60485300

CONTROL DATA

DMS-170

CYBER DATABASE CONTROL SYSTEM VERSION 2 APPLICATION PROGRAMMING REFERENCE MANUAL

CDC® OPERATING SYSTEMS:
NOS 2
NOS/BE 1

REVISION RECORD

Revision	Description
A (05/14/82)	Initial release under NOS 2 and NOS/BE 1; PSR level 564. This manual represents a complete reorganization of documentation of the COBOL 5, FORTRAN, and Query Update application programming interfaces to CDCS. This manual supersedes the FORTRAN Data Base Facility (FDBF) 1 reference manual (Pub. No. 60482200). This manual also documents new features of CDCS (namely, data base transactions, data base versions, and the immediate return option) at product levels as follows: CDCS 2.3, COBOL 5.3, FDBF 1.3, Query Update 3.4.
B (08/26/83)	Released at PSR level 587. This revision documents improved duration loading capabilities of CDCS and miscellaneous technical corrections.
C (02/20/84)	Released at PSR level 599. This revision documents FORTRAN interface support of concatenated keys and miscellaneous technical corrections.
D (04/23/86)	Released at PSR level 647. This revision removes references to FORTRAN 4.

REVISION LETTERS I, O, Q, AND X ARE NOT USED

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P. O. BOX 3492
SUNNYVALE, CALIFORNIA 94088-3492

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Front Cover	-
Title Page	-
ii	D
iii/iv	D
v	D
vi	A
vii	D
viii	D
ix/x	D
xi	A
1-1	A
1-2	A
1-3	D
1-4 thru 1-8	A
2-1 thru 2-11	A
2-12	D
2-13 thru 2-22	A
3-1	D
3-2 thru 3-4	C
3-5	D
3-6	A
3-7	A
3-8 thru 3-18	D
3-18.1	D
3-18.2	D
3-19	D
3-20 thru 3-26	A
4-1 thru 4-16	A
5-1 thru 5-22	A
6-1	B
6-2 thru 6-5	A
7-1 thru 7-6	D
8-1 thru 8-4	A
A-1 thru A-4	A
B-1	A
B-2	A
B-3	D
B-4	D
B-4.1/B-4.2	C
B-5	A
B-6	B
B-7	A
B-8 thru B-12	D
B-13	A
B-14 thru B-16	D
B-16.1/B-16.2	C
B-17 thru B-22	A
B-23	B
B-24	C
B-24.1/B-24.2	C
B-25 thru B-29	A
B-30	D
B-31	C
C-1	A
C-2	A
C-3	B
C-4	A
C-5	B
C-6 thru C-9	A
D-1 thru D-3	A
D-4 thru D-7	D
E-1 thru E-3	A
F-1	A
G-1	B
G-2	C
G-3	B
G-4	A
H-1	D
H-2 thru H-4	A
H-5 thru H-12	D
I-1	A
I-2	A
J-1	A
J-2	D
J-3	D
Index-1 thru -3	D
Index-4	B
Index-5	D
Comment Sheet/Mailer	D
Back Cover	-

PREFACE

This manual describes application program interfaces to CONTROL DATA® CYBER Database Control System (CDCS) Version 2. CDCS is one of the principal components of the DMS-170 data management system. The primary function of CDCS is to control access by an application program to a data base. For an application program to interface with CDCS, it must be coded in one of the following programming languages: COBOL Version 5, FORTRAN Version 5, or Query Update Version 3. FORTRAN programs must also be coded with Data Manipulation Language (DML) statements which are provided by the FORTRAN Data Base Facility (FDBF) Version 1.

The products that make up the CDCS-application program interface are CDCS 2.3, FDBF 1.3, COBOL 5.3,and Query Update 3.4. As described in this manual, these products operate under control of the following operating systems:

NOS 2 for the CONTROL DATA CYBER 180 Computer System; CYBER 170 Computer System; CYBER 70 Computer System Models 71, 72, 73, and 74; and 6000 Computer Systems

NOS/BE 1 for the CDC® CYBER 180 Computer System; CYBER 170 Computer System; CYBER 70 Computer System Models 71, 72, 73, and 74; and 6000 Computer Systems

This manual is designed for use by application programmers. The programmers are assumed to be experienced programmers who are familiar with the COBOL or FORTRAN application programming languages or the Query Update processing language. The programmers are also assumed to be familiar with data management concepts and with Control Data computers and software products.

Detailed information for the application programmers about subschemas is contained in the CDCS 2 Data Administration reference manual. Related material is contained in the publications listed below.

The NOS Manual Abstracts and the NOS/BE Manual Abstracts are instant-sized manuals containing brief descriptions of the contents and intended audience of all NOS and NOS product set manuals, and NOS/BE and NOS/BE product set manuals, respectively. The abstracts manuals can be useful in determining which manuals are of greatest interest to a particular user. The Software Publications Release History serves as a guide in determining which revision level of software documentation corresponds to the Programming Systems Report (PSR) level of installed site software.

The publications are listed alphabetically in groupings that indicate relative importance to readers of this manual.

The following manuals are of primary interest:

Publication	Publication Number
COBOL Version 5 Reference Manual	60497100
CYBER Record Manager Advanced Access Methods Reference Manual	60499300
DMS-170 CYBER Database Control System Version 2 Data Administration Reference Manual	60485200
DMS-170 CYBER Database Control System Version 2 FORTRAN Application Programming User Guide	60483500
FORTRAN Version 5 Reference Manual	60481300
Query Update Version 3 Reference Manual	60498300

The following manuals are of secondary interest:

Publication	Publication Number
Network Products Transaction Facility Version 1 Reference Manual	60459500
NOS Version 2 Manual Abstracts	60485500
NOS Version 2 Reference Set, Volume 3 System Commands	60459680
NOS/BE Version 1 Manual Abstracts	84000470
NOS/BE Version 1 Reference Manual	60493800
Software Publications Release History	60481000

CDC manuals can be ordered from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

NOTATIONS

Reference formats are presented throughout the manual to illustrate essential elements of syntax. The notations used in the reference formats follow two conventions: the COBOL convention and the FORTRAN convention. The COBOL convention is used in formats that describe syntax for COBOL statements and Query Update directives. The FORTRAN convention is used in formats that describe syntax for the FORTRAN Data Manipulation Language (DML) statements and for all control statements. The differences in the conventions are in the interpretation of uppercase words, the omission of underlined uppercase words from the FORTRAN convention, and the use of punctuation.

The notations used in reference formats are described as follows:

UPPERCASE	COBOL convention. Uppercase words are reserved words and must appear exactly as shown. Reserved words can be used only as specified in the reference formats. If not underlined, they are optional.
	FORTRAN convention. Uppercase words are keywords and must appear exactly as shown. Keywords can be used only as specified in the reference formats.
UNDERLINED UPPERCASE	COBOL convention. Underlined uppercase words are required when the format in which they appear is used.
	FORTRAN convention. Underlined uppercase words are not used.
Lowercase	Lowercase words are generic terms that represent the words or symbols supplied by the user. When generic terms are repeated in a format, a number is appended to the term for identification.
[ ]	Brackets enclose optional portions of a reference format. All of the format within the brackets can be omitted or included at the user's option. If items are stacked vertically within brackets, only one of the stacked items can be used.
{ }	Braces enclose one item or several vertically stacked items in a reference format. When one item is enclosed in braces and followed by ellipses, the item can be repeated at the user's option. When several items are enclosed in braces, only one of the enclosed items must be used.
|| ||	Vertical bars enclose two or more vertically stacked items in a reference format when at least one of the enclosed items must be used. Each of the vertically stacked items can be used once.
[|| ||]	Vertical bars within brackets enclose two or more vertically stacked items in a reference format when each of the stacked items can be used once or omitted. Any items can be used in any order.
...	Ellipses immediately follow a pair of brackets or braces to indicate that the enclosed material can be repeated at the user's option.

Punctuation use differs for the conventions as follows:

COBOL convention

Punctuation symbols shown within the formats are required unless enclosed in brackets and specifically noted as optional. In general, commas and semicolons are optional. One or more spaces separate the elements in a reference format.

FORTRAN convention

Punctuation symbols shown within formats are required unless enclosed in brackets and specifically noted as optional.

Numbers shown within formats are decimal unless otherwise specified.

INTRODUCTION TO DATA BASE PROCESSING WITH DMS-170

1

DMS-170 is a data management system for Control Data computer systems. Through this data management system, a data base can be defined, maintained, and controlled in an environment totally independent of the applications that are accessing it. Conventional files otherwise owned and processed by a number of distinct applications can be described through the data description language facilities of DMS-170. Consequently, the responsibility for tasks such as data description, data conversion, and validity checking is transferred from the application programmer to the data administrator.

The DMS-170 data management system is composed of the following elements:

Data Description Language (DDL), which creates the schema definition, as well as the COBOL, FORTRAN and Query Update subschema definitions.

CDC CYBER Database Control System (CDCS), which controls, monitors, and interprets data base requests from COBOL, FORTRAN, and Query Update application programs.

CDC CYBER Record Manager (CRM), which handles all input/output processing requests on a data base from an application program.

Data Manipulation Languages (DML), which provide for data base access through the COBOL and FORTRAN programming languages. The COBOL DML consists of features within the COBOL language. The FORTRAN DML consists of DML statements that are used within a FORTRAN-coded program and preprocessed by a DML preprocessor before FORTRAN compilation.

Query Update language, which provides for data base access in both interactive and batch modes. Query Update is a language that enables individuals with varying levels of technical expertise to access and manipulate the data base and to produce special-purpose reports.

Each element of the DMS-170 system is used either in the definition or in the processing of a data base. The definition of the data base is accomplished through the capabilities of DDL and CDCS. Processing of the data base involves retrieval and updating of the data by application programs through the facilities of CDCS.

The DMS-170 environment defines two roles: the data administrator and the application programmer. The data administrator is responsible for the definition of a data base. The data administrator is a person or group of persons who develop and define the data base as well as monitor and control the day-to-day processing of that data base. The application programmer uses the interface capabilities of COBOL, FORTRAN, and Query Update to develop programs for data base processing. The application programs use the detailed descriptions and facilities of CDCS when performing data base processing.

CDC offers guidelines for the use of the software described in this manual. These guidelines appear in appendix E. Before using the software described in this manual, the reader is strongly urged to review this appendix. The guidelines recommend use of this software in a manner that reduces the effort required to migrate application programs to future hardware or software systems.

DATA BASE DEFINITION

To define a data base, the data administrator uses the DDL. Through this language, four types of data descriptions can be created: the schema, the COBOL subschema, the FORTRAN subschema, and the Query Update subschema. Each of these data descriptions follows specific structuring conventions, includes unique clauses and statements, and conforms to an individual set of rules.

