
>0026	0038
>0028	0040
>0029	0041
>002A	0042
>002B	0043
>002C	0044
>002D	0045
>002E	0046
>002F	0047
>0030	0048
>0031	0049
>0032	0050
>0033	0051

Table A-4. Internal Message Codes (Continued)

Internal Message Code	DBMS Message Number
>0034	0052
>0035	0053
>0036	0054
>0037	0055
>0038	0056
>0039	0057
>003A	0058
>003B	0059
>003C	0060
>003D	0061
>003E	0062
>003F	0063
>0040	0064
>0041	0065
>0042	0066
>0043	0067
>0044	0068
>0045	0069
>0046	0070
>0053	0083
>0054	0084
>0055	0085
>0056	0086
>0057	0087
>0058	0088
>0059	0089
>005A	0090
>005B	0091
>005C	0092
>005D	0093
>005E	0094
>005F	0095
>0061	0097
>0062	0098
>0063	0099
>0065	0101
>0066	0102
>0067	0103
>0068	0104
>0069	0105
>006A	0106
>006B	0107
>006C	0108
>006D	0109
>006E	0110
>006F	0111
>0070	0112
>0071	0113
>0073	0115
>0074	0116

Table A-4. Internal Message Codes (Continued)

Internal Message Code	DBMS Message Number
>0077	0119
>0078	0120
>0079	0121
>007A	0122
>007B	0123
>007E	0126

Appendix B

Example DBMS Programs

B.1 INTRODUCTION

The programs and operating instructions in this appendix illustrate the comparative uses of COBOL, FORTRAN, and Pascal by writing the same program in all three languages. An additional COBOL program illustrates the use of transaction bracketing for transaction-level integrity. These example programs also illustrate the use of secondary keys in DBMS-990. The example program in B.7 illustrates the use of transactions. All programs run under the minimum DBMS-990 system.

The DBMS-990 installation disk contains the source necessary to execute each program. After the installation process is complete, the code is located on the system disk under the library .S$DBMS.TEST in the files listed in Table B-1. In the following paragraphs, DBMS-990 is assumed to be installed, active, and ready for use.

Table B-1. Files Used to Execute Example Programs

File	Paragraph Reference
.S$DBMS.TEST.CUST	B.3.1
.S$DBMS.TEST.ITEM	B.3.2
.S$DBMS.TEST.SOFL	B.3.3
.S$DBMS.TEST.ILDSOFL	B.3, B.4, B.5
.S$DBMS.TEST.ILDCUST	B.3, B.4, B.5
.S$DBMS.TEST.ILDITEM	B.3, B.4, B.5
.S$DBMS.TEST.TRAN	B.3, B.4, B.5
.S$DBMS.TEST.CEXMPL	B.4
.S$DBMS.TEST.PEXMPL	B.6
.S$DBMS.TEST.CNLNK	B.4
.S$DBMS.TEST.FNLNKN	B.5
.S$DBMS.TEST.PNLNK	B.6
.S$DBMS.TEST.CBATCHN	B.4
.S$DBMS.TEST.FBATCHN	B.5
.S$DBMS.TEST.PBATCHN	B.6
.S$DBMS.TEST.CBATCHTR	B.7
.S$DBMS.TEST.LOADFILN	B.3, B.4, B.5, B.6

B.2 THE EXAMPLE PROGRAMS

The COBOL example (CEXMPL), the FORTRAN example (FEXMPN), and the Pascal example (PEXMPL) are small programs. The programs execute the same logic and use the same data base and transaction files to produce the same output. The programs contain logic for security and file-access checking. If your system includes security checking, assign the password TEST to the three data base files SOFL, CUST, and ITEM. The programs will execute properly on systems that do not include security and file-access checking.

Each program extracts data from the sales order file (SOFL) about specific item numbers. These item numbers are obtained from a sequential input transaction file. Data retrieved from the data base files is sent to a user-specified sequential file. This file may be printed or displayed after the program terminates.

The first column of output is the item number obtained from the input transaction file. This number is checked against the file ITEM to verify that the item exists. It is also checked against the SOFL file by using the ITEM secondary key to verify that the item has been sold. The second and third columns of output are the item description (DESC) and unit price (UPRC) obtained from the file ITEM. The fourth, fifth, and sixth columns, obtained from the file SOFL, are the quantity on order (QUAN), the sales order number (SONM), and the ship-to-customer number (SHIP), respectively. The ship-to-customer number is used to obtain the customer name (NAME) from the customer file (CUST) and is shown in the seventh column of the output.

All three programs are documented with comments contained in the source. Appropriate error messages appear in the output file when errors occur. If more than one of the programs is being used, do not assign all of the output file synonyms to the same output file, since conflicts in usage might occur.

B.3 DBMS-990 FILES

The example programs use three data base files, the customer file CUST, the item file ITEM, and the sales order file SOFL. Figure B-1 and Figure B-2 show the relationship between these files.

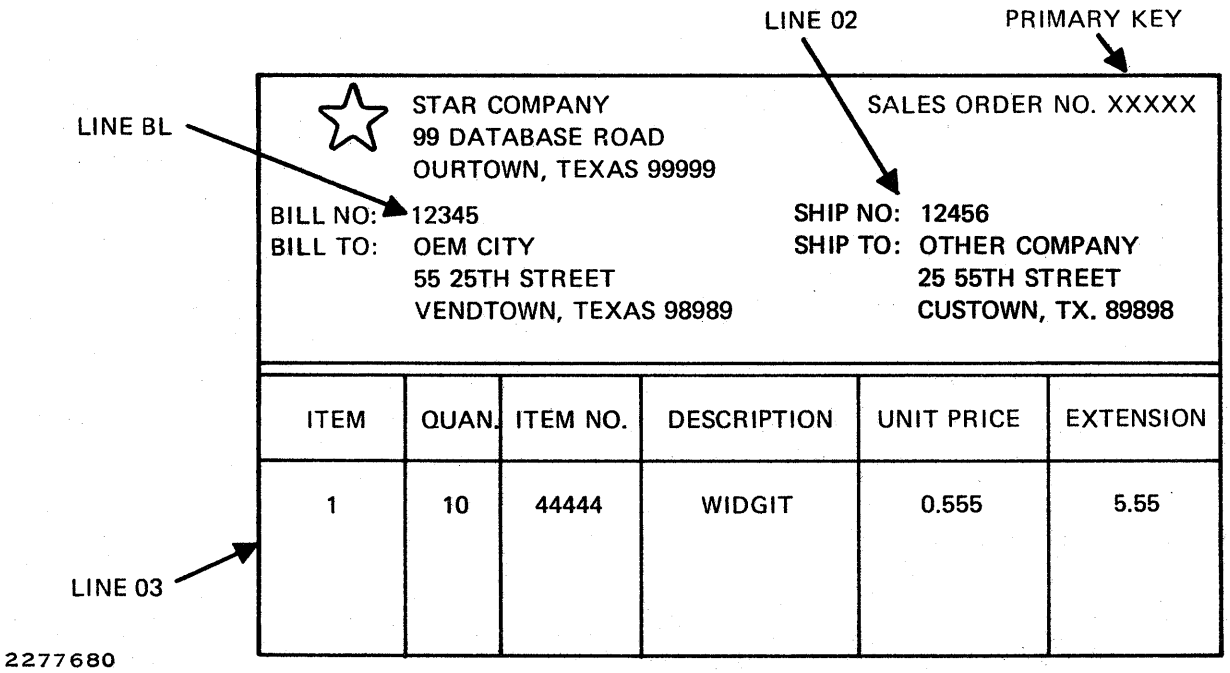


Figure B-1. Relationship of DBMS-990 File to Sales Order Document

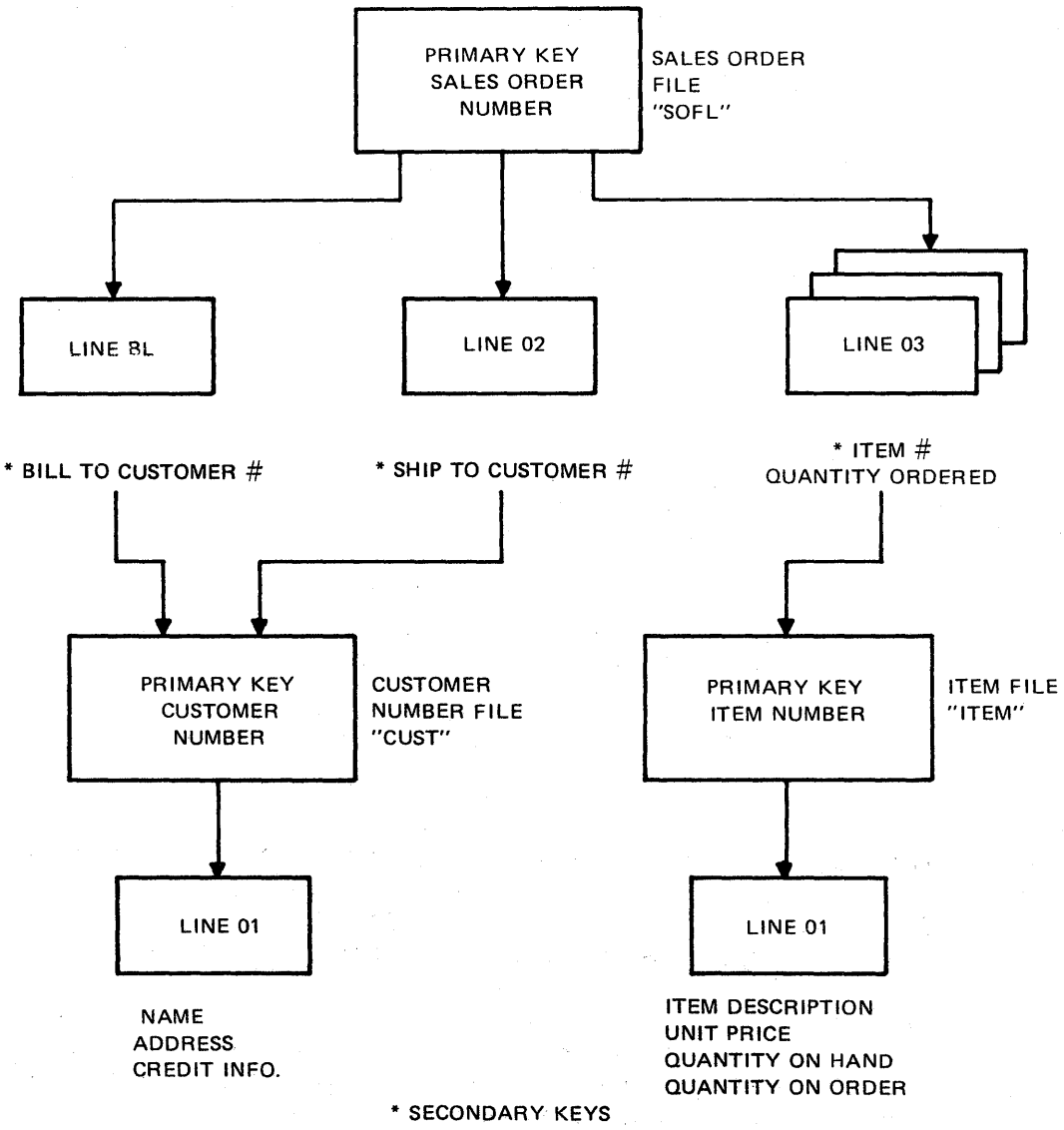


Figure B-2. Logical Relationship of Files

B.3.1 Customer File (CUST)

The customer file (CUST) contains the customer number, name, address, and credit information (CRED) for each customer. The customer number must be unique, since it is the primary key of the file. The customer file is referenced symbolically from the sales order file (SOFL) by using the customer number. Figure B-3 shows the DDL listing for the customer file. Note that the maximum number of customers allowed is 50. This is because the total number of lines cannot exceed 50, and only one line type is defined.

```
DBMS-990      <L. V. R. >        DDL TRANSLATOR     MM/DD/YY   HH: MM: SS

FILE=CUST, LINES=50
ID=CUSN=CH/5, VOL=50, ACCESS=RANDOM/1
*
LINE=01
  FIELD=NAME=CH/20
  GROUP=ADDR
    FIELD=STRT=CH/20
    FIELD=CITY=CH/20
    FIELD=STAT=CH/2
    FIELD=ZIPC=CH/5
    ENDG
  FIELD=CRED=CH/2
  ENDL
END.

      TOTAL PAGES REQUIRED - 24
       LINE LENGTH (BYTES) - 84
   TOTAL DESCRIPTION PAGES - 1
           TOTAL KEY PAGES - 5

 LINE 01 -- BASE = 15 , DATA = 69 , LINKAGE = 0 , TOTAL = 84

00112  **  NEW DATA BASE FILE CREATED **
```

Figure B-3. DDL Listing for the Customer File

B.3.2 Item File (ITEM)

The item file (ITEM) contains information about each part in the inventory. The primary key for this file is the item number. Consequently, each item number must be unique. This file contains the description (DESC), unit price (UPRC), quantity on order (QTYO), and quantity on hand (QTYH). You can enter a maximum 50 items in the item file. Figure B-4 shows the DDL listing for the file.

B.3.3 Sales Order File (SOFL)

The sales order file (SOFL) symbolically links the ITEM and CUST files to form the sales order data base. This file uses three line types. Line BL uses a pointer to designate which customer to bill for the sales order. The field BILL contains the customer number of a customer for whom information is maintained in CUST. Line 02 uses a pointer to designate the customer to which the order is to be shipped. The SHIP field contains the customer number, and the related customer information is maintained in CUST.

Line 03 contains the detail items for the sales order. Only one line BL and one line 02 are entered in SOFL, but you can enter more than one line 03. Each line 03 contains the item number (ITEM) and the quantity ordered (QUAN). The item number must be a valid item in the item file. The item file also contains any additional information concerning that particular item.

```
 DBMS-990     <L.V.R>            DDL TRANSLATOR      MM/DD/YY   HH:MM:SS

FILE=ITEM,LINES=50
ID=ITMN=CH/4,VOL=50,ACCESS=RANDOM/i
*
LINE=01
  FIELD=DESC=CH/20
  FIELD=UPRC=CN/6.3
  FIELD=QTYO=CN/4.0
  FIELD=QTYH=CN/4.0
  ENDL
END.

          TOTAL PAGES REQUIRED - 16
           LINE LENGTH (BYTES) - 48
       TOTAL DESCRIPTION PAGES - 1
                TOTAL KEY PAGES - 4

  LINE 01 --   BASE = 14 , DATA = 34 , LINKAGE = 0   , TOTAL = 48

0112  **  NEW DATA BASE FILE CREATED  **
```

Figure B-4. DDL Listing for the Item File

You can enter a maximum of 300 lines of data in SOFL; however, you can enter no more than 50 sales orders (VOL = 50 on the ID line). If each sales order has an equal number of items and if the file is full, line types BL and 02 occur 50 times each, and line type 03 occurs 200 times; also, each sales order has four items specified.

The BILL, SHIP, and ITEM fields are secondary keys. By using these keys, you can write a program to read and summarize the sales orders in order to bill the customer. Use the SHIP field to obtain a summary of shipping information. To read and summarize the items currently on order, use the ITEM secondary key. You can also use the ITEM secondary key to check for items still in the sales order file when the item is to be deleted. Figure B-5 shows the DDL listing for SOFL.

B.3.4 The Initial Load Files

Use the batch stream control file .S$DBMS.TEST.LOADFILN to create the program file .S$DBMS.TEST.PROG and the two directories .S$DBMS.TEST.FILE (contains the data base files) and .S$DBMS.TEST.LIST (contains the reports generated by the test programs). The procedure deletes the data base files SOFL, CUST, and ITEM, if they exist, and then recreates them using the DDL compiler with the appropriate DDL definition file. The procedure then reloads the data base files with the data from .S$DBMS.TEST.ILDSOFL, .S$DBMS.TEST.ILDCUST, and .S$DBMS.TEST.ILDITEM and assigns a LUNO to the program file. To restore DBMS-990 files to base point, run .S$DBMS.TEST.LOADFILN.

NOTE

If security is installed, you must assign the synonym $P to the master password before loading the data base files.

B.3.5 Verifying the File Data

After executing the batch stream for loading the files or after deleting and reloading a particular file, verify the contents of the files by executing the following PQUERY session against the file. You can also verify the data by executing the individual programs and comparing the output obtained against the outputs documented in this appendix.

```
 DBMS-990     <L.V.R>              DDL TRANSLATOR      MM/DD/YY   HH:MM:SS

FILE=SOFL,LINES=300
ID=SONM=CH/6,VOL=50,ACCESS=RANDOM/1
*
LINE=BL
  FIELD=BILL=CH/5
  FIELD=LOCK=CH/2
  ENDL
*
LINE=02
  FIELD=SHIP=CH/5
  ENDL
*
LINE=03
  FIELD=ITEM=CH/4
  FIELD=QUAN=CN/4.0
  ENDL
*
SECONDARY-REFERENCES
BILL=VOL=50
SHIP=VOL=50
ITEM=VOL=200
END.

          TOTAL PAGES REQUIRED - 71
           LINE LENGTH (BYTES) - 32
       TOTAL DESCRIPTION PAGES - 1
                TOTAL KEY PAGES - 31

  LINE BL --  BASE = 16 , DATA = 7  , LINKAGE = 8  , TOTAL = 31
  LINE 02 --  BASE = 16 , DATA = 5  , LINKAGE = 8  , TOTAL = 29
  LINE 03 --  BASE = 16 , DATA = 8  , LINKAGE = 8  , TOTAL = 32

0112  **  NEW DATA BASE FILE CREATED  **
```

Figure B-5. DDL Listing for the Sales Order File

B.3.5.1 PQUERY Session with SOFL. The following PQUERY session demonstrates the output obtained from the sales order file (SOFL). This session enables you to verify the loading of SOFL.

```
PRIMITIVE QUERY
            FUNCTION: RS
             FILE ID: SOFL
              KEY ID: SONM
           KEY VALUE:
           FIELD IDS: BILLLOCK
NO. OF OUTPUT LINES: 50
           TERMINATE: NO
=========================================
     LINE TYPE IS: BL

SONM     BILL    LOCK
J80001   D0001
J80002   D0002
J80003   D0003
J80004   S0004
J80005   S0005
J80006   S0006
J80007   S0007
J80008   S0088
J80009   S0009
=======  END OF DATA LINES ======

PRIMITIVE QUERY
            FUNCTION: RS
             FILE ID: SOFL
              KEY ID: SONM
           KEY VALUE:
           FIELD IDS: SHIP
NO. OF OUTPUT LINES: 50
           TERMINATE: NO
=====================================
     LINE TYPE IS: 02

SONM     SHIP
J80001   S0001
J80002   S0002
J80003   S0003
J80004   S0004
J80005   S0007
J80006   S0006
J80006   A0010
J80006   T0200
J80006   Y0250
J80007   S0008
J80007   I0090
J80008   S0088
J80009   S0001
=======  END OF DATA LINES ======
```

```
==========================================================================
PRIMITIVE QUERY
            FUNCTION: RS
             FILE ID: SOFL
              KEY ID: SONM
           KEY VALUE:
           FIELD IDS: ITEMQUAN
NO. OF OUTPUT LINES: 50
           TERMINATE: YES
==========================================
     LINE TYPE IS: 03

SONM     ITEM  QUAN
J80001   A001  30303235
J80001   D004  30303035
J80001   E005  30303530
J80001   G007  30303333
J80001   N014  30303135
J80002   B002  30303037
J80002   D004  30303830
J80002   F006  30303235
J80002   J010  31303530
J80002   L012  30303939
J80002   S019  30313030
J80003   A001  32353030
J80003   B002  30313030
J80003   F006  39393939
J80003   G007  30353030
J80003   I009  30393939
J80003   L012  39303030
J80003   N014  30323530
J80003   T020  30353030
J80003   Y025  30323530
J80004   A001  30303235
J80004   C003  30353030
J80004   H008  30323530
J80004   N014  30393939
J80004   Z026  30303032
J80005   A001  30313030
J80005   B002  30303031
J80005   E005  30303535
J80005   I009  30303939
J80008   A001  30313030
J80009   A001  30303235
J80008   K011  30303130
J80008   Y025  30303035
J80008   Z026  30303032
=======  END OF DATA LINES ======
```

B.3.5.2 PQUERY Session with CUST. The following PQUERY session demonstrates the output obtained from the customer file (CUST). This session enables you to verify the loading of CUST.

```
================================================================
PRIMITIVE QUERY
            FUNCTION: RS
             FILE ID: CUST
              KEY ID: CUSN
           KEY VALUE:
           FIELD IDS: NAMESTRTCITYSTATZIPCCRED
NO. OF OUTPUT LINES: 50
           TERMINATE: YES
========================================
     LINE TYPE IS: 01

CUSN    NAME                    STRT                    CITY
STAT  ZIPC   CRED
D0001  HOLE EARTH DIST.         1234 MOUNTAIN LN.       LITTLE HILL
TX    78123  A1
D0002  ROUND WORLD CORP.        99 CIRCLE CT.           SPHERE
TX    78099  D1
D0003  ALLOVER SUPPLY CO.       8891 UNDER ST.          INSIDE
TX    78333  D3
S0001  THING-A-MA-GIG CORP.     1 HEYTHERE BLVD.        SPINEY
TX    78001  A1
S0002  WIDGITS, INC.            2345 WIDGIT AVE.        WHYNOT
TX    78234  B2
S0003  TOYS FOR TEXANS          3456 TEJAS AVE.         AUSTIN
TX    78345  A1
S0004  THUNDERBOLT CO.          4567 FLASH ST.          BRIGHT
TX    78456  F3
S0005  RAINMAKERS, INC.         5678 WHETTER BLVD.      DRY PLAINS
TX    78567  G5
S0006  ODD JOB WYRKERS          6789 UNEVEN ST.         ANOMALY
TX    78678  A3
S0007  TURKEYS, INC.            7890 GOBBLER CT.        BIG BIRD
TX    78789  A1
S0008  THING MAKERS, INC.       880 WHATISIT ST.        IDONTKNOW
TX    78888  C1
S0009  SPEEDY SUPPLY CO.        923 FASTEST ST.         INSTANT
TX    78999  S1
=======  END OF DATA LINES =======
```

B.3.5.3 PQUERY Session with ITEM. The following PQUERY session demonstrates the output obtained from the item file (ITEM). This enables you to verify the loading of ITEM.

```
==========================================================================
PRIMITIVE QUERY
            FUNCTION: RS
             FILE ID: ITEM
              KEY ID: ITMN
           KEY VALUE:
           FIELD IDS: DESCUPRCQTYOQTYH
NO. OF OUTPUT LINES: 50
           TERMINATE: YES
==============================================
     LINE TYPE IS: 01

ITMN  DESC                    UPRC           QTYO      QTYH
A001  ARMADILLOS              313030313233   30303031  30313233
B002  BLACK HOLES             303230323334   30303032  30323334
C003  CLAY                    303033333435   30303033  30333435
D004  DIPS                    303030343536   30303034  30343536
E005  ERECTORS                303035353637   30303035  30353637
F006  FREEBIES                303630363738   30303036  30363738
G007  GOOBERS                 373030373839   30303037  30373839
H008  HERBS                   303830383930   30303038  30383930
I009  IDIOMS                  303039393030   30303039  30393030
J010  JUMPS                   303031303030   30303130  31303030
K011  KILNS                   303131313030   30303131  31313030
L012  LONE STARS              313230303030   30303132  31323030
N014  NIBBLES                 303431343030   30303134  31343030
S019  SHOVELS                 313031393031   39303139  31393030
T020  TALES                   303939303939   30313030  34303030
Y025  YARNS                   303235303030   30303235  32353030
Z026  ZEBRAS                  303032363030   30303236  32363030
=======  END OF DATA LINES =======
```

B.4 EXECUTION OF THE COBOL PROGRAM, CEXMPL

Install and execute the COBOL program, CEXMPL, as follows:

1. Start the DBMS-990. If security is installed assign the password TEST to the data base files and assign the synonym $P to the value of the master password. This should be done by first using the ADDPSW procedure to add the password to the security file. Next, use the ADDPE procedure to assign the password to the files used by this program.

2. Initialize the data base files by executing the batch stream control file .S$DBMS.TEST.LOADFILN (use the XB command, followed by a WAIT command). When finished, the batch stream displays the following message:

   LUNO >## ASSIGNED TO THE PROGRAM FILE

3. Execute the batch stream .S$DBMS.TEST.CBATCHN to compile and install the program.

   This procedure uses link control file .S$DBMS.TEST.CNLNK to link edit the program and install it on the program file S$DBMS.TEST.PROG.

4. Check the batch stream listing files for errors. Ignore errors U SVC-0316 (file already exists) and U SVC-0315 (file does not exist) that occur while creating or deleting files.

5. Use the Assign Synonym (AS) command to assign synonyms to the input and output files, as follows:

Synonym	Value
CINP	.S$DBMS.TEST.TRAN
COUT	.S$DBMS.TEST.LIST.CRPT

6. Execute CEXMPL using the Execute COBOL Task Foreground command (XCTF). For the first parameter, enter the program file. Then enter CEXMPL as the task name and tab through the remaining prompts. The program executes for about one minute.

7. Display or print the file .S$DBMS.TEST.LIST.CRPT and compare it to the listing in Figure B-6. They should be the same.

8. After the program executes, you can halt DBMS-990 unless others are using the data base. Use the End DBMS (EDBMS) command.