The relationship of the elements involved in defining a data base is shown in figure 1-1. The figure indicates that the schema directory must be available to the DDL compiler to generate subschema directories.

The data descriptions in the schema and subschema are organized into file-like structures. By convention, the following terms are used when referring to these structures:

Area in the schema

Realm in the subschema

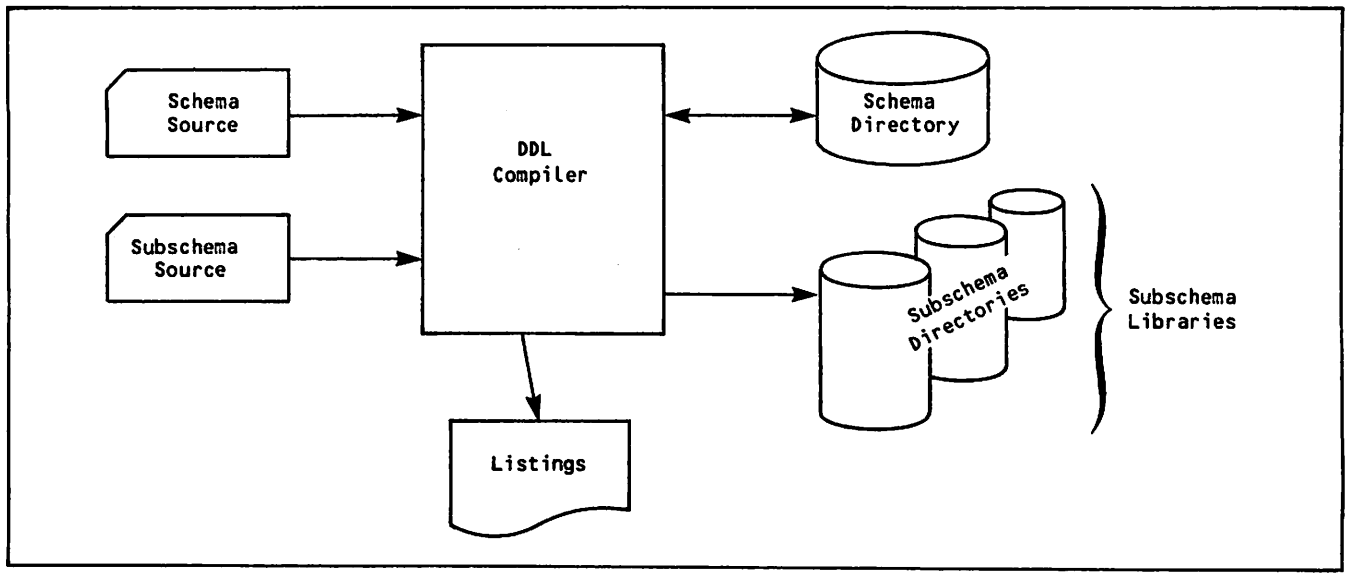


Figure 1-1. Defining a Data Base

The schema description of an area applies to all data within the structure. The subschema definition of a realm usually applies to a portion of data within the structure but can apply to all the data. The area provides the actual description of the data; the realm provides the description of data from the standpoint of the application program.

The term file is often used in this manual to refer to either an area or a realm.

SCHEMA DEFINITION

The schema is a detailed description in English-like syntax of the data in a data base. The schema description is created by DDL statements that name the schema, organize the schema into areas, describe each record type together with the characteristics of the data comprising the record, and describe relationships and constraints among areas. The schema also includes access control locks that provide privacy at the area level. The DDL source statements describing the data are used as input to the DDL compiler and are compiled into an object schema, or schema directory. The data administrator then uses the schema to define any number of subschemas.

SUBSCHEMA DEFINITION

A subschema is a detailed description of selected portions of data described by a schema. The subschema defines the portion of the data base available to the application program; the application program uses the subschema descriptions to access the data base. The subschema is based on the schema.

The subschema specifies realms and the contents and structure of records. The subschema indicates changes in data format required by an application program, identifies relations to be used, and specifies record qualification for relation processing.

The data descriptions in the subschema are organized into realms that correspond to areas in the schema. The realms included in a subschema can be a subset of the areas in the schema. The data items within the realm in a subschema can be a subset of the data items described for the corresponding area in the schema. Figure 1-2 illustrates the situation in which a subschema describes a portion of a data base.

Although only one schema is allowed for each data base, any number of subschemas can be defined to meet the needs of different types of applications. Subschemas are defined by the data administrator for use by application programs written in COBOL, FORTRAN, and Query Update.

The data administrator should provide the application programmer with a listing of the subschema. The listing is used to obtain the names and descriptions of the data referenced by the application program.

MASTER DIRECTORY DEFINITION

The data administrator must create a master directory before any application program accessing data base files can execute. The master directory contains information for all the elements of the data base known to CDCS. These elements include the schema, subschema, descriptions of data base files, and definitions of any data base versions. (Data base versions are described later in this section.)

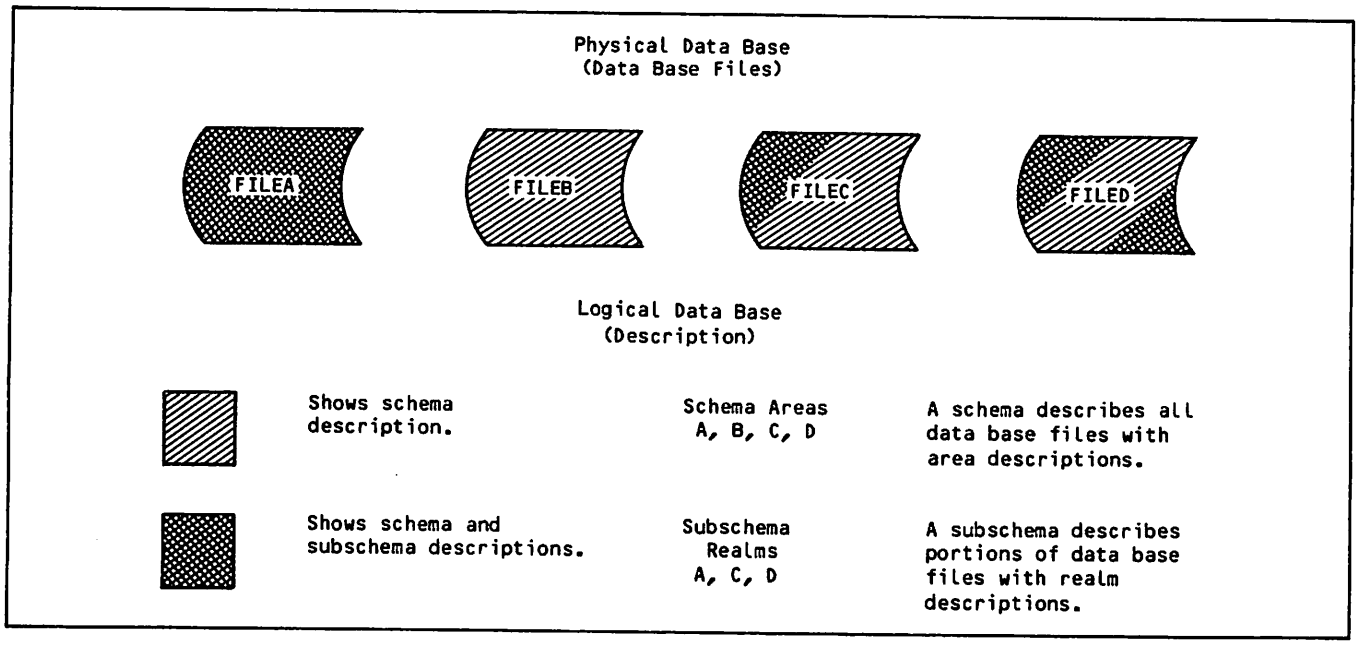


Figure 1-2. Subschema Describing a Portion of the Data Base

DATA BASE PROCESSING

Once a data base has been defined by the data administrator, it can be accessed by the following application languages: COBOL 5, FORTRAN 5, and Query Update 3. Processing of the data base by COBOL, FORTRAN, and Query Update programs is controlled and monitored by CDCS. These application languages can be used in either batch or interactive mode.

Figure 1-3 shows the relationship between the elements involved in processing a data base through CDCS.

COBOL PROCESSING

A COBOL program accesses data base files through conventional input/output statements. The files are opened and closed and records are read, written, deleted, and updated using similar means as for files that are not part of a data base. Relation processing is also accomplished by conventional COBOL statements. Data retrieved by the program is accessed according to the way it is described in the COBOL subschema.

When a COBOL program utilizing CDCS is to be compiled, the file containing the subschema directory must first be attached. Once the program is compiled using the subschema, it can be executed later without reattaching the subschema directory.

Execution of an input/output statement for a data base file in a COBOL program causes the COBOL object-time routines to route I/O calls to CDCS.

FORTRAN PROCESSING

A FORTRAN program accesses data base files through DML statements coded within the FORTRAN program. The DML consists of FORTRAN-like statements. These statements allow the FORTRAN user to access and modify data base files.

Before a program containing FORTRAN DML statements is compiled, the DML preprocessor is called via a control statement to translate the DML statements into FORTRAN specification statements and CALL statements. Data descriptions are obtained from the FORTRAN subschema directory, which must be attached during the preprocessing phase. Following the preprocessing, compilation of the FORTRAN program proceeds as for a conventional FORTRAN program; the translated DML statements are compiled like other FORTRAN statements. Once the program is preprocessed using the subschema, it can be compiled and executed later without reattaching the subschema directory.

QUERY UPDATE PROCESSING

Query Update functions within the data base environment whenever a Query Update subschema is specified by a Query Update user. The Query Update language, which is a special nonprocedural, interactive language, can be used by both programmers and nonprogramming personnel to perform several functions. Through simple directives, search, retrieval, update, and display operations can be performed on data base files as well as on conventional files. In addition, a single Query Update directive can be used in relation processing to display to the user data from more than one file. A comprehensive report writing capability is an integral part of Query Update.