Figure B-7 contains the source listing for CEXMPL. Figure B-8 contains the Link Editor control file for CEXMPL.

```
A001  ARMADILLOS          $100.123  0025  J80001  S0001  THING-A-MA-GIG CORP.
A001  ARMADILLOS          $100.123  2500  J80003  S0003  TOYS FOR TEXANS
A001  ARMADILLOS          $100.123  0025  J80004  S0004  THUNDERBOLT CO.
A001  ARMADILLOS          $100.123  0100  J80005  S0007  TURKEYS, INC.
A001  ARMADILLOS          $100.123  0100  J80008  S0088  ***NO SHIP NAME
A001  ARMADILLOS          $100.123  0025  J80009  S0001  THING-A-MA-GIG CORP.

B002  BLACK HOLES          $20.234  0007  J80002  S0002  WIDGITS, INC.
B002  BLACK HOLES          $20.234  0100  J80003  S0003  TOYS FOR TEXANS
B002  BLACK HOLES          $20.234  0001  J80005  S0007  TURKEYS, INC.

C003  CLAY                  $3.345  0500  J80004  S0004  THUNDERBOLT CO.

D004  DIPS                   $.456  0005  J80001  S0001  THING-A-MA-GIG CORP.
D004  DIPS                   $.456  0080  J80002  S0002  WIDGITS, INC.

E005  ERECTORS              $5.567  0050  J80001  S0001  THING-A-MA-GIG CORP.
E005  ERECTORS              $5.567  0055  J80005  S0007  TURKEYS, INC.

F006  FREEBIES             $60.678  0025  J80002  S0002  WIDGITS, INC.
F006  FREEBIES             $60.678  9999  J80003  S0003  TOYS FOR TEXANS

ABCD  *ITEM DOES NOT EXIST
ABCD  *ITEM DOES NOT EXIST                                  ***ITEM NOT SOLD

G007  GOOBERS             $700.789  0033  J80001  S0001  THING-A-MA-GIG CORP.
G007  GOOBERS             $700.789  0500  J80003  S0003  TOYS FOR TEXANS

H008  HERBS                $80.890  0250  J80004  S0004  THUNDERBOLT CO.

I009  IDIOMS                $9.900  0999  J80003  S0003  TOYS FOR TEXANS
I009  IDIOMS                $9.900  0099  J80005  S0007  TURKEYS, INC.

J010  JUMPS                 $1.000  1050  J80002  S0002  WIDGITS, INC.

K011  KILNS                $11.100  0010  J80008  S0088  ***NO SHIP NAME

L012  LONE STARS          $120.000  0099  J80002  S0002  WIDGITS, INC.
L012  LONE STARS          $120.000  9000  J80003  S0003  TOYS FOR TEXANS

N014  NIBBLES              $41.400  0015  J80001  S0001  THING-A-MA-GIG CORP.
N014  NIBBLES              $41.400  0250  J80003  S0003  TOYS FOR TEXANS
N014  NIBBLES              $41.400  0999  J80004  S0004  THUNDERBOLT CO.

S019  SHOVELS             $101.901  0100  J80002  S0002  WIDGITS, INC.

WXYZ  *ITEM DOES NOT EXIST
WXYZ  *ITEM DOES NOT EXIST                                  ***ITEM NOT SOLD

T020  TALES                $99.099  0500  J80003  S0003  TOYS FOR TEXANS

Y025  YARNS                $25.000  0250  J80003  S0003  TOYS FOR TEXANS
Y025  YARNS                $25.000  0005  J80008  S0088  ***NO SHIP NAME

Z026  ZEBRAS                $2.600  0002  J80004  S0004  THUNDERBOLT CO.
Z026  ZEBRAS                $2.600  0002  J80008  S0088  ***NO SHIP NAME
```

Figure B-6. Output from the COBOL Program CEXMPL

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CEXMPL.
 AUTHOR. TEXAS INSTRUMENTS, INC.
 DATE-WRITTEN. OCTOBER, 1978.
*
*    THIS PROGRAM WILL EXTRACT DATA FROM THE DBMS
*    SALES-ORDER FILE ABOUT SPECIFIC ITEM NUMBERS
*    READ FROM A SEQUENTIAL INPUT TRANSACTION FILE.
*    DATA THAT IS RETRIEVED FROM THE DATA BASE FILES
*    IS OUTPUT TO A SEQUENTIAL FILE WHICH CAN BE
*    DISPLAYED AFTER THE PROGRAM TERMINATES.
*
*    THE ITEM DESCRIPTION AND UNIT PRICE ARE OBTAINED
*    FROM THE ITEM FILE.  THE SALES-ORDER NUMBER,
*    SHIP-TO CUSTOMER NUMBER AND QUANTITY ON-ORDER
*    FOR EACH SALES ORDER THAT CONTAINS THE ITEM ARE
*    ARE OBTAINED FROM THE SOFL FILE.  THE SHIP-TO
*    CUSTOMER NAME IS RETRIEVED FROM THE CUST FILE.
*    APPROPRIATE ERROR MESSAGES ARE PRINTED WHERE
*    APPLICABLE.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. TI-990.
 OBJECT-COMPUTER. TI-990.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
*
*    A SEQUENTIAL FILE CONTAINING THE ITEM NUMBERS
*    TO BE PROCESSED MUST HAVE THE SYNONYM  CINP
*    ASSIGNED TO IT BEFORE THIS PROGRAM CAN BE
*    EXECUTED
*
 SELECT TINFILE
      ASSIGN TO INPUT, "CINP";
      ORGANIZATION IS SEQUENTIAL;
      ACCESS IS SEQUENTIAL.
*
*    A SEQUENTIAL FILE FOR OUTPUT MUST HAVE
*    THE SYNONYM  COUT  ASSIGNED TO IT BEFORE
*    THIS PROGRAM CAN BE EXECUTED
*
 SELECT OUTFILE
      ASSIGN TO OUTPUT, "COUT";
      ORGANIZATION IS SEQUENTIAL;
      ACCESS IS SEQUENTIAL.
 DATA DIVISION.
 FILE SECTION.
 FD TINFILE;
      LABEL RECORDS ARE OMITTED;
      DATA RECORD IS TINREC.
 01  TINREC.
     05  ITEMNO    PIC X(4).
     05  FILLER    PIC X(75).
 FD OUTFILE;
      LABEL RECORDS ARE OMITTED;
      DATA RECORD IS OUTREC.
 01  OUTREC.
     05  FILLER   PIC X(80).
```

Figure B-7. Listing of Program CEXMPL (Sheet 1 of 6)

```
 WORKING-STORAGE SECTION.
*
*     WORK AREAS AND INDICATORS
*
 77  SLOC1          PIC X(4) VALUE "****".
 77  SLOC2          PIC X(4) VALUE "****".
 01  EOF-IND        PIC X VALUE "N".
     88  TRAN-EOF VALUE "Y".
 01  ITEM-IND       PIC X VALUE SPACES.
     88  ITEM-EXISTS VALUE "Y".
 01  MORE-IND       PIC X VALUE SPACES.
     88  MORE-SOFL-ITEMS VALUE "M".
     88  NO-MORE-SOFL-ITEMS VALUE "N".
 01  SHIP-IND       PIC X  VALUE SPACES.
     88  SHIP-EXISTS VALUE "Y".
 01  SOLD-IND       PIC X  VALUE SPACES.
     88  ITEM-SOLD VALUE "Y".
 01  DATAREC.
     05  OITEMNO  PIC X(4).
     05  FILLER   PIC X(2).
     05  ODESCRPT PIC X(20).
     05  FILLER   PIC X(2).
     05  OPRICE   PIC $$$$.999.
     05  FILLER   PIC X(2).
     05  OQTYOO   PIC X(4).
     05  FILLER   PIC X(2).
     05  OSONO    PIC X(6).
     05  FILLER   PIC X(2).
     05  OSHIPNO  PIC X(5).
     05  FILLER   PIC X(2).
     05  OSHIPNA  PIC X(20).
     05  FILLER   PIC X(1).
 01  ERRREC.
     05  EMSG     PIC X(8) VALUE "DBERROR ".
     05  FILLER   PIC X(5) VALUE "STAT=".
     05  ESTAT    PIC X(2) VALUE SPACES.
     05  FILLER   PIC X(9) VALUE ", DBFILE=".
     05  EFILE    PIC X(4) VALUE SPACES.
     05  FILLER   PIC X(7) VALUE ", KEYN=".
     05  EKEYN    PIC X(4) VALUE SPACES.
     05  FILLER   PIC X(7) VALUE ", KEYV=".
     05  EKEYV    PIC X(6) VALUE SPACES.
 01 ERROR-MSG.
     10  FILLER   PIC X(9) VALUE "ERROR IN ".
     10  ERR-FILE PIC X(4).
     10  FILLER   PIC X(19) VALUE " FILE OPEN, STATUS=".
     10  ERR-STAT PIC XX.
*
*     DBMS DML CALL PARAMETER AREAS
*
*     IF SECURITY IS INSTALLED ON YOUR DBMS, THE VALUE OF
*     PSWD DATA ITEM IN THE CONTROL BLOCK MUST BE CHANGED
*     TO THE PASSWORD THAT WILL BE ASSIGNED TO THE SOFL,
*     CUST AND ITEM DATA BASE FILES. NOTE: SINCE THERE
*     IS ONLY ONE CONTROL BLOCK IN THIS PROGRAM ALL THREE
*     FILES SHOULD HAVE THE SAME PASSWORD.
*
```

Figure B-7. Listing of Program CEXMPL (Sheet 2 of 6)

```
*         DUMMY ADDRESSES USED WITH THE FILE ACCESS CHECKING
*         FOR OPEN AND CLOSE DATABASE FILE FUNCTIONS.
 01  D1                  PIC X.
 01  D2                  PIC X.
*         CONTROL BLOCK
 01  CB.
     02 PSWD             PIC X(4)   VALUE  "TEST".
     02 FUNC             PIC XX     VALUE  "OP".
     02 STAT             PIC XX     VALUE  "**".
     02 DBFILE           PIC X(4)   VALUE  "    ".
     02 LOC1             PIC X(4)   VALUE  "****".
     02 LOC2             PIC X(4)   VALUE  "****".
     02 KEYN             PIC X(4)   VALUE  "SHRD".
     02 KEYV             PIC X(6).
*       SALES ORDER FILE PRIMARY KEY READ LINE LIST
 01  SOFLPK-LL.
     02 SPLTYPE        PIC X(7)      VALUE  "LINE=02".
     02 SPRETIND       PIC X         VALUE  "*".
     02 SPFIELDS       PIC X(4)      VALUE  "SHIP".
     02 SPDISP         PIC X(8)      VALUE  "****RLSE".
*         SOFL SECONDARY KEY READ LINE LIST
 01  SOFLSK-LL.
     02 SSLTYPE        PIC X(7)      VALUE  "LINE=03".
     02 SSRETIND       PIC X         VALUE  "*".
     02 SSFIELDS       PIC X(8)      VALUE  "QUANSONM".
     02 SSDISP         PIC X(8)      VALUE  "****RLSE".
*         CUSTOMER FILE LINE LIST
 01  CUST-LL.
     02 CLTYPE       PIC X(7)      VALUE  "LINE=01".
     02 CRETIND      PIC X         VALUE  "*".
     02 CFIELDS      PIC X(4)      VALUE  "NAME".
     02 CDISP        PIC X(8)      VALUE  "****RLSE".
*         ITEM FILE LINE LIST
 01  ITEM-LL.
     02 ILTYPE       PIC X(7)      VALUE  "LINE=01".
     02 IRETIND      PIC X         VALUE  "*".
     02 IFIELDS      PIC X(8)      VALUE  "DESCUPRC".
     02 IDISP        PIC X(8)      VALUE  "****RLSE".
*         DBMS FILE DATA AREAS
 01  DA.
     02 FILLER      PIC X(26).
 01  SOFL02-DA REDEFINES DA.
     02 SHIP        PIC X(6).
     02 FILLER      PIC X(20).
 01  SOFL03-DA REDEFINES DA.
     02 QUAN        PIC X(4).
     02 SONM        PIC X(6).
     02 FILLER      PIC X(16).
 01  CUST-DA REDEFINES DA.
     02 NAME        PIC X(20).
     02 FILLER      PIC X(6).
 01  ITEM-DA REDEFINES DA.
     02 DESC        PIC X(20).
     02 UPRC        PIC 9(3)V9(3).
 01  DB-DELIM            PIC XX    VALUE "/*".
/
```

Figure B-7. Listing of Program CEXMPL (Sheet 3 of 6)

```
PROCEDURE DIVISION.
MAIN-PROG.
     OPEN INPUT TINFILE, OUTPUT OUTFILE.
     MOVE SPACES TO OUTREC DATAREC.
     MOVE "OF" TO FUNC. MOVE "SHRD" TO KEYN.
     MOVE "CUST" TO DBFILE.
     PERFORM OPEN-DATABASE-FILE.
     MOVE "ITEM" TO DBFILE.
     PERFORM OPEN-DATABASE-FILE.
     MOVE "SOFL" TO DBFILE.
     PERFORM OPEN-DATABASE-FILE.
     MOVE "RF" TO FUNC.
     PERFORM PROCESS-TRAN UNTIL TRAN-EOF.
END-OF-RUN.
     MOVE "CF" TO FUNC.
     MOVE "CUST" TO DBFILE.
     CALL "DBMSYS" USING CB, SOFLPK-LL, D1, D2, D1, D2.
     MOVE "ITEM" TO DBFILE.
     CALL "DBMSYS" USING CB, SOFLPK-LL, D1, D2, D1, D2.
     MOVE "SOFL" TO DBFILE.
     CALL "DBMSYS" USING CB, SOFLPK-LL, D1, D2, D1, D2.
     CLOSE TINFILE OUTFILE.
     STOP RUN.
/
OPEN-DATABASE-FILE.
     CALL "DBMSYS" USING CB, SOFLPK-LL, D1, D2, D1, D2.
     IF STAT NOT = "**"
        MOVE DBFILE TO ERR-FILE
        MOVE STAT TO ERR-STAT
        DISPLAY ERROR-MSG LINE 24
        ACCEPT D1 LINE 24
        PROMPT GO TO END-OF-RUN.
PROCESS-TRAN.
     PERFORM READ-TRAN.
     IF NOT TRAN-EOF
         PERFORM DBMS-ROUTINES
         MOVE SPACES TO TINREC DATAREC ESTAT EFILE EKEYN
                     EKEYV SOLD-IND ITEM-IND SHIP-IND
                     MORE-IND.
         PERFORM OUTPUT-INFO.
DBMS-ROUTINES.
     PERFORM GET-ITEM-FROM-ITEM.
     PERFORM GET-ITEM-FROM-SOFL.
     IF ITEM-EXISTS AND ITEM-SOLD
        PERFORM PROCESS-ITEM UNTIL NO-MORE-SOFL-ITEMS.
PROCESS-ITEM.
     PERFORM GET-SHIP-FROM-SOFL.
     IF SHIP-EXISTS
         PERFORM GET-NAME-FROM-CUST.
     IF STAT IS EQUAL TO "**" AND SHIP-EXISTS
         PERFORM OUTPUT-INFO.
     IF SLOC2 EQUAL "****"
         MOVE "N" TO MORE-IND
         MOVE "****" TO SLOC1
     ELSE
         MOVE "M" TO MORE-IND
         PERFORM GET-ITEM-FROM-SOFL.
```

Figure B-7. Listing of Program CEXMPL (Sheet 4 of 6)

```
READ-TRAN.
    READ TINFILE RECORD
        AT END MOVE "Y" TO EOF-IND.
    IF NOT TRAN-EOF
        MOVE "****" TO LOC1 LOC2
        MOVE ITEMNO TO KEYV OITEMNO.
OUTPUT-INFO.
    MOVE DATAREC TO OUTREC.
    PERFORM WRITE-DATA.
WRITE-DATA.
    WRITE OUTREC.
INIT-LOCS.
     MOVE "****" TO LOC1 LOC2.
GET-ITEM-FROM-ITEM.
     MOVE "ITEM" TO DBFILE.
     MOVE "ITMN" TO KEYN.
     PERFORM INIT-LOCS.
     CALL "DBMSYS" USING CB SOFLPK-LL, ITEM-LL DA,
                         DA DB-DELIM.
     IF STAT IS EQUAL TO "**"
         AND LOC1 NOT EQUAL "****"
            MOVE DESC TO ODESCRPT
            MOVE UPRC TO OPRICE
            MOVE "Y" TO ITEM-IND
     ELSE
          IF STAT EQUAL TO "NK"
            MOVE "*ITEM DOES NOT EXIST" TO ODESCRPT
            PERFORM OUTPUT-INFO
          ELSE
            PERFORM ERR-ROUTINE.
GET-ITEM-FROM-SOFL.
     MOVE "SOFL" TO DBFILE.
     MOVE "ITEM" TO KEYN.
     MOVE ITEMNO TO KEYV.
     IF MORE-SOFL-ITEMS
        MOVE SLOC1 TO LOC1
        MOVE SLOC2 TO LOC2
     ELSE
        PERFORM INIT-LOCS.
     CALL "DBMSYS" USING CB SOFLSK-LL, SOFLSK-LL CUST-LL,
                         DA DB-DELIM.
     IF STAT IS EQUAL TO "**"
         IF LOC1 EQUAL "****"
            MOVE "N" TO MORE-IND
         ELSE
            MOVE QUAN TO OQTYOO
            MOVE SONM TO OSONO
            MOVE "Y" TO SOLD-IND
            MOVE LOC1 TO SLOC1
            MOVE LOC2 TO SLOC2
      ELSE
            MOVE "N" TO MORE-IND
        IF STAT EQUAL TO "NK"
            MOVE "***ITEM NOT SOLD" TO OSHIPNA
            PERFORM OUTPUT-INFO
        ELSE
            PERFORM ERR-ROUTINE.
```

Figure B-7. Listing of Program CEXMPL (Sheet 5 of 6)

```
GET-SHIP-FROM-SOFL.
     MOVE "SOFL" TO DBFILE.
     MOVE "SONM" TO KEYN.
     MOVE SONM TO KEYV.
     PERFORM INIT-LOCS.
     CALL "DBMSYS" USING CB SOFLPK-LL, SOFLPK-LL SOFLSK-LL,
                         DA DB-DELIM.
     IF STAT IS EQUAL TO "**"
         IF LOC1 NOT EQUAL "****"
            MOVE "Y" TO SHIP-IND
            MOVE SHIP TO OSHIPNO
         ELSE
            MOVE SPACES TO SHIP-IND
            MOVE "***NO SHIP IN SOFL " TO OSHIPNA
            PERFORM OUTPUT-INFO
     ELSE
          PERFORM ERR-ROUTINE.
GET-NAME-FROM-CUST.
    MOVE "CUST" TO DBFILE.
    MOVE "CUSN" TO KEYN.
    MOVE SHIP TO KEYV.
    PERFORM INIT-LOCS.
    CALL "DBMSYS" USING CB SOFLPK-LL, CUST-LL ITEM-LL,
                         DA DB-DELIM.
     IF STAT IS EQUAL TO "**"
         AND LOC1 NOT EQUAL "****"
            MOVE NAME TO OSHIPNA
     ELSE
        IF STAT EQUAL TO "NK"
            MOVE "***NO SHIP NAME " TO OSHIPNA
            PERFORM OUTPUT-INFO
        ELSE
            PERFORM ERR-ROUTINE.
ERR-ROUTINE.
    MOVE DBFILE TO EFILE.
    MOVE KEYN TO EKEYN.
    MOVE KEYV TO EKEYV.
    MOVE STAT TO ESTAT.
    MOVE ERRREC TO OUTREC.
    PERFORM WRITE-DATA.
END PROGRAM.
```

Figure B-7. Listing of Program CEXMPL (Sheet 6 of 6)

```
 SDSLNK        3.2.1          78.275              PAGE        1
COMMAND LIST

FORMAT IMAGE      ,REPLACE
PROC RCOBOL
DUMMY
INCLUDE .S$SYSLIB.RCBPRC
TASK CEXMPL
INCLUDE .S$SYSLIB.RCBTSK
INCLUDE .S$SYSLIB.RCBMPD
INCLUDE S$DBMS.TEST.COBJECT
INCLUDE S$DBMS.SNDMSG
INCLUDE S$DBMS.COBINT
END
```

Figure B-8. Link Control File for Program CEXMPL

B.5 EXECUTION OF THE FORTRAN PROGRAM, FEXMPN

Install and execute the FORTRAN program, FEXMPN, as follows:

1. Start the DBMS-990. If security is installed assign the password TEST to the data base files and assign the synonym $P to the value of the master password. This should be done by first using the ADDPSW procedure to add the password to the security file. Next, use the ADDPE procedure to assign the password to the files used by this program.

2. Initialize the data base files by executing the batch stream control file .S$DBMS.TEST.LOADFILN (use the XB command, followed by a WAIT command). When finished, the batch stream displays the following message:

   LUNO >## ASSIGNED TO THE PROGRAM FILE

3. Compile and install the program by executing the batch stream .S$DBMS.TEST.FBATCHN. This procedure uses the link control file .S$DBMS.TEST.FNLNKN to link edit the program and install it on the file .S$DBMS.TEST.PROG.

4. Check the batch stream listing files for errors. Ignore errors 0026 (file already exists) and 0027 (file does not exist) that occur while creating or deleting files.

5. Use the Assign Synonym (AS) command to assign synonyms to the input and output files, as follows:

Synonym	Value
UNIT5	.S$DBMS.TEST.TRAN
UNIT6	.S$DBMS.TEST.LIST.FRPT

6. Execute a Map Program File (MPF) command on the program file .S$DBMS.TEST.PROG to find the task identification (TASK ID) of FEXMPN. Execute the program using the Execute FORTRAN Task Foreground (XFTF) command, specifying the LUNO reported in step 2. Enter the TASK ID for the program.

7. Display or print the file .S$DBMS.TEST.LIST.FRPT and compare it to the listing in Figure B-9. They should be the same.

8. After the program executes, you can halt DBMS-990 unless others are using the data base. Use the End DBMS (EDBMS) command.

Figure B-9 contains the output from the FORTRAN program FEXMPN. Figure B-10 contains the source listing of the program FEXMPN. Figure B-11 contains the Link Editor control file for the program FEXMPN.

```
A001   ARMADILLOS              100123  0025  J80001  S0001  THING-A-MA-GIG CORP.
A001   ARMADILLOS              100123  2500  J80003  S0003  TOYS FOR TEXANS
A001   ARMADILLOS              100123  0025  J80004  S0004  THUNDERBOLT CO.
A001   ARMADILLOS              100123  0100  J80005  S0007  TURKEYS, INC.
A001   ARMADILLOS              100123  0100  J80008  S0088  ***NO SHIP NAME
A001   ARMADILLOS              100123  0025  J80009  S0001  THING-A-MA-GIG CORP.

B002   BLACK HOLES             020234  0007  J80002  S0002  WIDGITS, INC.
B002   BLACK HOLES             020234  0100  J80003  S0003  TOYS FOR TEXANS
B002   BLACK HOLES             020234  0001  J80005  S0007  TURKEYS, INC.

C0003  CLAY                    003345  0500  J80004  S0004  THUNDERBOLT CO.

D004   DIPS                    000456  0005  J80001  S0001  THING-A-MA-GIG CORP.
D004   DIPS                    000456  0080  J80002  S0002  WIDGITS, INC.

E005   ERECTORS                005567  0050  J80001  S0001  THING-A-MA-GIG CORP.
E005   ERECTORS                005567  0055  J80005  S0007  TURKEYS, INC.

F006   FREEBIES                060678  0025  J80002  S0002  WIDGITS, INC.
F006   FREEBIES                060678  9999  J80003  S0003  TOYS FOR TEXANS

ABCD   *ITEM DOES NOT EXIST
ABCD   *ITEM DOES NOT EXIST                                   ***ITEM NOT SOLD

G007   GOOBERS                 700789  0033  J80001  S0001  THING-A-MA-GIG CORP.
G007   GOOBERS                 700789  0500  J80003  S0003  TOYS FOR TEXANS

H008   HERBS                   080890  0250  J80004  S0004  THUNDERBOLT CO.

I009   IDIOMS                  009900  0999  J80003  S0003  TOYS FOR TEXANS
I009   IDIOMS                  009900  0099  J80005  S0007  TURKEYS, INC.

J010   JUMPS                   001000  1050  J80002  S0002  WIDGITS, INC.

K011   KILNS                   011100  0010  J80008  S0088  ***NO SHIP NAME

L012   LONE STARS              120000  0099  J80002  S0002  WIDGITS, INC.
L012   LONE STARS              120000  9000  J80003  S0003  TOYS FOR TEXANS

N014   NIBBLES                 041400  0015  J80001  S0001  THING-A-MA-GIG CORP.
N014   NIBBLES                 041400  0250  J80003  S0003  TOYS FOR TEXANS
N014   NIBBLES                 041400  0999  J80004  S0004  THUNDERBOLT CO.

S019   SHOVELS                 101901  0100  J80002  S0002  WIDGITS, INC.

WXYZ   *ITEM DOES NOT EXIST
WXYZ   *ITEM DOES NOT EXIST                                   ***ITEM NOT SOLD

T020   TALES                   099099  0500  J80003  S0003  TOYS FOR TEXANS

Y025   YARNS                   025000  0250  J80003  S0003  TOYS FOR TEXANS
Y025   YARNS                   025000  0005  J80008  S0088  ***NO SHIP NAME

Z026   ZEBRAS                  002600  0002  J80004  S0004  THUNDERBOLT CO.
Z026   ZEBRAS                  002600  0002  J80008  S0088  ***NO SHIP NAME
```

Figure B-9. Output from the FORTRAN Program FEXMPN

```
C*    THIS PROGRAM WILL EXTRACT DATA FROM THE DBMS SALES-ORDER
C*    FILE ABOUT SPECIFIC ITEM NUMBERS READ FROM A SEQUENTIAL
C*    INPUT TRANSACTION FILE.  DATA THAT IS RETRIEVED FROM THE
C*    DATA BASE FILES IS OUTPUT TO A SEQUENTIAL FILE WHICH CAN
C*    BE DISPLAYED AFTER THE PROGRAM TERMINATES.
C*
C*    THE ITEM DESCRIPTION AND UNIT PRICE ARE OBTAINED FROM
C*    THE ITEM FILE.  THE SALES-ORDER NUMBER, SHIP-TO CUSTOMER
C*    NUMBER AND QUANTITY ON ORDER FOR EACH SALES ORDER THAT
C*    CONTAINS THE ITEM ARE OBTAINED FROM THE SOFL FILE.
C*    THE SHIP-TO CUSTOMER NAME IS RETRIEVED FROM THE CUST
C*    FILE.  APPROPRIATE ERROR MESSAGES ARE PRINTED WHERE
C*    APPLICABLE.
C*
C*        TWO SYNONYMS ARE REQUIRED TO RUN THE PROGRAM:
C*
C*           SYNONYM              VALUE
C*            UNIT5            INPUT TRANSACTION FILE PATHNAME
C*                                FILE .DBLIB.TEST.TRAN IS SUPPLIED
C*                                AND INSTALLED AT DBGEN TIME
C*            UNIT6            OUTPUT FILE OR PRINTER PATHNAME
C*
      IMPLICIT INTEGER (A-Z)
      DIMENSION INKEY(2)
      DIMENSION DMSFLS(15),DBDMY(1)
      DIMENSION ITEMCB(15),ITEMLL(13),ITEMDA(14)
      DIMENSION SOFLC3(15),SOFLL3(13),SOFLD3(6)
      DIMENSION SOFLC2(16),SOFLL2(11),SOFLD2(4)
      DIMENSION CUSTCB(16),CUSTLL(11),CUSTDA(11)
C*
C*  INITIALIZE ARRAYS
C*
      DATA DMSFLS /'TESTOF  CUST********SHRD       '/
      DATA ITEMCB /'TESTRF  ITEM********ITMN       '/
      DATA ITEMLL /'LINE=01*DESCUPRC****RLSE   '/
      DATA SOFLC3 /'TESTRF  SOFL********ITEM       '/
      DATA SOFLL3 /'LINE=03*QUANSONM****RLSE   '/
      DATA SOFLC2 /'TESTRF  SOFL********SONM         '/
      DATA SOFLL2 /'LINE=02*SHIP****RLSE   '/
      DATA CUSTCB /'TESTRF  CUST********CUSN         '/
      DATA CUSTLL /'LINE=01*NAME****RLSE   '/
C*
C*  SET UP CONSTANTS
C*
      CAST = '**'
      CNK  = 'NK'
C*
C*    THE FOLLOWING SECTION OF CODE IS USED TO OPEN ALL
C*    OF THE DATABASE FILES.  THE PROCEDURE IS NEEDED
C*    IF FILE ACCESS CHECKING WAS REQUESTED AT SYSTEM
C*    GENERATION.  HOWEVER, IF IT WAS NOT REQUESTED,
C*    NO HARM WILL BE CAUSED BY OPENING THE FILES BEFORE
C*    ACCESSING THEM THROUGH THE PROGRAM.
C*
      CALL DBMSYS ( DMSFLS (1), DMSFLS (15),
     1              DBDMY (1),  DBDMY (1),
     2              DBDMY (1),  DBDMY (1) )
      IF ( DMSFLS (4) .NE. CAST )  GO TO 998
      DMSFLS(5) = 'IT'
      DMSFLS(6) = 'EM'
```

Figure B-10. Listing of Program FEXMPN (Sheet 1 of 5)