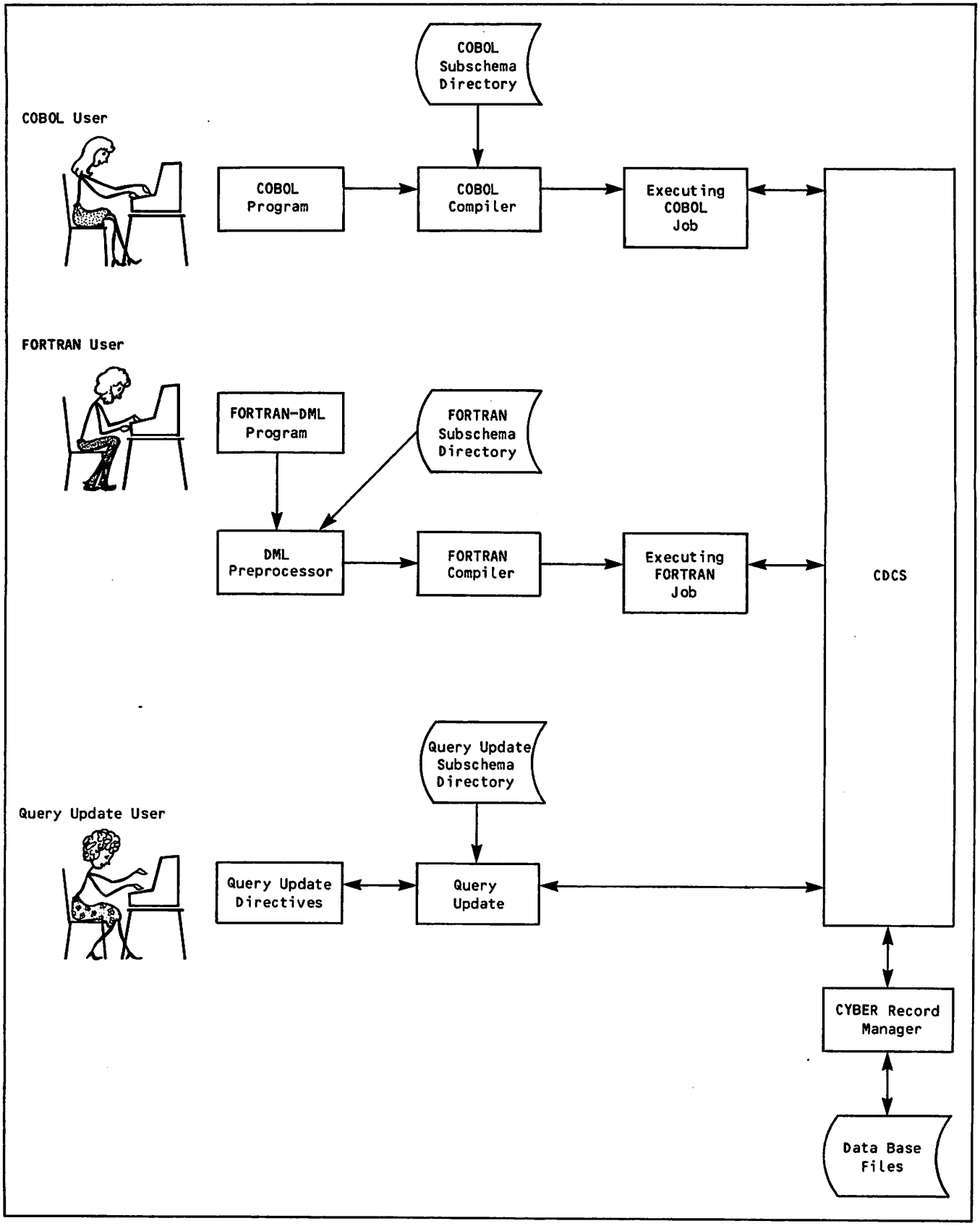


Figure 1-3. Relationship of the Elements Involved in Processing a Data Base

SPECIAL FEATURES INVOLVED IN CDCS PROCESSING

The special features described in the following subsections can affect the way an application program executes. Some features are used automatically with all CDCS processing; other features must be specifically requested by the data administrator during data base definition.

Concurrency

An important feature provided by CDCS is the concurrency feature. Concurrency means that two or more application programs can access the same data base file and the same data at the same time. Programs can access a file concurrently for retrieval or update purposes. During concurrent update operations, CDCS provides a locking mechanism by which files and records can be locked and unlocked at appropriate times. Automatic locking and unlocking are performed by CDCS when certain input/output operations are specified. In addition, explicit lock and unlock requests can be issued from the application program.

A deadlock situation can occur when two programs attempt to access files or records that have been locked by CDCS or by other programs. When this situation occurs, CDCS selects one of the contending programs, releases all locked resources held by that program, and issues an error status code. Appropriate code to handle recovery from a deadlock should be included in application programs.

File Privacy

Another valuable feature provided by CDCS is the access control (privacy checking) mechanism. Through this mechanism, access to data base files can be controlled on the basis of criteria specified by the data administrator. When the data administrator has specified controlled access to a data base file, an application program must specify the appropriate access control key (privacy key) to gain access to the controlled data base file. The data administrator should provide the user with any information necessary to access controlled data base files.

Relations

The relation processing component of CDCS allows data base files to be linked together into a logical structure called a relation. A relation is defined in the schema by the data administrator. Any relation that is available to the application program is defined by the data administrator in the subschema.

An application can access a relation by specifying a single read request (a display or extract request for Query Update). When the request is processed by CDCS, CDCS returns a record occurrence from each file in the relation to the application's working storage area.

Limitations can be placed on a relation by restrictions included in the subschema. These restrictions are in the form of qualification criteria that must be satisfied before a record occurrence is made available to the application.

A subschema listing provides the name of any relation and indicates what specific restrictions apply.

Constraints

The constraint component of CDCS is an independent feature that allows controls to be placed on update operations involving logically associated files. Constraints protect the integrity of the data base by allowing updates to be performed only when specific conditions are satisfied. Constraints are specified by the data administrator in the schema and are enforced by CDCS. Information concerning constraints should be provided by the data administrator to the application programmer.

Data Base Versions

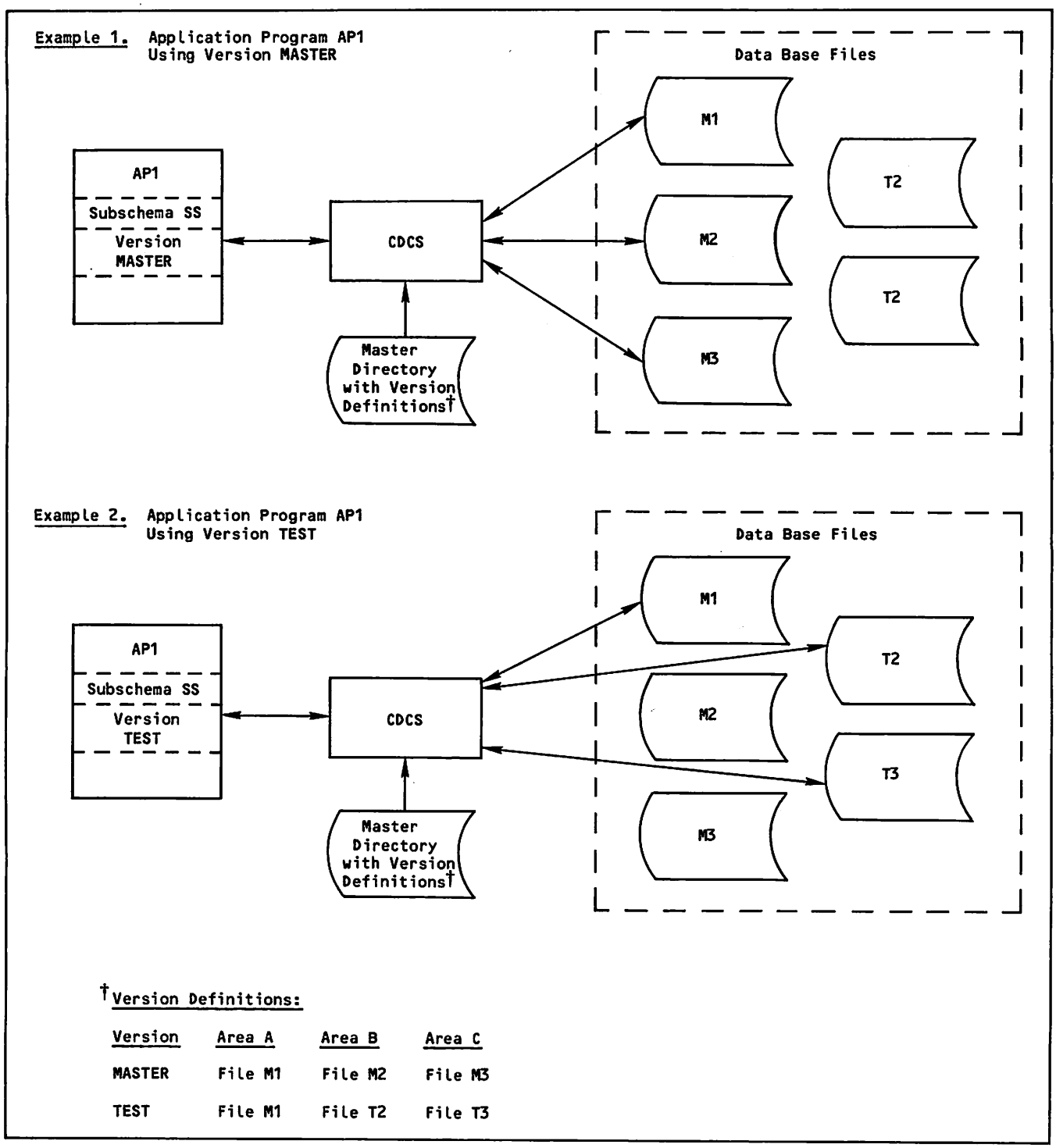
The data base version feature of CDCS allows an application program to use the same schema and subschema to access more than one group of permanent files corresponding to the areas in the schema, each group of which is defined as a data base version. Data base versions are defined by the data administrator in the master directory. If no data base versions are defined, a version named MASTER is assumed by default. If alternate data base versions are defined, one version named MASTER exists along with the alternate data base versions.

By specifying use of different versions, an application program can perform operations on different groups of files, each group forming a data base version. Within a single execution, an application program can perform processing on any number of data base versions. However, an application program can perform processing on only one data base version at a given time.

Figure 1-4 illustrates the use of data base versions. When the application program specifies a version name, CDCS makes available to the program all the files of that data base version that are associated with the realms included in the subschema being used. The illustration shows file M1 being shared by the versions MASTER and TEST.

Data Base Procedures

Data base procedures are special subprograms written by the data administrator to perform a variety of supplemental operations not otherwise performed by CDCS. The procedures are called at execution time when specific conditions occur during CDCS processing. The conditions under which data base procedures are to be executed are specified in the schema. If a data base procedure requires a response from an application program, the data administrator must provide the application programmer with the information required by the data base procedure.