```
      CALL DBMSYS ( DMSFLS (1), DMSFLS (15),
     1                DBDMY (1),  DBDMY (1),
     2                DBDMY (1),  DBDMY (1) )
      IF ( DMSFLS (4) .NE. CAST )  GO TO 998
      DMSFLS(5) = 'SO'
      DMSFLS(6) = 'FL'
      CALL DBMSYS ( DMSFLS (1), DMSFLS (15),
     1                DBDMY (1),  DBDMY (1),
     2                DBDMY (1),  DBDMY (1) )
      IF ( DMSFLS (4) .NE. CAST )  GO TO 998
C*
1     CONTINUE
C*
C*  READ ITEM NUMBER KEY FROM INPUT FILE.
      READ (5,100,END=999) INKEY(1), INKEY(2)
100   FORMAT (2A2)
      WRITE (6,211)
211   FORMAT (1X)
C*
C*  GET ITEM FROM ITEM FILE.
C*  SET  LOC1 AND LOC2 TO "*"
      ITEMCB  (7) = '**'
      ITEMCB  (8) = '**'
      ITEMCB  (9) = '**'
      ITEMCB (10) = '**'
C*
C*  SET KEY VALUE
      ITEMCB (13) = INKEY (1)
      ITEMCB (14) = INKEY (2)
C*
C*  CALL DATA BASE
      CALL DBMSYS ( ITEMCB(1), ITEMCB(15),
     1                ITEMLL(1), ITEMLL(13),
     2                ITEMDA(1), ITEMDA(14) )
      IF ( ITEMCB (4) .NE. CAST )  GO TO 900
C*
C*  GET ITEM FOR LINE 3 OF SALES ORDER FILE
C*  SET  LOC1 AND LOC2 TO "*"
      SOFLC3  (7) = '**'
      SOFLC3  (8) = '**'
      SOFLC3  (9) = '**'
      SOFLC3 (10) = '**'
C*
C*  SET KEY VALUE
      SOFLC3 (13) = INKEY (1)
      SOFLC3 (14) = INKEY (2)
C*
C*  CALL DATA BASE
      CALL DBMSYS ( SOFLC3(1), SOFLC3(16),
     1                SOFLL3(1), SOFLL3(13),
     2                SOFLD3(1), SOFLD3(6) )
      IF ( SOFLC3 (4) .NE. CAST )  GO TO 920
C*
C*  START LOOP TO PRINT INFORMATION - FOUND AT LEAST 1 ORDER
C*  SET  LOC1 AND LOC2 TO "*"
      SOFLC3 (7) = '**'
      SOFLC3 (8) = '**'
      SOFLC3 (9) = '**'
      SOFLC3 (10) = '**'
```

Figure B-10. Listing of Program FEXMPN (Sheet 2 of 5)

```
C*
C*  SET KEY VALUE
       SOFLC3 (13) = INKEY (1)
       SOFLC3 (14) = INKEY (2)
C*
C*  SET UP LOOP
2      CONTINUE
C*
C*  CALL DBMS TO RETRIEVE NEXT LINE=03
C*
       CALL DBMSYS ( SOFLC3(1), SOFLC3(15),
     1               SOFLL3(1), SOFLL3(13),
     2               SOFLD3(1), SOFLD3(6) )
       IF ( SOFLC3 (4) .NE. CAST )  GO TO 930
C*
C*  MOVE SALES ORDER NUMBER TO SALES ORDER RETRIEVAL
       SOFLC2 (13) = SOFLD3 (3)
       SOFLC2 (14) = SOFLD3 (4)
       SOFLC2 (15) = SOFLD3 (5)
C*
C*  MOVE "*" TO LOC1 AND LOC2
       SOFLC2  (7) = '**'
       SOFLC2  (8) = '**'
       SOFLC2  (9) = '**'
       SOFLC2 (10) = '**'
C*
C*  RETRIEVE SHIP-TO CUSTOMER NUMBER FROM SALES ORDER FILE
       CALL DBMSYS (  SOFLC2 (1), SOFLC2 (16), SOFLL2 (1),
     1     SOFLL2 (11), SOFLD2 (1), SOFLD2 (4)  )
       IF ( SOFLC2 (4) .NE. CAST ) GO TO 935
C*
C*  SET UP TO RETRIEVE NAME OF SHIP TO CUSTOMER FROM THE
C*  CUSTOMER FILE  MOVE "*" TO LOC1 AND LOC2
       CUSTCB  (7) = '**'
       CUSTCB  (8) = '**'
       CUSTCB  (9) = '**'
       CUSTCB (10) = '**'
C*
C*  MOVE CUSTOMER NUMBER RETRIEVED FROM SOFL LINE 02 TO
C*  CUSTOMER FILE KEY VALUE
       CUSTCB (13) = SOFLD2 (1)
       CUSTCB (14) = SOFLD2 (2)
       CUSTCB (15) = SOFLD2 (3)
C*
C*  RETRIEVE CUSTOMER NAME
       CALL DBMSYS (  CUSTCB (1), CUSTCB (16), CUSTLL (1),
     1     CUSTLL (11), CUSTDA (1), CUSTDA (11)   )
       IF ( CUSTCB (4) .NE. CAST ) GO TO 940
       WRITE (6,200) INKEY (1), INKEY (2), (ITEMDA (I), I = 1, 13),
     1   (SOFLD3 (I), I = 1, 5), (SOFLD2 (I), I = 1, 3),
     2   (CUSTDA (I), I = 1, 10)
200    FORMAT (1X2A2, 2X10A2, 3X3A2, 2X2A2, 2(2X3A2), 2X10A2)
C*
C*  CHECK FOR LAST LINE=03 IN SOFL FOR ITEM.
5      CONTINUE
       IF ( SOFLC3 (9) .EQ. CAST .AND. SOFLC3 (10) .EQ. CAST )
     1       GO TO 1
       GO TO 2
```

Figure B-10. Listing of Program FEXMPN (Sheet 3 of 5)

```
C*
C*  EXCEPTION PROCESSING
C*
C*  ITEM WAS NOT FOUND OR ERROR
900     IF ( ITEMCB (4) .NE. CNK ) GO TO 995
        WRITE (6,201) INKEY (1), INKEY (2)
201     FORMAT (1X2A2, 2X'*ITEM DOES NOT EXIST ')
C*
C*  CHECK TO SEE THAT NO ONE HAS ORDER ANY OF THE ITEM
C*  WHICH DOES NOT EXIST
C*  GET ITEM FOR LINE 3 OF SALES ORDER FILE
C*  SET  LOC1 AND LOC2 TO "*"
        SOFLC3  (7) = '**'
        SOFLC3  (8) = '**'
        SOFLC3  (9) = '**'
        SOFLC3 (10) = '**'
C*
C*  SET KEY VALUE
        SOFLC3 (13) = INKEY (1)
        SOFLC3 (14) = INKEY (2)
C*
C*  CALL DATA BASE
        CALL DBMSYS (  SOFLC3 (1), SOFLC3 (15), SOFLL3 (1),
     1       SOFLL3 (13), SOFLD3 (1), SOFLD3 (6)   )
        IF ( SOFLC3 (4) .EQ. CNK )  GO TO 901
C*
C*  FOUND ADDITIONAL EXCEPTION
        WRITE (6,202) INKEY (1), INKEY (2), SOFLC3 (4)
202     FORMAT (1X2A2, 2X'*ITEM DOES NOT EXIST, BUT THE '
     1    'STATUS ', 2XA2, ' WAS RETURNED FROM SOFL ')
        GO TO 1
C*
C* NO ORDER WAS ENTERED
901     WRITE (6,203) INKEY (1), INKEY (2)
203     FORMAT (1X2A2, 2X'*ITEM DOES NOT EXIST', 32X
     1    '***ITEM NOT SOLD' )
        GO TO 1
C*
C*  FOUND EXCEPTION ON READ OF SALES ORDER FILE
920     IF ( SOFLC3 (4) .NE. CNK ) GO TO 996
        WRITE (6,204) INKEY (1), INKEY (2), (ITEMDA (I), I= 1, 13)
204     FORMAT (1X2A2, 2X10A2, 3X3A2, 23X'***NO SALES ORDERS')
        GO TO 1
C*
C*  FOUND EXCEPTION ON SECOND READ OF SALES ORDER FILE
930     WRITE (6,205) INKEY (1), INKEY (2), (ITEMDA (I), I = 1, 13),
     1    SOFLC3 (4)
205     FORMAT (1X2A2, 2X10A2, 3X3A2, 2X' EXCEPTION STATUS ',A2)
        GO TO 1
C*
C*  FOUND EXCEPTION ON LINE=02 READ FOR SHIP-TO NUMBER
935     WRITE (6,206) INKEY (1), INKEY (2), SOFLC2 (13),
     1   SOFLC2 (14), SOFLC2 (15), SOFLC2 (4)
206     FORMAT (1X2A2, 2X' EXCEPTION STATUS ON LINE=02 OF SONM ',
     1    3A2, 2X'STATUS', 2XA2 )
        GO TO 1
```

Figure B-10. Listing of Program FEXMPN (Sheet 4 of 5)

```
C*  FOUND EXCEPTION ON READ OF CUSTOMER FILE
940     IF ( CUSTCB (4) .NE. CNK ) GO TO 997
        WRITE (6,207) INKEY (1), INKEY (2), (ITEMDA (I), I = 1, 13),
     1    (SOFLD3 (I), I = 1, 5), (SOFLD2 (I), I = 1, 3)
207     FORMAT (1X2A2, 2X10A2, 3X3A2, 2X2A2, 2(2X3A2),
     1    2X'***NO SHIP NAME' )
        GO TO 5
C*
C*    GENERAL EXCEPTION
995     WRITE (6,208) ITEMCB (4)
208     FORMAT (' EXCEPTION ON ITEM FILE ... STATUS ',A2)
        GO TO 1
996     WRITE (6,209) SOFLC3 (4)
209     FORMAT (' EXCEPTION ON SOFL FILE ... STATUS ',A2)
        GO TO 1
997     WRITE (6,210) CUSTCB (4)
210     FORMAT (' EXCEPTION ON CUST FILE ... STATUS ',A2)
        GO TO 1
C*
C*    FILE OPEN EXCEPTION PROCESSING
C*
998     WRITE (6,299) DMSFLS(5), DMSFLS(6), DMSFLS(4)
299     FORMAT (' ERROR IN ',2A2,' FILE OPEN, STATUS = ',A2)
C*
C*  END OF FILE ON TRANSACTION INPUT
C*
999     CONTINUE
        DMSFLS(3) = 'CF'
        CALL DBMSYS ( DMSFLS (1), DMSFLS (15),
     1                DBDMY (1),  DBDMY (1),
     2                DBDMY (1),  DBDMY (1) )
        DMSFLS(5) = 'IT'
        DMSFLS(6) = 'EM'
        CALL DBMSYS ( DMSFLS (1), DMSFLS (15),
     1                DBDMY (1),  DBDMY (1),
     2                DBDMY (1),  DBDMY (1) )
        DMSFLS(5) = 'CU'
        DMSFLS(6) = 'ST'
        CALL DBMSYS ( DMSFLS (1), DMSFLS (15),
     1                DBDMY (1),  DBDMY (1),
     2                DBDMY (1),  DBDMY (1) )
        STOP
        END
```

Figure B-10. Listing of Program FEXMPN (Sheet 5 of 5)

```
NOSYMT
FORMAT IMAGE,REPLACE
LIBRARY .FORT78.OSLOBJ
LIBRARY .FORT78.STLOBJ
TASK FEXMPL
INCLUDE S$DBMS.TEST.FOBJECT
INCLUDE S$DBMS.SNDMSG
END
```

Figure B-11. Link Control File for Program FEXMPN

B.6 EXECUTION OF THE PASCAL PROGRAM, PEXMPL

Install and execute the Pascal program, PEXMPL, as follows:

1. Start the DBMS-990. If security is installed assign the password TEST to the data base files and assign the synonym $P to the value of the master password.

2. Initialize the data base files by executing the batch stream control file .S$DBMS.TEST.LOADFILN (use the XB command, followed by a WAIT command). When finished, the batch stream displays the following message:

   LUNO >## ASSIGNED TO THE PROGRAM FILE

3. Execute the batch stream .S$DBMS.TEST.PBATCHN to compile and install the program. This procedure uses the link control file .S$DBMS.TEST.PNLNK to link edit the program and install it on the file S$DBMS.TEST.PROG.

4. Check the batch stream listing files for errors. Ignore errors U SVC-0316 (file already exists) and U SVC-0315 (file does not exist) that occur while creating or deleting files.

5. Use the Assign Synonym (AS) command to assign synonyms to the input and output files, as follows:

Synonym	Value
PINP	.S$DBMS.TEST.TRAN
POUT	.S$DBMS.TEST.LIST.PRPT

6. Execute PEXMPL using the XPT command. Specify the program file .S$DBMS.TEST.PROG and the task name PEXMPL. Tab through the remaining prompts.

7. Display or print the file .S$DBMS.TEST.LIST.PRPT and compare it to the listing in Figure B-9. They should be the same.

8. After the program executes, you can halt DBMS-990 unless others are using the data base. Use the End DBMS (EDBMS) command.

Figure B-12 contains the output from the Pascal program PEXMPL. Figure B-13 contains the listing for PEXMPL. Figure B-14 contains the Link Editor control file for PEXMPL.

```
A001  ARMADILLOS              100123  0025  J80001  S0001  THING-A-MA-GIG CORP.
A001  ARMADILLOS              100123  2500  J80003  S0003  TOYS FOR TEXANS
A001  ARMADILLOS              100123  0025  J80004  S0004  THUNDERBOLT CO.
A001  ARMADILLOS              100123  0100  J80005  S0007  TURKEYS, INC.
A001  ARMADILLOS              100123  0100  J80008  S0088  *** NO SHIP NAME
A001  ARMADILLOS              100123  0025  J80009  S0001  THING-A-MA-GIG CORP.

B002  BLACK HOLES             020234  0007  J80002  S0002  WIDGITS, INC.
B002  BLACK HOLES             020234  0100  J80003  S0003  TOYS FOR TEXANS
B002  BLACK HOLES             020234  0001  J80005  S0007  TURKEYS, INC.

C003  CLAY                    003345  0500  J80004  S0004  THUNDERBOLT CO.

D004  DIPS                    000456  0005  J80001  S0001  THING-A-MA-GIG CORP.
D004  DIPS                    000456  0080  J80002  S0002  WIDGITS, INC.

E005  ERECTORS                005567  0050  J80001  S0001  THING-A-MA-GIG CORP.
E005  ERECTORS                005567  0055  J80005  S0007  TURKEYS, INC.

F006  FREEBIES                060678  0025  J80002  S0002  WIDGITS, INC.
F006  FREEBIES                060678  9999  J80003  S0003  TOYS FOR TEXANS

ABCD  *ITEM DOES NOT EXIST
ABCD  *ITEM DOES NOT EXIST                                  ***ITEM NOT SOLD

G007  GOOBERS                 700789  0033  J80001  S0001  THING-A-MA-GIG CORP.
G007  GOOBERS                 700789  0500  J80003  S0003  TOYS FOR TEXANS

H008  HERBS                   080890  0250  J80004  S0004  THUNDERBOLT CO.

I009  IDIOMS                  009900  0999  J80003  S0003  TOYS FOR TEXANS
I009  IDIOMS                  009900  0099  J80005  S0007  TURKEYS, INC.

J010  JUMPS                   001000  1050  J80002  S0002  WIDGITS, INC.

K011  KILNS                   011100  0010  J80008  S0088  *** NO SHIP NAME

L012  LONE STARS              120000  0099  J80002  S0002  WIDGITS, INC.
L012  LONE STARS              120000  9000  J80003  S0003  TOYS FOR TEXANS

N014  NIBBLES                 041400  0015  J80001  S0001  THING-A-MA-GIG CORP.
N014  NIBBLES                 041400  0250  J80003  S0003  TOYS FOR TEXANS
N014  NIBBLES                 041400  0999  J80004  S0004  THUNDERBOLT CO.

S019  SHOVELS                 101901  0100  J80002  S0002  WIDGITS, INC.

WXYZ  *ITEM DOES NOT EXIST
WXYZ  *ITEM DOES NOT EXIST                                  ***ITEM NOT SOLD

T020  TALES                   099099  0500  J80003  S0003  TOYS FOR TEXANS

Y025  YARNS                   025000  0250  J80003  S0003  TOYS FOR TEXANS
Y025  YARNS                   025000  0005  J80008  S0088  *** NO SHIP NAME

Z026  ZEBRAS                  002600  0002  J80004  S0004  THUNDERBOLT CO.
Z026  ZEBRAS                  002600  0002  J80008  S0088  *** NO SHIP NAME
```

Figure B-12. Output from the Pascal Program PEXMPL

```
PROGRAM PEXMPL;
(*****************************************************************
   THIS PROGRAM WILL EXTRACT DATA FROM THE DBMS SALES ORDER
   FILE ABOUT SPECIFIC ITEM NUMBERS READ FROM A SEQUENTIAL
   INPUT TRANSACTION FILE.  DATA THAT IS RETRIEVED FROM THE
   DATA BASE FILES IS OUTPUT TO A SEQUENTIAL FILE WHICH
   CAN BE DISPLAYED AFTER THE PROGRAM TERMINATES.

   THE ITEM DESCRIPTION AND UNIT PRICE ARE OBTAINED FROM
   THE ITEM FILE.  THE SALES-ORDER NUMBER, SHIP-TO CUSTOMER
   NUMBER AND QUANTITY ON-ORDER FOR EACH SALES ORDER THAT
   CONTAINS THE ITEM ARE OBTAINED FROM THE SOFL FILE.
   THE SHIP-TO CUSTOMER NAME IS RETEIEVED FROM THE CUST FILE.
   APPROPRIATE ERROR MESSAGES ARE PRINTED WHERE APPLICABLE.

      THE FOLLOWING SYNONYMS MUST BE DEFINED:
           PINP  = PASCAL INPUT FILE ACCESS NAME
           POUT  = PASCAL OUTPUT FILE ACCESS NAME
*****************************************************************)
CONST
   EM1 = 'ERROR IN ';
   EM2 = ' FILE OPEN, STATUS=';
(*  DEFINE DATA TYPES  *)
TYPE
   C2       = PACKED ARRAY [1..2] OF CHAR;
   C4       = PACKED ARRAY [1..4] OF CHAR;
   C6       = PACKED ARRAY [1..6] OF CHAR;
   C20      = PACKED ARRAY [1..20] OF CHAR;
   DA_TYPE = (SOF2, SOF3, CUST, ITEM);
(* DEFINE RECORD AREAS *)
DATAREA = RECORD
   CASE DA_TYPE OF
      SOF2 : (SHIP : C6);
      SOF3 : (QUAN : C4;  SONM : C6);
      CUST : (NAME : C20);
      ITEM : (DESC : C20; UPRC : C6);
END;
LINELIST = RECORD
   LL   : PACKED ARRAY [1..24] OF CHAR;
   TERM : INTEGER;
END;
CONTROLBLOCK = RECORD
   PSWD   : C4;
   FUNC   : C2;
   STAT   : C2;
   DBFILE : C4;
   LOC1   : C4;
   LOC2   : C4;
   KEYN   : C4;
   KEYV   : C6;
   TERM   : INTEGER;
END;
(* VARIABLE DEFINITIONS *)
VAR
   ERR                 : BOOLEAN;
   ITEMNO              : C6;
   SOFLPK_LL           : LINELIST;
   SOFLSK_LL           : LINELIST;
   CUST_LL             : LINELIST;
   ITEM_LL             : LINELIST;
```

Figure B-13. Listing of Program PEXMPL (Sheet 1 of 5)

```
   DA                    : RECORD
      DATA : DATAREA;
      TERM : INTEGER;
      END;
   CB                    : CONTROLBLOCK;
   POUT                  : TEXT;
   PINP                  : TEXT;
   ITEM_EXISTS           : BOOLEAN;
   MORE_ITEMS            : BOOLEAN;
   SHIP_EXISTS           : BOOLEAN;
   ITEM_SOLD             : BOOLEAN;
   SLOC1                 : C4;
   SLOC2                 : C4;
   ODESCRPT              : C20;
   OPRICE                : C6;
   OQTYOO                : C4;
   OSONO                 : C6;
   OSHIPNO               : C6;
   OSHIPNA               : C20;
(* DEFINE EXTERNAL  PROCEDURE TO CALL DBMS *)
PROCEDURE DBMSYS (VAR CB : CONTROLBLOCK; VAR CBE : INTEGER;
                  VAR LL : LINELIST; VAR LLE : INTEGER;
                  VAR DA : DATAREA; VAR DAE : INTEGER);
                  EXTERNAL FORTRAN;
(*  ERROR ROUTINE *)
PROCEDURE ERR_ROUTINE;
BEGIN
   WRITELN (POUT, 'DBERROR STAT=', CB.STAT, ', DBFILE=', CB.DBFILE,
            ', KEYN=', CB.KEYN, ', KEYV=', CB.KEYV);
   END;
(*  SET LOC1 AND LOC2 TO "*" TO START *)
PROCEDURE INIT_LOCS;
BEGIN
   CB.LOC1 := '****';
   CB.LOC2 := '****';
   END;
(* WRITE OUTPUT LINE *)
PROCEDURE WRITE_DATA;
BEGIN
   WRITELN(POUT, ITEMNO, ODESCRPT, '  ', OPRICE, '  ',
   OQTYOO, '  ', OSONO, '  ', OSHIPNO,  '  ' , OSHIPNA);
   END;
(* PROCEDURE TO GET CUSTOMER NAME FROM DBMS *)
(* CUSTOMER NAME IS RETRIEVED FROM THE CUSTOMER FILE "CUST" *)
PROCEDURE GET_NAME;
BEGIN
   CB.DBFILE := 'CUST';
   CB.KEYN := 'CUSN';
   CB.KEYV := DA.DATA.SHIP;
   INIT_LOCS;
   DBMSYS (CB, CB.TERM, CUST_LL, CUST_LL.TERM, DA.DATA, DA.TERM);
   IF CB.STAT = '**' AND CB.LOC1 <> '****'
      THEN OSHIPNA := DA.DATA.NAME
   ELSE
      IF CB.STAT = 'NK'
         THEN BEGIN
         OSHIPNA := '*** NO SHIP NAME    ';
         WRITE_DATA;
         END
      ELSE ERR_ROUTINE;
   END;
```

Figure B-13. Listing of Program PEXMPL (Sheet 2 of 5)

```
(*****   PROCEDURE TO GET THE SHIP-TO NAME FROM THE DBMS     *****)
(*                 SHIP-TO NUMBER IS LINE=02 OF SOFL FILE         *)
PROCEDURE GET_SHIP;
BEGIN
   CB.DBFILE := 'SOFL';
   CB.KEYN := 'SONM';
   CB.KEYV := DA.DATA.SONM;
   INIT_LOCS;
   DA.DATA.SHIP := '       ';
   DBMSYS (CB, CB.TERM, SOFLPK_LL, SOFLPK_LL.TERM, DA.DATA, DA.TERM);
   IF CB.STAT = '**' THEN
       IF CB.LOC1 <> '****'
          THEN BEGIN
          SHIP_EXISTS := TRUE;
          OSHIPNO := DA.DATA.SHIP;
          END
       ELSE BEGIN
          SHIP_EXISTS := FALSE;
          OSHIPNA := '***NO SHIP IN SOFL  ';
          WRITE_DATA;
          END
   ELSE ERR_ROUTINE;
   END;

(*****          GET SECONDARY KEY FROM SALES ORDER FILE           *****)
(*               SECONDARY KEY ITEM NUMBER IS RETRIEVED               *)
PROCEDURE GET_SOFL;
BEGIN
   CB.DBFILE := 'SOFL';
   CB.KEYN := 'ITEM';
   CB.KEYV := ITEMNO;
   IF MORE_ITEMS
      THEN BEGIN
      CB.LOC1 := SLOC1;
      CB.LOC2 := SLOC2;
      END
   ELSE INIT_LOCS;
   DBMSYS (CB, CB.TERM, SOFLSK_LL, SOFLSK_LL.TERM, DA.DATA, DA.TERM);
   IF CB.STAT ='**' THEN
      IF CB.LOC1 = '****'
         THEN MORE_ITEMS := FALSE
         ELSE BEGIN
            OQTYOO := DA.DATA.QUAN;
            OSONO  := DA.DATA.SONM;
            ITEM_SOLD := TRUE;
            SLOC1 := CB.LOC1;
            SLOC2 := CB.LOC2;
            END
   ELSE BEGIN
      MORE_ITEMS := FALSE;
      IF CB.STAT = 'NK'
         THEN  BEGIN
         OSHIPNA := '***ITEM NOT SOLD .  ';
         WRITE_DATA;
         END
      ELSE ERR_ROUTINE;
      END;
   END;
```

Figure B-13. Listing of Program PEXMPL (Sheet 3 of 5)