†Version Definitions:

Version	Area A	Area B	Area C
MASTER	File M1	File M2	File M3
TEST	File M1	File T2	File T3

Figure 1-4. Processing Using Data Base Versions

Recovery

The recovery feature of CDCS provides for reconstruction of a damaged or inconsistent data base and provides for the removal of updates made with erroneous logic. The data administrator can restore the data base to a previous checkpoint or the beginning of a job when an application program failure or logic error occurs. The application program can define recovery points as an aid to recovery operations.

The transaction feature of CDCS is the application programming interface to automatic recovery. The transaction feature of CDCS is described in the following subsection.

Data Base Transaction

The data base transaction feature of CDCS provides the FORTRAN or COBOL application program with the ability to group a series of data base updates into a logical unit, called a data base transaction.

The application program specifies the beginning of the transaction, performs the update operations, and specifies the end of the transaction, which can be either a commit or a drop. When the application program specifies a commit operation, all updates made within the transaction become permanent. When the application program specifies a drop operation, all updates within the transaction are undone; therefore, the data base remains in the state it was in before the beginning of the transaction.

If the application program fails to commit a transaction because of system or program failure, automatic recovery is performed; that is, the transaction is dropped and the data base is restored to its state before the beginning of the transaction.

Transaction processing also provides an application program with the ability to determine the point at which to restart processing after a system failure. The application program can use this feature to determine the last transaction that was committed before the system failure occurred. The program can then determine the point at which processing should be restarted.

Immediate Return

The immediate return feature of CDCS provides COBOL and FORTRAN application programs with the ability to receive an immediate response from CDCS when either a resource conflict or a fatal error occurs. When this feature is utilized, CDCS returns control to the application program.

Resource conflicts occur when CDCS attempts to gain access to realms or records locked by other users. Normally, the application program waits for CDCS to gain access to these realms or records; however, when the immediate return feature is enabled, the application program can contain logic to determine the action taken in this situation.

When a fatal error occurs and the immediate return feature is enabled, CDCS disconnects the application program and disables the immediate return feature. A FORTRAN or COBOL program can complete any processing necessary before program termination.

The immediate return feature cannot be enabled before CDCS is invoked.

INPUT/OUTPUT PROCESSING

The input/output capabilities of CRM handle all operations concerning the physical storage and access of data in a data base. When an application program requests execution-time processing of input/output statements for data base files, CDCS directs the request to CRM. CRM processes the requests according to the requirements and restrictions for conventional files. Because CRM handles all the input and output processing, an application program might receive CRM errors.

All data base files supported by CDCS are conventional extended CYBER Record Manager Advanced Access Methods (AAM) files. File organization information is stored in the schema directory. This information includes the data item that is defined as the primary key. Records are stored in the data base by the value of the primary key. Duplicate primary keys are not permitted in a data base file.

The data administrator can also define alternate keys. The value of the primary or alternate keys can be used to access records in data base files.

PROCESSING THROUGH CDCS AND TAF

CDCS supports the Transaction Facility (TAF), which allows processing through TAF under NOS. Processing through TAF provides for high speed handling or repetitive executions of a relatively small number of jobs called tasks. The tasks can be executed by many different people from many locations. A task usually performs one of the following manipulations on a data base:

Stores a new record

Alters or deletes an existing record

Produces formatted output

An example of TAF-transaction processing is an online banking system. Tellers in many locations use terminals connected online to a central processor to make deposits or withdrawals for an account and to print confirmations. A task (a deposit or withdrawal) is initiated by a teller; once initiated, the task is executed through TAF and CDCS. The task can communicate with the terminal through TAF and the Network Access Method (NAM) and can initiate subsequent tasks.

Figure 1-5 shows the CDCS/TAF interface. Access to the data base through TAF is concurrent with access in both batch and interactive modes. All access to the data base is monitored by CDCS.

Special commands must be used for processing through TAF. Refer to the Transaction Facility reference manual for information about these commands.

In this manual, any reference to a transaction or to transaction processing refers to data base operations and not to TAF operations.

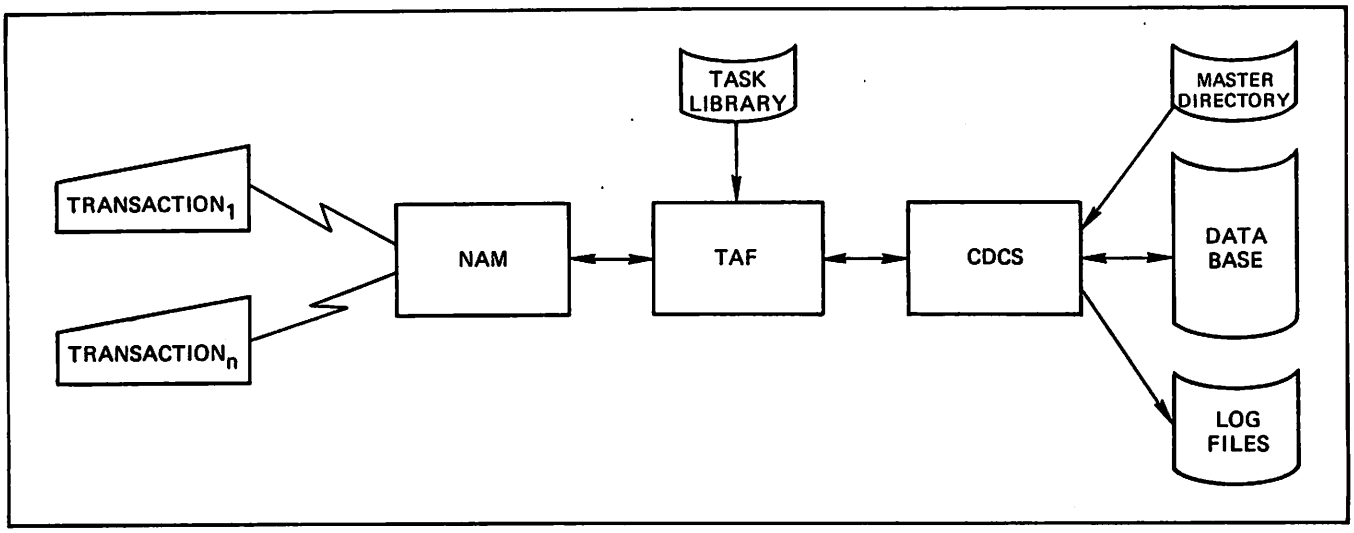


Figure 1-5. CDCS/TAF Interface

COBOL application programs can be used to access and manipulate a data base controlled by CYBER Data Base Control System (CDCS). The COBOL interface with CDCS consists of the COBOL subschema, the COBOL statements and routines used to access the data base, and the status checking elements available to a COBOL program.

In this section a basic knowledge of COBOL is assumed. The COBOL reference manual should be consulted for detailed information about COBOL.

COBOL SUBSCHEMA

The DMS-170 data base files that are to be accessed by a COBOL application program must be described in a directory called a COBOL subschema. The data administrator, working with the application programmers, is responsible for creating the subschemas.

When the subschema is compiled, a listing is produced. The listing provides information required by the application programmer to code the COBOL application program. The data administrator should provide the application programmer with a compilation listing of the subschema.

Some information required to access data base files is not included in the subschema listing. It is the responsibility of the data administrator to provide the application programmer with any necessary additional information.

This subsection documents the information provided by the subschema listing and the additional information that the data administrator must provide when the information is necessary for data base processing.

SUBSCHEMA LISTING

A subschema compilation listing can be used to obtain data names, item descriptors, and other information needed to process a realm. If a listing of the subschema used by the application program is not available, the parameter LO=M can be specified on the COBOL5 control statement. Specification of this parameter results in a listing that contains the description of each data item defined in the referenced subschema.

The information provided by the subschema is indicated by the following alphabetic list of items. Not all items are included in every subschema; those items that always appear in the subschema listing are so indicated. Figure 2-1 shows an example of each item in a sample subschema.

Alias names

The Alias Division identifies the data names and the names of records and realms that are used in the subschema in place of the name used in the schema. The name that follows the keyword BECOMES in the AD clause is the name that must be referenced in a COBOL application program.

Alternate key

The notation ALTERNATE KEY indicates a data item that is an alternate record key. The realm for which the data item is an alternate key is also indicated.

Checksum

Always present. A checksum is a one-word attribute generated by the DDL compiler for the subschema. The checksum of the subschema referenced by the COBOL application program is incorporated in the object program at compilation time and must agree at execution time with the checksum associated with the subschema in the master directory. Refer to the subsection Recompilation Guidelines for more information.

Data description entry (name and description)

Always present. Data description entries are entries identified by a level number 02 or greater. Data description entries identify the name and description of data items. For any data name described by only a PICTURE clause, the usage of DISPLAY is assumed. Any USAGE clause in the subschema indicates usage as described in the COBOL reference manual. The sample subschema includes only a few of the data description entries that could be in a subschema. For detailed information about data description entries, refer to the CDCS 2 Data Administration reference manual.

Only data base items represented by data description entries are available to the application program. Data read from the data base and received in the working storage area of an application program is mapped according to the subschema description.

Group item

A group item is made up of other data items. The lower level number, 03 in the subschema, denotes the group item name. The group item name can be used in COBOL statements to qualify the name of a data item that is in the group item.

```
                    * SOURCE LISTING *   (82029)  DDL  3.2+552.

00001          TITLE DIVISION.
00002              SS  PRODUCT-PERSONNEL WITHIN MANUFACTURING-DB.
00003
00004          ALIAS DIVISION.
00005              AD  REALM JOBDETAIL BECOMES WORK-FILE.
00006              AD  RECORD JOBREC BECOMES WORK-REC.
00007              AD  RECORD EMPREC BECOMES EMP-REC.
00008              AD  DATA LOC-CODE BECOMES LOCATION.
00009
00010          REALM DIVISION.
00011              RD EMPLOYEE, WORK-FILE.
00012
00013          RECORD DIVISION.
00014          01  EMP-REC.                                    ** WITHIN EMPLOYE
00015              03  EMP-ID              PICTURE X(8).       ** ORDINAL    1
00016              03  SALARY              PICTURE 9(6)V99     ** ORDINAL    2
00017                                        USAGE IS COMP-1.
00018              03  EMP-LAST-NAME       PICTURE A(20).      ** ORDINAL    3
00019              03  EMP-INITIALS        PICTURE A(4).       ** ORDINAL    4
00020               .
00021               .
00022               .
00026              03  PHONE-NO            PICTURE 9(10).      ** ORDINAL   11
00027              03  PHONE               REDEFINES PHONE-NO. ** ORDINAL   12
00028                  05  AREA-CODE       PICTURE 999.        ** ORDINAL   13
00029                  05  PREFIX          PICTURE 999.        ** ORDINAL   14
00030                  05  DIGITS          PICTURE 9999.       ** ORDINAL   15
00031              03  JOB-CLASS           PICTURE A.          ** ORDINAL   16
00032              03  GRADE-LEVEL         PICTURE 9.          ** ORDINAL   17
00033              03  DEPARTMENT          PICTURE X(4).       ** ORDINAL   18
00034
00035          01  WORK-REC.                                   ** WITHIN WORK-FI
00036              03  CONCATKEY.                              ** ORDINAL    1
00037                  05  EMP-ID          PICTURE X(8).       ** ORDINAL    2
00038                  05  SEQ-NO          PICTURE X(4).       ** ORDINAL    3
00039              03  PRODUCT-ID          PICTURE X(10).      ** ORDINAL    4
00040              03  SECURITY-CODE       PICTURE X(2).       ** ORDINAL    5
00041              03  MONTHLY-COMPENSATION USAGE IS COMP-1    ** ORDINAL    6
00042                                        OCCURS 12 TIMES.
00043                  05  REG-HOURS       PICTURE 9(3)V99.    ** ORDINAL    7
00044                  05  REG-COMPENSATION PICTURE 9(4)V99.   ** ORDINAL    8
00045                  05  OT-HOURS        PICTURE 9(3)V99.    ** ORDINAL    9
00046                  05  OT-COMPENSATION PICTURE 9(4)V99.    ** ORDINAL   10
00047              03  HOURS-YTD           PICTURE 9(4)V99     ** ORDINAL   11
00048                                        USAGE IS COMP-1.
00049              03  COMPENSATION-YTD    PICTURE 9(8)V99     ** ORDINAL   12
00050                                        USAGE IS COMP-1.
00051              03  LOCATION            PICTURE X(4).       ** ORDINAL   13
```

Schema name
Subschema name
Alias name
Realm name
Record name
Subschema item ordinal
Data description
Data name
Concatenated key, also Group item
Major key

Figure 2-1. Sample COBOL Subschema (Sheet 1 of 2)

```
                00052
PRIMARY KEY     00015       EMP-ID FOR AREA EMPLOYEE                                  <- Primary key and
ALTERNATE KEY   00033       DEPARTMENT FOR AREA EMPLOYEE                              <- associated realm
PRIMARY KEY     00036       CONCATKEY FOR AREA WORK-FILE                              <- Alternate key and
                *****       RECORD MAPPING IS NEEDED FOR REALM -   EMPLOYEE           <- associated realm
                *****       RECORD MAPPING IS NEEDED FOR REALM -   WORK-FILE

                00053               RELATION DIVISION.
                00054                   RN IS EMP-REL                                 <- Relation name
                00055                       RESTRICT WORK-REC WHERE SECURITY-CODE GE "B1".
                00056                                                                 <- Restriction
                00057
                *****       END OF SUB-SCHEMA SOURCE INPUT

                *****               RELATION   STATISTICS              *****          <- Relation
RELATION 001                EMP-REL JOINS                AREA - WORK-FILE             <- Rank1
                                                         AREA - EMPLOYEE              <- Rank2


                -----       BEGIN SUB-SCHEMA FILE MAINTENANCE              -----

          SUBSCHEMA                              CHECKSUM
          PRODUCT-PERSONNEL                      02066614100322474160                <- Checksum

                -----       END OF FILE MAINTENANCE              -----
DDL COMPLETE.            0 DIAGNOSTICS.
  47700B CM USED.      0.334  CP SECS.
```

Figure 2-1. Sample COBOL Subschema (Sheet 2 of 2)

Major key

A major key is the leading data name of a group item that is defined as a record key. In this example, CONCATKEY is the group item that is a record key. EMP-ID is the major key and can be used in major key processing.

Primary key

Always present for each realm included in the subschema. The notation PRIMARY KEY indicates each data item that is a primary record key. The realm for which the data item is a primary record key is also indicated.

Ranks of the relation

The realms joined in a relation are listed by realm name in order of rank. The first realm listed has a rank of 1; the second, a rank of 2; and so forth.

Realm (file) names

Always present. The Realm Division identifies the realms available to the application program. Unless ALL is specified, only those realms included in the Realm Division are available to the application program.

The complete realm name always appears in the subschema listing on a line with the notation PRIMARY KEY and the data name of the primary key for the realm.

Record names, associated realms and data names

Always present. An 01 level data description identifies the record name. Each record available to the application program is included in the subschema listing.

The notation WITHIN identifies the particular realm associated with a record by giving the first seven characters of the realm name.

The data descriptions of 02 level or greater that follow the record name identify the data names that comprise the record.

The record name can be used in COBOL statements to qualify any data name within the record.

Relation names

The RN clause in the Relation Division identifies a relation name. An RN clause is included for each relation available to the application program.

Restriction

A RESTRICT clause in the Relation Division identifies a restriction. If a restriction is placed on a relation, only records that meet all the qualification criteria specified in the RESTRICT clause are returned to the application program when the relation is read.

If the RESTRICT clause contains a data name that is not included in the subschema, the data name must be defined in the Data Division of the program with the same usage as the data name used for comparison. The data name must be set to a value before any read that uses the relation.

Schema name

Always present. The SS clause in the Title Division identifies the schema. The schema name follows the keyword WITHIN.

Subschema item ordinal

Always present. The subschema item ordinal is a unique identifier within a record that is assigned to each data name in a subschema when the subschema is compiled. A subschema item ordinal is used in conjunction with the data base status block.

Subschema name

Always present. The SS clause in the Title Division identifies the subschema name. An application program must reference the subschema in the SUB-SCHEMA clause by using the subschema name.

INFORMATION PROVIDED BY DATA ADMINISTRATOR

It is the responsibility of the data administrator to provide the following information when it is required for data base processing:

Constraints

If constraints are defined in the schema and updates are likely to violate them, the application programmer should be provided with information about the constraints. Refer to section 5 for more information about constraints.

Join items of the relation

Files are joined in a relation by identical data items that exist in two files. The information about the join items is contained in the schema. Usually, an application programmer does not need to know the join items to use a relation. Often the programmer can determine the join items from the subschema listing. (Refer to section 5 for more information on relations.) However, in some situations, the data administrator should provide the application programmer with the information about join items.

Permanent file information for the subschema library

The subschema directory must be available for compilation of the COBOL application program; therefore, the application programmer must be provided with the information required to attach the subschema library file containing the subschema directory.

Privacy keys

If an area has been defined with an access control lock in the schema, a USE FOR ACCESS CONTROL statement must be included in the COBOL application program to access the realm. The application programmer must be provided with all access control keys required for data base access. (Refer to the description of the USE FOR ACCESS CONTROL Declarative Statement subsection for further information.)

Requirements imposed by any data base procedure

If data base procedures are defined in the schema, the application programmer must be provided with information required by the data base procedures.

Transaction update limits

If limits have been imposed on the number of transactions allowed for all users of the schema, the application programmer should be provided with this information.

Version name

If alternate data base versions are defined for the data base, the application programmer must be provided with information about the name and use of the alternate data base versions.

SUBSCHEMA DIRECTORY

When the subschema source input is compiled, the object subschema, called the subschema directory, is generated. The subschema directory is usually included in a subschema library. The subschema directory must be available to the COBOL compiler; therefore, the subschema library must be attached before compilation.

During compilation, the COBOL compiler includes in the COBOL program the descriptions of all files, data names, and relations that are described in the subschema.

PROGRAM CODING

Coding a COBOL program that accesses a data base is basically the same as coding any other COBOL program. Table 2-1 shows the COBOL statements and routines that can be used. This table also briefly describes the statements and routines and indicates their position in a COBOL program. The statements and routines are described in more detail in the following subsections. The following subsections also describe restrictions and requirements imposed on the COBOL program.

ENVIRONMENT DIVISION

In the Environment Division, the SPECIAL-NAMES paragraph and the FILE-CONTROL paragraph are affected by subschema use. The SUB-SCHEMA clause is required in the SPECIAL-NAMES paragraph. Only restricted File-Control entries can be specified for data base files.

SUB-SCHEMA Clause

The SUB-SCHEMA clause, shown in figure 2-2, appears in the SPECIAL-NAMES paragraph and is required to identify the subschema and acknowledge subschema use.

```
SUB-SCHEMA IS subschema-name
```

Figure 2-2. SUB-SCHEMA Clause Format

Subschema-name specifies the 1- to 30-character subschema name, which is obtained from the Title Division of the subschema. The subschema directory must reside on the file identified by the D parameter of the compiler call statement.

File-Control Entry

The File-Control entry is not required; however, it can be included for compatibility with previous versions of CDCS and COBOL. It must be included if a FILE STATUS clause is used. In the FILE-CONTROL paragraph a limited File-Control entry can be coded for any realm for which a FILE STATUS clause applies in returning CRM errors and end-of-file status to the program. If the FILE-STATUS clause is to be used, the File-Control entry must include the SELECT clause to specify the realm name and the ASSIGN clause to specify the local file name associated with the realm, as well as the FILE STATUS clause itself. (The local file name in the ASSIGN clause is ignored, however, because the file is not attached at the user control point.) If no File-Control entries are specified, the FILE-CONTROL and INPUT-OUTPUT SECTION headers are not required.

The file-name for the SELECT clause is obtained from the Realm Division of the subschema; implementor-name-1 is the first 7 characters of the realm name.

DATA DIVISION

File Description and associated Record Description entries for a realm are contained in the COBOL subschema. Therefore, the File Section in the Data Division of a COBOL program must not include an FD entry or Record Description entry for any realm. Files that are not data base files can be described in the File Section.