```
(*                    GET ITEM DESCRIPTION FROM ITEM FILE                    *)
PROCEDURE GET_ITEM;
BEGIN
   CB.DBFILE := 'ITEM';
   CB.KEYN := 'ITMN';
   CB.KEYV := ITEMNO;
   INIT_LOCS;
   DBMSYS (CB, CB.TERM, ITEM_LL, ITEM_LL.TERM, DA.DATA, DA.TERM);
   IF CB.STAT = '**' AND CB.LOC1 <> '****'
      THEN BEGIN
      ODESCRPT := DA.DATA.DESC;
      OPRICE := DA.DATA.UPRC;
      ITEM_EXISTS := TRUE;
      END
   ELSE
      IF CB.STAT = 'NK'
         THEN BEGIN
         ODESCRPT := '*ITEM DOES NOT EXIST';
         WRITE_DATA;
         END
      ELSE ERR_ROUTINE;
   END;
PROCEDURE PROC_ITEM;
BEGIN ( PROCESSES EACH ITEM TO SEE IF A SHIP-TO CUSTOMER EXISTS )
   GET_SHIP;
   IF SHIP_EXISTS THEN GET_NAME;
   IF CB.STAT = '**' AND SHIP_EXISTS THEN WRITE_DATA;
   MORE_ITEMS := TRUE;
   IF SLOC2 = '****'
      THEN MORE_ITEMS := FALSE
      ELSE GET_SOFL;
   END;
(*****   GET ITEMS FROM "ITEM" FILE AND PROCESS THE SECONDARY    *****
*****              KEY ON THE SALES ORDER FILE.                  *****)
PROCEDURE DBMS_ROUTINES;
BEGIN
   ITEM_EXISTS := FALSE;
   MORE_ITEMS := FALSE;
   ITEM_SOLD := FALSE;
   GET_ITEM;
   GET_SOFL;
   IF ITEM_EXISTS AND ITEM_SOLD
      THEN REPEAT PROC_ITEM UNTIL NOT MORE_ITEMS;
   END;
(*  PROCEDURE TO OPEN THE DATABASE FILES FOR FILE ACCESS CHECKING  *)
PROCEDURE OPEN_DATABASE_FILE(X:C4; VAR E:BOOLEAN);
BEGIN
   CB.DBFILE:=X;
   DBMSYS(CB,CB.TERM, CUST_LL,CUST_LL.TERM, DA.DATA,DA.TERM);
   IF CB.STAT <> '**'
      THEN BEGIN
      E:=TRUE;
      WRITELN(OUTPUT, EM1, X, EM2, CB.STAT);
      RESET(INPUT);
      END
   ELSE E:=FALSE;
   END;
```

Figure B-13. Listing of Program PEXMPL (Sheet 4 of 5)

```
PROCEDURE INITIALIZATION;
BEGIN
{        *** INITIALIZE REMAINING AREAS AND PROCESS DATA    ***
}
   SOFLPK_LL.LL := 'LINE=02*SHIP****RLSE     ';
   SOFLSK_LL.LL := 'LINE=03*QUANSONM****RLSE';
   CUST_LL.LL := 'LINE=01*NAME****RLSE     ';
   ITEM_LL.LL := 'LINE=01*DESCUPRC****RLSE';
   CB.PSWD := 'TEST';
{
       ***  INITIALIZE THE CONTROL BLOCK TO OPEN THE DATABASE FILES ***
}
   CB.FUNC := 'OF';
   CB.KEYN := 'SHRD';
   OPEN_DATABASE_FILE('CUST',ERR);IF ERR THEN ESCAPE INITIALIZATION;
   OPEN_DATABASE_FILE('ITEM',ERR);IF ERR THEN ESCAPE INITIALIZATION;
   OPEN_DATABASE_FILE('SOFL',ERR);IF ERR THEN ESCAPE INITIALIZATION;
   CB.FUNC := 'RF';
   CB.KEYN := 'SOMN';
   INIT_LOCS;
   END;
(* MAIN PROCESSING CYCLE *)
BEGIN
   RESET(PINP);
   REWRITE(POUT);
   INITIALIZATION;
   IF NOT ERR
      THEN REPEAT BEGIN
         READLN(PINP, ITEMNO);
         INIT_LOCS;
         ODESCRPT := '                          ';
         OPRICE := '       ';
         OQTYOO := '     ';
         OSONO := '       ';
         OSHIPNO := '       ';
         OSHIPNA := '                          ';
         DBMS_ROUTINES;
         WRITELN(POUT);
         END;
      UNTIL EOF(PINP);
   CLOSE (PINP);
   CLOSE (POUT);
END.
```

Figure B-13. Listing of Program PEXMPL (Sheet 5 of 5)

```
NOSYMT
FORMAT IMAGE,REPLACE
LIBRARY S$TIP.OBJ
TASK PEXMPL
INCLUDE S$TIP.OBJ.MAIN
INCLUDE S$DBMS.TEST.POBJECT
INCLUDE S$DBMS.SNDMSG
INCLUDE S$DBMS.FRGMY
END
```

Figure B-14. Link Control File for Program PEXMPL

B.7 EXECUTION OF THE COBOL PROGRAM, CEXTRN

The following program demonstrates the use of transaction boundaries for transaction-level integrity.

This program adds a new item to an existing sales order in the sales-order file.

It reads from a sequential input file. The data on the file are the sales order number, the item number to add, and the quantity desired.

The sales order number, customer name, item number and description, quantity, unit price, and total cost are printed on the output file. When items are not in stock, a back order message is printed and items are not deducted from the stock.

Install and execute the COBOL program, CEXTRN, as follows:

1. Start the DBMS-990. If security is installed assign the password TEST to the data base files and assign the synonym $P to the value of the master password.

2. Initialize the data base files by executing the batch stream control file .S$DBMS.TEST.LOADFILN (use the XB command, followed by a WAIT command). When finished, the batch stream displays the following message:

   LUNO >## ASSIGNED TO THE PROGRAM FILE

   NOTE

   Steps 1 and 2 must be completed before continuing with the execution of the COBOL program CEXTRN.

3. Execute the batch stream .S$DBMS.TEST.CBATCHTR to compile and install the program.

4. Check the batch stream listing files for errors. Ignore errors U SVC-0316 (file already exists) and U SVC-0315 (file does not exist) that occur while creating or deleting files.

5. Use the Assign Synonym (AS) command to assign synonyms to the input and output files, as follows:

Synonym	Value
CINP	.S$DBMS.TEST.UPDATE
COUT	.S$DBMS.TEST.LIST.CRPTUPD

6. Execute CEXTRN using the Execute COBOL Task Foreground command (XCTF). For the first parameter, enter the program file. Then enter CEXTRN as the task name and tab through the remaining prompts. The program executes for about one minute.

7. Display or print the file .S$DBMS.TEST.LIST.CRPTUPD and compare it to the listing in Figure B-15. They should be the same.

8. After the program executes, you can halt DBMS-990 unless others are using the data base. Use the End DBMS (EDBMS) command.

Figure B-16 contains the source listing for CEXTRN. Figure B-17 contains the Link Editor control file for CEXMPL.

```
J80001 THING-A-MA-GIG CORP.   B002 BLACK HOLES            0300   $20.234     $6070.20
                                   *** BACK-ORDERED ***

J80001 THING-A-MA-GIG CORP.   F006 FREEBIES               0300   $60.678    $18203.40

J80002 WIDGITS, INC.          A001 ARMADILLOS             0300  $100.123    $30036.90
                                   *** BACK-ORDERED ***

J80003 TOYS FOR TEXANS        C003 CLAY                   0300    $3.345     $1003.50

J80003 TOYS FOR TEXANS        D004 DIPS                   0300     $.456      $136.80

J80003 TOYS FOR TEXANS        E005 ERECTORS               0300    $5.567     $1670.10

J80004 THUNDERBOLT CO.        G007 GOOBERS                0300  $700.789   $210236.70

J80005 TURKEYS, INC.          H008 HERBS                  0300   $80.890    $24267.00

J80005 TURKEYS, INC.          I009 IDIOMS                 0300    $9.900     $2970.00

J80006 ODD JOB WYRKERS        J010 JUMPS                  0300    $1.000      $300.00

J80006 ODD JOB WYRKERS        K011 KILNS                  0300   $11.100     $3330.00

J80006 ODD JOB WYRKERS        L012 LONE STARS             0300  $120.000    $36000.00
```

Figure B-15. Output from the COBOL Program CEXTRN

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  CEXTRN.
 AUTHOR.  TEXAS INSTRUMENTS, INC.
 DATE-WRITTEN.  FEBRUARY, 1982.
*
*     THIS PROGRAM WILL ADD A NEW ITEM TO AN EXISTING
*     SALES ORDER IN THE SALES-ORDER FILE.
*
*     IT WILL READ FROM A SEQUENTIAL INPUT FILE.   THE
*     DATA ON THE FILE WILL BE THE SALES ORDER NUMBER,
*     THE ITEM NUMBER TO ADD, AND THE QUANTITY DESIRED.
*
*     THE SALES ORDER NUMBER, CUSTOMER NAME, ITEM NUMBER
*     AND DESCRIPTION, QUANTITY, UNIT PRICE, AND TOTAL
*     COST WILL BE PRINTED ON THE OUTPUT FILE.   IF NOT
*     ENOUGH IN STOCK TO FILL THE ORDER, A "BACK ORDER"
*     MESSAGE WILL BE PRINTED AND NO ITEMS WILL BE
*     DEDUCTED FROM THE STOCK.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  TI-990.
 OBJECT-COMPUTER.  TI-990.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL..
*
*     A SEQUENTIAL FILE CONTAINING THE SALES ORDER,
*     ITEM NUMBER, AND QUANTITY TO BE ADDED MUST HAVE
*     THE SYNONYM "CINP" ASSIGNED TO IT BEFORE THIS
*     PROGRAM CAN BE EXECUTED.
*
 SELECT TINFILE
       ASSIGN TO INPUT, "CINP";
       ORGANIZATION IS SEQUENTIAL;
       ACCESS IS SEQUENTIAL.
*
*     A SEQUENTIAL FILE FOR OUTPUT MUST HAVE
*     THE SYNONYM "COUT" ASSIGNED TO IT BEFORE
*     THIS PROGRAM CAN BE EXECUTED.
*
 SELECT OUTFILE
       ASSIGN TO OUTPUT, "COUT";
       ORGANIZATION IS SEQUENTIAL;
       ACCESS IS SEQUENTIAL.
 DATA DIVISION.
 FILE SECTION
 FD TINFILE;
       LABEL RECORDS ARE OMITTED;
       DATA RECORD IS TINREC.
 01   TINREC.
      05  SALENO     PIC X(6).
      05  ITEMNO     PIC X(4).
      05  QUANT      PIC 9(4).
      05  FILLER     PIC X(65).
 FD OUTFILE;
       LABEL RECORDS ARE OMITTED;
       DATA RECORD IS OUTREC.
 01   OUTREC.
      05  FILLER.   PIC X(80).
 WORKING-STORAGE SECTION.
*
*     WORK AREAS AND INDICATORS
```

Figure B-16. Listing of Program CEXTRN (Sheet 1 of 6)

```
*
 77  TOTCOST       PIC 9(7)V9(2).
 77  DL-COUNT      PIC 9(2).
 01  EOF-IND       PIC X VALUE "N".
     88  TRAN-EOF VALUE "Y".
 01  BACK-IND      PIC X VALUE SPACES.
     88  BACKORDER-NEEDED VALUE "Y".
 01  TR-IND        PIC X  VALUE SPACES.
     88  TR-NEEDED VALUE "Y".
 01  DATAREC.
     05  OSONO     PIC X(6).
     05  FILLER    PIC X(1).
     05  OSHIPNA   PIC X(20).
     05  FILLER    PIC X(2).
     05  OITEMNO   PIC X(4).
     05  FILLER    PIC X(1).
     05  ODESCRPT  PIC X(20).
     05  FILLER    PIC X(1).
     05  OQTYOO    PIC X(4).
     05  FILLER    PIC X(1).
     05  OPRICE    PIC $$$$.999.
     05  FILLER    PIC X(1).
     05  OTOTALO   PIC $$$$$$$$.99.
 01  ERRREC.
     05  EMSG      PIC X(8) VALUE "DBERROR ".
     05  FILLER    PIC X(5) VALUE "STAT=".
     05  ESTAT     PIC X(2) VALUE SPACES.
     05  FILLER    PIC X(9) VALUE ", DBFILE=".
     05  EFILE     PIC X(4) VALUE SPACES.
     05  FILLER    PIC X(7) VALUE ", KEYN=".
     05  EKEYN     PIC X(4) VALUE SPACES.
     05  FILLER    PIC X(7) VALUE ", KEYV=".
     05  EKEYV     PIC X(6) VALUE SPACES.
 01 ERROR-MSG.
     10  FILLER    PIC X(9) VALUE "ERROR IN ".
     10  ERR-FILE PIC X(4).
     10  FILLER    PIC X(19) VALUE " FILE OPEN, STATUS=".
     10  ERR-STAT PIC XX.
*
*    DBMS DML CALL PARAMETER AREAS
*
*    IF SECURITY IS INSTALLED ON YOUR DBMS, THE VALUE OF
*    PSWD DATA ITEM IN THE CONTROL BLOCK MUST BE CHANGED
*    TO THE PASSWORD THAT WILL BE ASSIGNED TO THE SOFL,
*    CUST AND ITEM DATA BASE FILES. NOTE: SINCE THERE
*    IS ONLY ONE CONTROL BLOCK IN THIS PROGRAM ALL THREE
*    FILES SHOULD HAVE THE SAME PASSWORD.
*
*        DUMMY ADDRESSES USED WITH THE FILE ACCESS CHECKING
*        FOR OPEN AND CLOSE DATABASE FILE FUNCTIONS.
 01  D1                PIC X.
 01  D2                PIC X.
*        CONTROL BLOCK
 01  CB.
     02 PSWD           PIC X(4)  VALUE  "TEST".
     02 FUNC           PIC XX    VALUE  "OF".
     02 STAT           PIC XX    VALUE  "**".
     02 DBFILE         PIC X(4)  VALUE  "    ".
     02 LOC1           PIC X(4)  VALUE  "****".
     02 LOC2           PIC X(4)  VALUE  "****".
     02 KEYN           PIC X(4)  VALUE  "SHRD".
     02 KEYV           PIC X(6).
```