TABLE 2-1. COBOL STATEMENTS AND ROUTINES

Statement	Function	Position in Program
	Establishing and Terminating Data Base Access	
SUB-SCHEMA	Identifies the subschema to be used by the program or subprogram.	Must appear in the SPECIAL-NAMES paragraph of the Environment Division.
DB$VERS	Changes the data base version used by an application program.	Anywhere in the Procedure Division as long as no realms are open; must not appear in a transaction.
	Opening and Closing a Realm or Relation	
OPEN realm	Initiates processing of a realm.	Anywhere in the Procedure Division, except within a transaction; can only be executed when the specified realm is closed.
OPEN relation	Initiates processing of the realms joined in a relation.	Anywhere in the Procedure Division, except within a transaction; can only be executed if the realms in the specified relation are closed.
CLOSE realm	Ends processing of a realm.	Anywhere in the Procedure Division, except within a transaction; can only be executed if the specified realm is open. A realm can be opened or closed any number of times in the same run-unit.
CLOSE relation	Ends processing of the realms joined in a relation.	Anywhere in the Procedure Division, except within a transaction. The realms in a relation can be opened and closed any numbers of times in the same run-unit.
	Manipulating a Data Base	
START realm	Logically positions a realm for subsequent retrieval of records through a sequential READ statement.	Anywhere in the Procedure Division after the relevant realm is opened.
START relation	Logically positions the root realm of the specified relation for subsequent retrieval of records through a READ relation statement.	Anywhere in the Procedure Division after the relevant realm is opened.
READ realm	Transfers data from a record in the specified realm to the data items included in the subschema description of the record.	Anywhere in the Procedure Division after the relevant realm is opened.
READ relation	Transfer data from a record in each of the realms joined in the relation to the data items included in the subschema descriptions of the records.	Anywhere in the Procedure Division after the relevant realm is opened.
C.LOK	Establishes a lock that prevents other programs from updating the the specified realm.	Anywhere in the Procedure Division after the relevant realm is opened, except within a transaction.
C.UNLOK	Releases a lock on a specified realm; releases any locks the program holds on records in the specified realm.	Anywhere in the Procedure Division after the relevant realm is opened, except within a transaction.
DB$LKAR	Establishes a lock on the specified realm that prevents other programs from reading and/or updating the realm.	Anywhere in the Procedure Division after the relevant realm is opened, except within a transaction.

TABLE 2-1. COBOL STATEMENTS AND ROUTINES (Contd)

Statement	Function	Position in Program
	Manipulating a Data Base (Contd)	
DELETE	Logically removes a record from a realm.	Anywhere in the Procedure Division after the relevant realm is opened; should be preceded by READ, C.LOK, or DB$LKAR.
REWRITE	Logically replaces a record in a realm.	Anywhere in the Procedure Division after the relevant realm is opened; should be preceded by READ, C.LOK, or DB$LKAR.
USE FOR ACCESS CONTROL	Establishes the right of a program to access a realm or realms.	Must be the first sentence after a section header within the declaratives portion of the Procedure Division; must not appear within a transaction.
USE FOR DEADLOCK	Identifies the procedure to be used when a deadlock situation occurs.	Must be the first sentence after a section header within the declaratives portion of the Procedure Division.
WRITE	Causes a record to be stored in a realm consisting of the current values of the data items included in the subschema description of the record.	Anywhere in the Procedure Division after the relevant realm is opened.
	Processing a Data Base Transaction	
DB$ASK	Obtains the restart identifier assigned by CDCS.	Anywhere in the Procedure Division, except within a transaction. Normally, DB$ASK is specified before a transaction is initiated.
DB$BEG	Identifies and begins a transaction.	Anywhere in the Procedure Division after the relevant realm is opened; transaction processing must be allowed for the schema.
DB$CMT	Causes all updates of a successful transaction to be made permanent.	Anywhere in the Procedure Division after the associated DB$BEG; a transaction must have been initiated.
DB$DROP	Cancels an active transaction; the records updated within the transaction are restored to the states that they were in before the transaction began.	Anywhere in the Procedure Division after DB$BEG but before DB$CMT.
DB$GTID	Determines the appropriate place to restart transaction processing when when a program is recovering from a system failure.	Anywhere in the Procedure Division, usually in the program logic used when recovering from a system failure.
	Miscellaneous	
DB$DBST	Communicates the location and length of the data base status block.	Anywhere in the Procedure Division.
DB$RPT	Defines a recovery point to CDCS.	Anywhere in the Procedure Division, except within a transaction.
DB$WAIT	Forces a deadlock situation when CDCS is running as CDCSBTF so that deadlock code can be tested.	Anywhere in the Procedure Division after the relevant realm is opened.

Note: Other routines that provide status information are available; however, the use of DB$DBST is recommended.

A database status block, described later in this section, can be defined in the Working-Storage Section or the Common-Storage Section of a COBOL program to provide CDCS with an area of memory to which it can return error and status information.

PROCEDURE DIVISION

In the Procedure Division, a data base file is processed as if it had been described in the File Section of the Data Division. Record names and data names are obtained from the Record Division of the subschema as described in the preceding subsection. Some statements cannot be used when accessing a data base. A data base can be processed using any of the Procedure Division statements except the following:

ACCEPT

DISPLAY

GIVING phrase of SORT or MERGE

USING phrase of MERGE

Some of the Procedure Division statements and routines cannot be used within a data base transaction. (Any reference to a transaction in this section refers to a data base transaction.) A transaction begins when the DB$BEG routine is entered and includes any statements used for updates within the transaction. A transaction ends when either the DB$DROP or the DB$CMT routine is entered.

The following subsections describe the Procedure Division statements and routines used to access a data base and state whether they can be used within a transaction. The statements are listed alphabetically.

CLOSE Statement

The CLOSE statement, shown in figure 2-3, ends processing of the specified realm or of the realms joined in the specified relation. The CLOSE statement is not allowed within a transaction.

```
Format 1:

  CLOSE realm-name-1 [, realm-name-2] ...

Format 2:

  CLOSE relation-name-1 [, relation-name-2] ...
```

Figure 2-3. CLOSE Statement Format

The CLOSE realm statement (format 1) terminates the availability of one or more specified realms. This statement is a subset of the format 1 CLOSE statement that appears in the COBOL reference manual. In the CLOSE realm statement, realm-name specifies the realm to be closed.

The CLOSE relation statement (format 2) terminates the availability of each of the realms comprising the specified relation. The CLOSE relation statement is performed as if a separate close were executed for each realm, in the order of rank of the realms in the relation. Realms closed by a CLOSE relation statement should not be explicitly closed by a CLOSE realm statement.

C.DMRST Routine

The C.DMRST routine, shown in figure 2-4, obtains the status of the realms within a relation. This routine scans the realms in the order the realms are traversed, beginning with the root realm (the realm in a relation with rank 1) and continuing until a nonzero error code is encountered or all realms have been scanned. C.DMRST should be entered after execution of a READ relation statement to determine whether an error occurred during the relation read operation and on which realm the error occurred. The C.DMRST routine can be executed as many times as needed to check for errors or relation conditions on the remaining realms in the relation.

```
ENTER "C.DMRST" USING relation-name,
    rlm-name, err-code
```

Figure 2-4. C.DMRST Routine Format

The USING phrase specifies the name of the relation to be checked for file status and two data items for returned information.

If an error, a null condition, or a control break occurs for a realm, rlm-name receives the implementor-name (the local file name) of the realm. If none of the preceding conditions occur, the contents of rlm-name are undefined. Rlm-name must be specified as a word-aligned (synchronized left) elementary data item described as PICTURE X(7).

If an error, a null condition, or a control break occurs in a realm, err-code receives the CDCS status or error code. If no errors, null records, or control breaks occur on any of the realms in the relation, err-code receives a zero. Err-code must be specified as a numeric integer data item with a size greater than two and a usage of COMPUTATIONAL-1.

The use of the DB$DBST routine is recommended rather than the use of the C.DMRST routine, since the DB$DBST routine provides the user with more relevant information about relation processing.

C.IOST Routine

The C.IOST routine, shown in figure 2-5, obtains CRM error codes. The CRM error codes are returned for the error that occurred while processing the realm specified in the statement.

In the USING phrase, realm-name must specify the name of the realm in which the error was detected.

```
ENTER "C.IOST" USING realm-name, err-code,
    err-type
```

Figure 2-5. C.IOST Routine Format

Err-code receives the decimal equivalent of one of the CRM error codes (an octal value) listed in the CRM Advanced Access Methods reference manual. Err-code should be specified as a 3-digit integer described as USAGE IS COMPUTATIONAL-1 in its Data Description entry.

Err-type receives either the letter T or the letter F (the severity of the error as classified by COBOL); the significance of these letters can be found in the COBOL reference manual. Err-type must be specified as a 1-character alphanumeric data item described in its Data Description entry as PICTURE X.

The use of the DB$DBST routine is recommended rather than the use of the C.IOST routine. The use of the DB$DBST routine provides the user with more relevant information than this routine provides.

C.LOK Routine

The C.LOK routine, shown in figure 2-6, prevents other jobs from updating the specified realm until either the C.UNLOK routine is entered or a CLOSE statement is entered. Once the realm has been locked, other programs can access it through read, but not update, operations. The realm must be successfully opened before it can be locked. The realm lock provided by this routine is comparable to the protected lock provided by the DB$LKAR routine. (See the DB$LKAR Routine subsection for more information about realm locks.) The C.LOK routine is not allowed within a transaction.

The USING phrase indicates the realm to be locked. Realm-name must be included in the subschema.

```
ENTER "C.LOK" USING realm-name
```

Figure 2-6. C.LOK Routine Format

C.UNLOK Routine

The C.UNLOK routine, shown in figure 2-7, releases the lock on a specified realm; the routine also releases all record locks the run-unit holds on records in the specified realm. The C.UNLOK routine is not allowed within a transaction.

The USING phrase indicates the realm to be unlocked. Realm-name must be included in the subschema.

```
ENTER "C.UNLOK" USING realm-name
```

Figure 2-7. C.UNLOK Routine Format

DB$ASK Routine

The DB$ASK routine, shown in figure 2-8, obtains information for a program restart operation after a system failure. This routine is normally specified in the restart section of the run-unit.

```
ENTER "DB$ASK" USING restart-id,
    tran-id, err-code
```

Figure 2-8. DB$ASK Routine Format

In the USING phrase, restart-id identifies the 1- to 10-character restart identifier that was assigned to the program prior to the system failure. Restart-id must be specified either as a nonnumeric literal or as a word-aligned (synchronized left) alphanumeric data item described as PICTURE X(10) in its Data Description entry.

Tran-id receives the 1- to 10-character transaction identifier of the last completed transaction. The tran-id that is returned is a left-justified and blank field in the 10-character field. Tran-id is returned only if prior to the system failure a transaction associated with the specified restart identifier was committed. Tran-id must be specified as a word-aligned (synchronized left) alphanumeric data item described in its Data Description entry as PICTURE X(10).

Tran-id receives the characters ********** (10 asterisks) if the restart identifier is unknown to CDCS. The restart identifier is unknown to CDCS if the wrong value is specified for restart-id or if the run-unit terminated normally. If the run-unit terminated normally, a new restart identifer should be obtained before processing continues.

If a transaction identifier is returned, no new restart identifer need be obtained; the restart identifer specified as restart-id is reassigned to the run-unit.

If no transaction has been committed for the specified restart identifier, tran-id receives spaces. The DB$ASK routine completes execution normally. No new restart identifier need be obtained; the restart identifier specified as restart-id is reassigned to the run-unit.

If an error occurs in the execution of the DB$ASK routine, err-code receives the appropriate 3-digit integer error code. Err-code should be described as USAGE IS COMPUTATIONAL-1 in its Data Description entry.

Error information is also returned in the data base status block if one is supplied. (Refer to the Data Base Status Block subsection for more information about this feature.)

The user must ensure that all of the parameters of this routine are specified. Omission of any parameter causes indeterminate results.

DB$BEG Routine

The DB$BEG routine, shown in figure 2-9, defines the beginning of a transaction to CDCS. Records that are subsequently updated remain exclusively locked until the transaction is either completed or dropped. Updates (write, rewrite, and delete operations) are considered temporary until the transaction is successfully completed. If an attempt is made to execute the DB$BEG routine when transaction processing is not allowed for the schema, a fatal error occurs. Refer to section 5 for a description of transaction processing.

```
ENTER "DB$BEG" USING tran-id, err-code
```

Figure 2-9. DB$BEG Routine Format

In the USING phrase, tran-id is the 1- to 10-character transaction identifier supplied by the user. Tran-id should be specified as either a nonnumeric literal or as a word-aligned (synchronized left) alphanumeric data item described in its Data Description entry as PICTURE X(10).

Err-code receives the appropriate 3-digit integer CDCS error code if an error occurs during the execution of the DB$BEG routine. Err-code should be described as USAGE is COMPUTATIONAL-1 in its Data Description entry.

The user must ensure that all of the parameters of this routine are specified. Omission of any parameter causes indeterminate results.

DB$CMT Routine

The DB$CMT routine, shown in figure 2-10, indicates the completion of a transaction to CDCS. Execution of the DB$CMT routine causes all updates made within the present transaction to become permanent; all records locked within the transaction are released and made accessible to other application programs. If a transaction has not been initiated (refer to the DB$BEG routine) and an attempt is made to execute the DB$CMT routine, a fatal error occurs.

```
ENTER "DB$CMT" USING err-code
```

Figure 2-10. DB$CMT Routine Format

In the USING phrase, err-code receives the 3-digit integer CDCS error code if an error occurs during the execution of the DB$CMT routine. Err-code must be described as USAGE IS COMPUTATIONAL-1 in its Data Description entry. Err-code must be provided; if err-code is omitted, results are indeterminate.

DB$DBST Routine

The DB$DBST routine, shown in figure 2-11, communicates the location and length of the data base status block to CDCS. Only one data base status block can exist at a time for a run-unit. This routine can be called at any time in a COBOL program. Each time DB$DBST is entered, the status block is intialized to zero or spaces; therefore,

```
ENTER "DB$DBST" USING status-block, len
```

Figure 2-11. DB$DBST Routine Format

this routine should not be entered after a request to CDCS because status elements could be set after the request. If DB$DBST is called more than once in a program, the data base status block defined in the last call executed is the one that is updated by CDCS.

The data base status block specified in the USING phrase as status-block receives data base status information after each CDCS request. Status-block must be specifed as the 01 level data-name of the group item or table to which the information is returned.

Len identifies the length of the data base status block. Len must be specified as a COMPUTATIONAL-1 data item which defines the length of the block in words. (Refer to the Data Base Status Block subsection for more information.)

DB$DROP Routine

The DB$DROP routine, shown in figure 2-12, cancels the current transaction. Execution of the DB$DROP routine causes CDCS to restore the records updated within the transaction to their original states that existed just before the transaction was initiated (refer to the DB$BEG routine). Execution of the DB$DROP routine also causes CDCS to release all record locks established during the transaction. If a transaction has not been initiated and an attempt to execute the DB$DROP routine is made, a fatal error occurs.

```
ENTER "DB$DROP" USING err-code
```

Figure 2-12. DB$DROP Routine Format

In the USING phrase, err-code receives the appropriate 3-digit integer CDCS error code if an error occurs during the execution of the DB$DROP routine. Err-code must be described as USAGE IS COMPUTATIONAL-1 in its Data Description entry. Err-code must be provided; if err-code is omitted, results are indeterminate.

DB$GTID Routine

The DB$GTID routine, shown in figure 2-13, obtains the restart identifier assigned by CDCS. This identifier can subsequently be used to determine the status of a transaction when a system failure occurs. This routine must not appear within a transaction. If the data administrator has not assigned a restart identifier file to the data base, this routine cannot be used; if the routine is used, an error results.

```
ENTER "DB$GTID" USING restart-id, err-code
```

Figure 2-13. DB$GTID Routine Format

If the DB$GTID routine executes successfully, CDCS returns the assigned restart identifier. The restart identifier should be saved for use in case of a system failure. Because the restart identifier is saved for recovery of data base files, it should not be saved on a data base file. The application should contain logic to save the restart identifier outside of the program.

In the USING phrase, restart-id receives the 1- to 10-character restart identifier. Restart-id must be specified as a word-aligned (synchronized left) alphanumeric data item described as PICTURE X(10) in its Data Description entry.

Err-code receives the appropriate 3-digit integer CDCS error code if an error occurs during the execution of the DB$GTID routine. Err-code should be described as USAGE IS COMPUTATIONAL-1 in its Data Description entry.

The user must ensure that all of the parameters of this routine are specified. Omission of any parameter causes indeterminate results.

DB$LKAR Routine

The DB$LKAR routine, shown in figure 2-14, provides two modes of locking a realm: exclusive and protected. Exclusive locking prohibits concurrent reading or updating of the realm; protected locking allows concurrent reading of a realm, but prohibits concurrent updating. A realm lock remains in effect until either the C.UNLOK routine is entered or the realm is closed (a CLOSE statement is entered). For transaction processing, the use of this routine is not recommended. The DB$LKAR routine is not allowed within a transaction.

```
ENTER "DB$LKAR" USING realm-name,
    lock-type, err-code
```

Figure 2-14. DB$LKAR Routine Format

When a realm has been locked, the recommended procedure for deleting or rewriting a record in that realm is to read the record and then perform the delete or rewrite operation. This procedure offers protection for the data base, because program logic can check for an error on the read before proceeding with the delete or rewrite. Another procedure can also be used: the value of the primary key can be set to the value of the key of the record to be deleted or rewritten, and then the delete or rewrite operation can be performed. This procedure does not offer as much protection to the data base.

The use of a realm lock limits access by other users to the realm; also, it overrides the record lock and the checking capabilities through the record lock. For these reasons, the DB$LKAR routine should not be used without careful consideration. Refer to section 5 for more information about using realm locks.

In the USING phrase, realm-name specifies the realm to be locked. Realm-name must be included in the subschema.

Lock-type specifies the type of locking desired. Lock-type can be specified as either a nonnumeric literal or as a word-aligned (synchronized left) alphanumeric data item described as PICTURE X(9) in its Data Description entry. The data item must have a value of EXCLUSIVE or PROTECTED. This value can be given to the data item or specified in a value clause. EXCLUSIVE and PROTECTED have the meanings described previously.

Err-code receives the appropriate 3-digit integer CDCS error code if an error occurs during the execution of the DB$LKAR routine. Err-code must be described as USAGE IS COMPUTATIONAL-1 in its Data Description entry.

The user must ensure that all of the parameters of this routine are specified. Omission of any parameter causes indeterminate results.

DB$RPT Routine

The DB$RPT routine, shown in figure 2-15, defines a recovery point to CDCS. This is a point to which the data base is restored so that the program can be easily restarted should recovery of the data base become necessary.

```
ENTER "DB$RPT" USING rpt-num, comment
```

Figure 2-15. DB$RPT Routine Format

When the DB$RPT routine executes, the COBOL program is suspended until three events occur in the following order:

1. All I/O buffers for data base realms for this schema are flushed.
2. A recovery point log is force written to the log file for the data base.
3. The quick recovery file for the data base is emptied.

By using this routine, the user is assured the data base can be recovered to its current state (barring such disasters as simultaneous destruction of both the data base and the journal log file).

The creation of a recovery point does incur overhead since CDCS activity halts for all users until the preceding events have occurred. The data administrator should be consulted to determine if the use of a recovery point is applicable to a particular file.

In the USING phrase, rpt-num receives a unique recovery point number upon execution of the DB$RPT routine. Rpt-num is a numeric integer data item that must be described as USAGE IS COMPUTATIONAL-1 in its Data Description entry.

Comment specifies a 1- to 30-character user-supplied explanatory message that is written to the CDCS journal log file along with the recovery point number. Comment must be specified as a word-aligned (synchronized left) alphanumeric item of 30 characters in length described in its Data Description entry as PICTURE X(30).

DB SIR Routine

The DB$SIR routine, shown in figure 2-16, enables and disables the immediate return feature. If the immediate return feature is enabled, a COBOL application program can receive an immediate response from CDCS when resource conflicts or fatal errors that are caused by resource conflicts occur. If the immediate return feature is disabled, a COBOL application program cannot provide program logic to determine the action taken when resource conflicts or fatal errors that are caused by resource conflicts occur.

```
ENTER "DB$SIR" USING item-name
```

Figure 2-16. DB$SIR Routine

In the USING phrase, item-name determines whether the immediate return feature is enabled or disabled. If item-name is not equal to zero, immediate return is enabled. If item-name is equal to zero, immediate return is disabled. Item-name must be described in its Data Description entry as COMPUTATIONAL-1. Refer to section 5 for more information about the immediate return feature.

A data base status block must be defined if a COBOL program is to be able to detect a fatal error or resource conflict. Refer to the Data Base Status Block subsection later in this section for more information.

DB$VERS Routine

The DB$VERS routine, shown in figure 2-17, can be used to change versions associated with a COBOL application program during execution of the run-unit; the routine can be executed several times within the same run-unit. The DB$VERS routine is not allowed within a transaction. Before each execution of DB$VERS, all of the subschema realms associated with the program must be closed or a fatal error results.

```
ENTER "DB$VERS" USING version-name, err-code
```

Figure 2-17. DB$VERS Routine Format

At the beginning of execution of any COBOL run-unit, the use of version MASTER is assumed. Thus, if alternate data base versions are not being used (DB$VERS is not specified), the data base files included in version MASTER are used. The permanent files of version MASTER are always attached for the run-unit by CDCS at the beginning of execution, even if the DB$VERS routine is the first executable statement in the run-unit. Therefore, even though a run-unit uses only alternate versions, all the permanent files included in version MASTER must be available to CDCS at the time of run-unit execution.

In the USING phrase, version-name specifies the 1- to 7-character version name described in the master directory for the schema being used. Version-name can be specified as either a nonnumeric literal or a word-aligned (synchronized left) data name described in its Data Description entry as PICTURE X(7). (MASTER is a valid version name.)

Err-code receives the appropriate 3-digit integer CDCS error code if an error occurs during the execution of the DB$VERS routine. Err-code must be described as USAGE IS COMPUTATIONAL-1 in its Data Description entry.