Figure B-16. Listing of Program CEXTRN (Sheet 2 of 6)

```
*       SALES ORDER FILE PRIMARY KEY READ LINE LIST
 01  SOFLPK-LL.
     02 SPLTYPE       PIC X(7)      VALUE  "LINE=02".
     02 SPRETIND      PIC X         VALUE  "*".
     02 SPFIELDS      PIC X(4)      VALUE  "SHIP".
     02 SPDISP        PIC X(8)      VALUE  "****RLSE".
*       SALES ORDER FILE PRIMARY KEY ADD LINE LIST
 01  SOFL03-LL.
     02 SPLTYPE       PIC X(7)      VALUE  "LINE=03".
     02 SPRETIND      PIC X         VALUE  "*".
     02 SPFIELDS      PIC X(8)      VALUE  "ITEMQUAN".
     02 SPDISP        PIC X(8)      VALUE  "****RLSE".
*        CUSTOMER FILE LINE LIST
 01  CUST-LL.
     02 CLTYPE       PIC X(7)      VALUE  "LINE=01".
     02 CRETIND      PIC X         VALUE  "*".
     02 CFIELDS      PIC X(4)      VALUE  "NAME".
     02 CDISP        PIC X(8)      VALUE  "****RLSE".
*         ITEM FILE LINE LIST
 01  ITEM-LL.
     02 ILTYPE       PIC X(7)      VALUE  "LINE=01".
     02 IRETIND      PIC X         VALUE  "*".
     02 IFIELDS      PIC X(16)     VALUE  "DESCUPRCQTYOQTYH".
     02 IDISP        PIC X(8)      VALUE  "****HOLD".
*        DBMS FILE DATA AREAS
 01  DA.
     02 FILLER      PIC X(34).
 01  SOFL02-DA REDEFINES DA.
     02 SHIP        PIC X(5).
     02 FILLER      PIC X(29).
 01  SOFL03-DA REDEFINES DA.
     02 ITEM        PIC X(4).
     02 QUAN        PIC 9(4).
     02 FILLER      PIC X(26).
 01  CUST-DA REDEFINES DA.
     02 NAME        PIC X(20).
     02 FILLER      PIC X(14).
 01  ITEM-DA REDEFINES DA.
     02 DESC        PIC X(20).
     02 UPRC        PIC 9(3)V9(3).
     02 QTYO        PIC 9(4).
     02 QTYH        PIC 9(4).
 01  DB-DELIM             PIC XX    VALUE "/*".
/
 PROCEDURE DIVISION.
 MAIN-PROG.
     OPEN INPUT TINFILE, OUTPUT OUTFILE.
     MOVE SPACES TO OUTREC DATAREC.
     MOVE "OF" TO FUNC. MOVE "SHRD" TO KEYN.
     MOVE "CUST" TO DBFILE.
     PERFORM OPEN-DATABASE-FILE.
     MOVE "ITEM" TO DBFILE.
     PERFORM OPEN-DATABASE-FILE.
     MOVE "SOFL" TO DBFILE.
     PERFORM OPEN-DATABASE-FILE.
     PERFORM PROCESS-TRAN UNTIL TRAN-EOF.
 END-OF-RUN.
     MOVE "CF" TO FUNC.
     MOVE "CUST" TO DBFILE.
     CALL "DBMSYS" USING CB, SOFLPK-LL, D1, D2, D1, D2.
     MOVE "ITEM" TO DBFILE.
     CALL "DBMSYS" USING CB, SOFLPK-LL, D1, D2, D1, D2.
```

Figure B-16. Listing of Program CEXTRN (Sheet 3 of 6)

```
     MOVE "SOFL" TO DBFILE.
     CALL "DBMSYS" USING CB, SOFLPK-LL, D1, D2, D1, D2.
     CLOSE TINFILE OUTFILE.
     STOP RUN.
/
 OPEN-DATABASE-FILE.
     CALL "DBMSYS" USING CB, SOFLPK-LL, D1, D2, D1, D2.
     IF STAT NOT = "**"
         MOVE DBFILE TO ERR-FILE
         MOVE STAT TO ERR-STAT
         DISPLAY ERROR-MSG LINE 24
         ACCEPT D1 LINE 24 PROMPT
         GO TO END-OF-RUN.
 START-TRANSACTION.
     MOVE "TS" TO FUNC.
     PERFORM TRANSACTION-CALL.
 COMMIT-TRANSACTION.
     MOVE "TC" TO FUNC
     PERFORM TRANSACTION-CALL.
 ROLLBACK-TRANSACTION.
     MOVE "TR" TO FUNC
     PERFORM TRANSACTION-CALL.
 TRANSACTION-CALL.
     CALL "DBMSYS" USING CB SOFLPK-LL, D1 D2, D1 D2.
 INIT-LOCS.
     MOVE "****" TO LOC1 LOC2.
 CHECK-STAT.
     IF STAT IS NOT EQUAL TO "**"
         MOVE "Y" TO TR-IND
         PERFORM ERR-ROUTINE.
 READ-TRAN.
     READ TINFILE RECORD
         AT END MOVE "Y" TO EOF-IND.
     IF NOT TRAN-EOF
         MOVE "****" TO LOC1 LOC2
         MOVE SALENO TO OSONO
         MOVE ITEMNO TO KEYV OITEMNO
         MOVE QUANT TO OQTYOO.
 OUTPUT-INFO.
     MOVE DATAREC TO OUTREC.
     PERFORM WRITE-DATA.
 WRITE-DATA.
     WRITE OUTREC.
/
 PROCESS-TRAN.
     PERFORM READ-TRAN.
     IF NOT TRAN-EOF
         MOVE 0 TO DL-COUNT
         PERFORM DBMS-ROUTINES
         MOVE SPACES TO TINREC DATAREC ESTAT EFILE EKEYN
                        EKEYV BACK-IND TR-IND.
     PERFORM OUTPUT-INFO.
 DBMS-ROUTINES.
     PERFORM START-TRANSACTION.
     PERFORM GET-ITEM-FROM-ITEM.
     IF TR-NEEDED
         PERFORM ROLLBACK-TRANSACTION
       ELSE
         PERFORM GET-SHIP-FROM-SOFL
         IF TR-NEEDED
             PERFORM ROLLBACK-TRANSACTION
           ELSE
```

Figure B-16. Listing of Program CEXTRN (Sheet 4 of 6)

```
                PERFORM COMMIT-TRANSACTION
                IF STAT IS EQUAL TO "DL"
                    IF DL-COUNT IS LESS THAN 10
                        ADD 1 TO DL-COUNT
                        GO TO DBMS-ROUTINES
                      ELSE
                        PERFORM ERR-ROUTINE
                  ELSE
                    PERFORM GET-NAME-FROM-CUST
                    PERFORM OUTPUT-INFO
                    IF BACKORDER-NEEDED
                        MOVE SPACES TO DATAREC
                        MOVE "*** BACK-ORDERED ***" TO ODESCRPT
                        PERFORM OUTPUT-INFO.
GET-ITEM-FROM-ITEM.
    MOVE "RF" TO FUNC.
    MOVE "ITEM" TO DBFILE.
    MOVE "ITMN" TO KEYN.
    MOVE ITEMNO TO KEYV.
    PERFORM INIT-LOCS.
    CALL "DBMSYS" USING CB SOFLPK-LL, ITEM-LL DA,
                        DA DB-DELIM.
    IF STAT IS EQUAL TO "**"
            AND LOC1 NOT EQUAL "****"
        MOVE DESC TO ODESCRPT
        MOVE UPRC TO OPRICE
        MULTIPLY UPRC BY QUANT GIVING TOTCOST ROUNDED
        MOVE TOTCOST TO OTOTALO
        PERFORM UPDATE-ITEM
      ELSE
        MOVE "Y" TO TR-IND
        IF STAT EQUAL TO "NK"
            MOVE "*ITEM DOES NOT EXIST" TO ODESCRPT
            PERFORM OUTPUT-INFO
          ELSE
            PERFORM ERR-ROUTINE.
UPDATE-ITEM.
    ADD QUANT TO QTYO.
    IF QTYH IS LESS THAN QUANT
        MOVE "Y" TO BACK-IND
      ELSE
        SUBTRACT QUANT FROM QTYH.
    MOVE "WT" TO FUNC
    CALL "DBMSYS" USING CB SOFLPK-LL, ITEM-LL DA,
                        DA DB-DELIM.
    PERFORM CHECK-STAT.
GET-SHIP-FROM-SOFL.
    MOVE "RF" TO FUNC.
    MOVE "SOFL" TO DBFILE.
    MOVE "SONM" TO KEYN.
    MOVE SALENO TO KEYV.
    PERFORM INIT-LOCS.
    CALL "DBMSYS" USING CB SOFLPK-LL, SOFLPK-LL SOFL03-LL,
                        DA DB-DELIM.
    IF STAT IS EQUAL TO "**"
        IF LOC1 EQUAL "****"
            MOVE "***NO SHIP IN SOFL " TO OSHIPNA
            PERFORM UPDATE-SO
          ELSE
            MOVE SHIP TO OSHIPNA
            PERFORM UPDATE-SO
      ELSE
```

Figure B-16. Listing of Program CEXTRN (Sheet 5 of 6)

```
           MOVE "Y" TO TR-IND
           IF STAT EQUAL TO "NK"
               MOVE "***SO DOES NOT EXIST" TO OSHIPNA
               PERFORM OUTPUT-INFO
             ELSE
               PERFORM ERR-ROUTINE.
UPDATE-SO.
     MOVE ITEMNO TO ITEM.
     MOVE QUANT TO QUAN.
     PERFORM INIT-LOCS.
     MOVE "AA" TO FUNC.
     CALL "DBMSYS" USING CB SOFLPK-LL, SOFL03-LL CUST-LL,
                         DA DB-DELIM.
     PERFORM CHECK-STAT.
GET-NAME-FROM-CUST.
     MOVE "RF" TO FUNC.
     MOVE "CUST" TO DBFILE.
     MOVE "CUSN" TO KEYN.
     MOVE OSHIPNA TO KEYV.
     PERFORM INIT-LOCS.
     CALL "DBMSYS" USING CB SOFLPK-LL, CUST-LL ITEM-LL,
                          DA DB-DELIM.
     IF STAT IS EQUAL TO "**"
             AND LOC1 NOT EQUAL "****"
         MOVE NAME TO OSHIPNA
       ELSE
         IF STAT EQUAL TO "NK"
             MOVE "***NO SHIP NAME " TO OSHIPNA
             PERFORM OUTPUT-INFO
           ELSE
             PERFORM ERR-ROUTINE.
ERR-ROUTINE.
     MOVE DBFILE TO EFILE.
     MOVE KEYN TO EKEYN.
     MOVE KEYV TO EKEYV.
     MOVE STAT TO ESTAT.
     MOVE ERRREC TO OUTREC.
     PERFORM WRITE-DATA.
END PROGRAM.
```

Figure B-16. Listing of Program CEXTRN (Sheet 6 of 6)

```
FORMAT IMAGE,REPLACE
PROC DBINFACE
DUMMY
INCLUDE .DBLIB.DBINFACE
TASK CEXTRN
INCLUDE .S$SYSLIB.RCBTSK
INCLUDE .S$SYSLIB.RCBMPD
INCLUDE .DBLIB.TEST.COBJTRN
INCLUDE .DBLIB.SNDMSG
INCLUDE .DBLIB.COBINT
END
```

Figure B-17. Link Control File for Program CEXTRN

Alphabetical Index

Introduction

HOW TO USE INDEX

The index, table of contents, list of illustrations, and list of tables are used in conjunction to obtain the location of the desired subject. Once the subject or topic has been located in the index, use the appropriate paragraph number, figure number, or table number to obtain the corresponding page number from the table of contents, list of illustrations, or list of tables.

INDEX ENTRIES

The following index lists key words and concepts from the subject material of the manual together with the area(s) in the manual that supply major coverage of the listed concept. The numbers along the right side of the listing reference the following manual areas:

- Sections — Reference to Sections of the manual appear as "Sections x" with the symbol x representing any numeric quantity.
- Appendixes — Reference to Appendixes of the manual appear as "Appendix y" with the symbol y representing any capital letter.
- Paragraphs — Reference to paragraphs of the manual appear as a series of alphanumeric or numeric characters punctuated with decimal points. Only the first character of the string may be a letter; all subsequent characters are numbers. The first character refers to the section or appendix of the manual in which the paragraph may be found.
- Tables — References to tables in the manual are represented by the capital letter T followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the table). The second character is followed by a dash (-) and a number.

  Tx-yy

- Figures — References to figures in the manual are represented by the capital letter F followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the figure). The second character is followed by a dash (-) and a number.

  Fx-yy

- Other entries in the Index — References to other entries in the index preceded by the word "See" followed by the referenced entry.

CUT ALONG LINE

USER'S RESPONSE SHEET

Manual Title: Model 990 Computer DNOS Data Base Management System Programmer's Guide (2272058-9701)

Manual Date: 15 July 1982 **Date of This Letter:** _____

User's Name: _____ **Telephone:** _____

Company: _____ **Office/Department:** _____

Street Address: _____

City/State/Zip Code: _____

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. If there are any other suggestions that you wish to make, feel free to include them. Thank you.

Location in Manual	Comment/Suggestion
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), TAPE AND MAIL

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7284 DALLAS, TX

POSTAGE WILL BE PAID BY ADDRESSEE

TEXAS INSTRUMENTS INCORPORATED

DIGITAL SYSTEMS GROUP

ATTN: TECHNICAL PUBLICATIONS
P.O. Box 2909 M/S 2146
Austin, Texas 78769

FOLD

TEXAS INSTRUMENTS
INCORPORATED

DIGITAL SYSTEMS GROUP
P.O. BOX 2909 • AUSTIN, TEXAS 78769