The user must ensure that all of the parameters of this routine are specified. Omission of any parameter causes indeterminate results.

Figure 2-18 shows an example of a COBOL program using the DB$VERS routine. The program begins by using version MASTER. The DB$VERS routine is specified to change the version used to BRANCH1.

```
PROCEDURE DIVISION.
     OPEN I-O PRODUCTS.
     .
     .
     .
     CLOSE PRODUCTS.
     ENTER "DB$VERS" USING "BRANCH1", ERROR-NO
     OPEN I-O PRODUCTS
     .
     .
     .
     CLOSE PRODUCTS.
     STOP RUN.
```

Figure 2-18. COBOL Example Showing Use of Version Master and an Alternate Version

DB$WAIT Routine

The DB$WAIT routine, shown in figure 2-19, can be used to force a deadlock situation to occur when CDCS is running as the CDCS Batch Test Facility (CDCSBTF). Thus, the code that deals with recovery from a deadlock can be tested. The DB$WAIT routine should be called only if more than one run-unit is being run with CDCSBTF. When a run-unit issues a call to DB$WAIT, execution of that run-unit is suspended until all other user run-units have either issued calls to DB$WAIT or completed execution. Once all calls have been issued, the run-units are released from suspension. The DB$WAIT routine can be requested from within a transaction.

```
ENTER "DB$WAIT"
```

Figure 2-19. DB$WAIT Routine Format

DELETE Statement

The DELETE statement, shown in figure 2-20, removes a record from a realm. This format is a subset of the DELETE statement shown in the COBOL reference manual.

```
DELETE realm-name RECORD
     [; INVALID KEY imperative-statement]
```

Figure 2-20. DELETE Statement Format

Before an attempt is made to delete a record, the record must be locked either with a record lock or with a realm lock. A protected record lock is established when a READ statement is issued for a realm opened for I-O. A protected record lock can be established outside of a transaction or within a transaction. An exclusive record lock is established when the delete operation is performed within a transaction. A realm lock is established by entering either the C.LOK routine or the DB$LKAR routine. The realm lock overrides the record lock.

The recommended procedure for deleting a record is to precede the DELETE statement with a READ statement. When the DELETE statement executes, the record is deleted. The value of the primary key should not be changed between the read and the delete of the record, because a record is identified by the primary key value. This procedure ensures that the record deleted is the correct record.

This procedure must be followed when deleting a record with a protected record lock outside of a transaction. When this procedure is followed, CDCS checks the value of the primary key to ensure that it has not changed since the read. If the value of the primary key has changed, CDCS issues a diagnostic message and ignores the delete request.

This procedure should be followed when deleting a record within a transaction or when deleting a record protected by a realm lock. If the value of the primary key is changed before the delete operation, the wrong record could inadvertently be deleted.

Refer to section 5 for information about both exclusive record locks and realm locks.

In the statement, realm-name specifies the realm from which the record is removed. The realm must have been included in the subschema.

The imperative statement of the INVALID KEY phrase is executed when the record to be deleted (as specified by the primary key) is not found in the realm.

OPEN Statement

The OPEN statement, shown in figure 2-21, prepares a realm or a relation for processing. The OPEN statement must be successfully executed prior to execution of any other statement (except USE FOR ACCESS CONTROL) that references a realm or the realms associated in a relation. The OPEN statement is not allowed within a transaction.

The OPEN realm statement (format 1) prepares a realm for processing and positions the file at the beginning of information. This statement is a subset of the format 1 OPEN statement found in the COBOL reference manual. Realm-name must be described in the subschema. A specific realm can be opened more than once by a run-unit as long as an intervening CLOSE statement precedes a subsequent OPEN statement.

The OPEN relation statement (format 2) opens all realms that are associated in a specified relation. The statement is performed as if a separate open were executed for each realm in the order of the rank of the realms in the relation. Relation-name must be included in the subschema. Relations are normally opened for input only, but can be opened for I-O if locking of records read is desired or if updating of the individual realms is desired. Realms opened by an OPEN relation statement should not be explicitly opened by an OPEN realm statement.

When the OPEN statement executes successfully, the key of reference is the primary key of the first record in the realm (in the root realm for an OPEN relation statement). The FILE STATUS data item, if any, is updated. Successful execution of the OPEN statement makes the record associated with the value of the primary key available to the run-unit; it does not obtain or release a data record.

One of the keywords INPUT, OUTPUT, or I-O must be specified for the OPEN statement. These keywords specify the type of processing allowed for a realm as follows:

When a realm is opened for INPUT, only the READ and CLOSE statements, and the C.LOK, C.UNLOK, and DB$LKAR routines can be executed.

When a realm is opened for OUTPUT, only the WRITE and CLOSE statements, and the C.LOK, C.UNLOK, and DB$LKAR routines can be executed.

When a realm is opened for I-O, the READ, WRITE, REWRITE, DELETE, and CLOSE statements, and the C.LOK, C.UNLOK, DB$LKAR routines can be executed.

Opening a realm for OUTPUT is valid only for creation of a new realm; it is not valid for an existing realm.

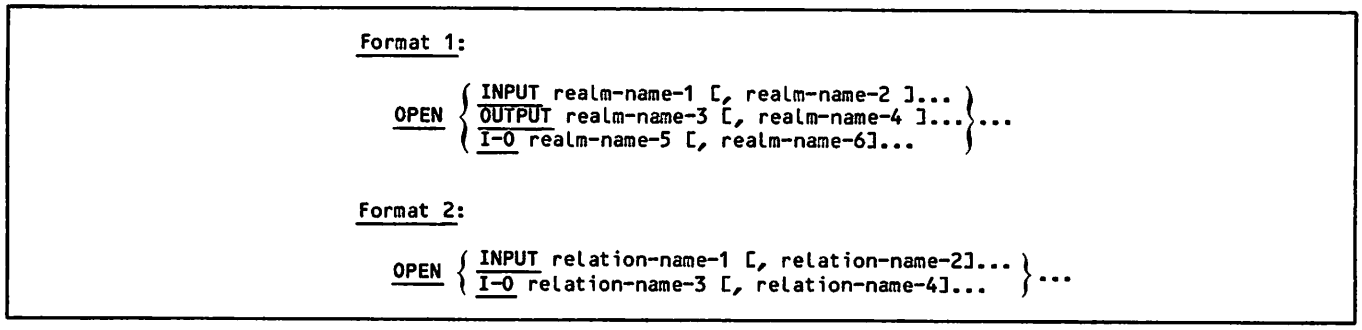


Figure 2-21. OPEN Statement Format

READ Statement

The READ statement, shown in figure 2-22, makes a record available to the application program from either a specified realm or from the realms associated with a specified relation. The realm or realms must be opened for INPUT or I-O.

```
Format 1:

READ realm-name [NEXT] RECORD [INTO identifier]
  [; AT END imperative-statement]

Format 2:

READ realm-name RECORD [INTO identifier]
  [; KEY IS data-name]
  [; INVALID KEY imperative-statement]

Format 3:

READ relation-name [NEXT] RECORD
  [; AT END imperative-statement]

Format 4:

READ relation-name RECORD
  [; KEY IS data-name]
  [; INVALID KEY imperative-statement]
```

Figure 2-22. READ Statement Format

The READ realm statement (formats 1 and 2) can be used to retrieve a logical record from a realm. Realm-name must be included in the subschema.

A realm can be read either randomly or sequentially. A realm can be read sequentially by the format 1 READ realm statement. This format of the READ statement retrieves the next logical record with the same key of reference from the realm. A realm can be read randomly by the format 2 READ realm statement. This format of the READ statement retrieves the record indicated by the value of the key from the specified realm; it can also be used to establish the key of reference and to position a realm that is to be subsequently read sequentially.

The READ relation statements (formats 3 and 4) can be used to retrieve a record from each of the realms associated with a specified relation. Relation-name must be included in the subschema.

A relation can be read either sequentially or randomly. The format 3 READ relation statement reads a relation sequentially. The format 4 READ relation statement reads a relation randomly. When either of the READ relation statements executes sucessfully, a record from each realm in the specified relation is available to the COBOL program. (Refer to section 5 for more information about reading relations.)

The NEXT phrase is for a sequential read of either a realm or a relation. The presence of this phrase in the READ realm statement (format 1) or the READ relation statement (format 3) specifies that the next record or relation occurrence with the same key of reference is to be retrieved. The next record or relation occurrence is determined by the collating sequence defined in the schema for the realm or for the root realm of a relation. (The data administrator can provide this information.) The collating sequences are shown in appendix I.

The INTO phrase applies to the READ realm statements (formats 1 and 2). This phrase moves the record being read from the user's work area associated with the specified realm to the storage area specified by identifier. The record is available in both places after READ execution. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before the record is moved to identifier. Identifier is blank-filled if the size of the record is less than the size of identifier.

The AT END phrase is for a sequential read of either a realm or a relation. In the READ realm statement (format 1) or the READ relation statement (format 3), this phrase specifies the imperative statement to be executed when an at-end condition occurs. The condition occurs when no next record exists. The program is responsible for subsequent actions if realm access is to continue. Realms must be repositioned according to the subsequent processing desired.

The INVALID KEY phrase is for a random read of either a realm or relation. In either the READ realm statement (format 2) or the READ relation statement (format 4), this phrase specifies the imperative statement to be executed if an invalid key condition occurs. For a READ realm statement, an invalid key condition occurs when an existing record does not have the value specified by the key of reference for a realm. For a READ relation statement, an invalid key condition occurs when no existing record in the root realm of the specified relation has the value specified by the key of reference.

The KEY IS phrase is used for a random read of either a realm or a relation. In either the READ realm statement (format 2) or the READ relation statement (format 4), the key of reference is identified by the presence or absence of the KEY IS phrase. If the KEY IS phrase is omitted, the key of reference is the primary key. If the phrase is specified, data-name must be set to the value of either the primary or alternate key of the realm (the root realm in a relation) before execution of the READ statement. If a key is specified in a READ relation statement for a realm in the relation other than the root realm, the COBOL compiler issues a diagnostic message.

When a random read of a realm or a relation is requested, the desired record is located when the comparison specified by the relational operator of the KEY IS phrase is performed. The record returned is the first record in the realm (the root realm of a relation) that satisfies the specified comparison. The comparison is performed by numeric value. The value of either the primary or the alternate key is compared to the value specified in data-name. The values are determined by the collating sequence

defined for the realm (the root realm of a relation) in the schema. (The data administrator can provide this information.) Collating sequences are shown in appendix I.

When a sequential read of a realm or a relation is requested, the key of reference is established by a preceding OPEN statement or by the value of the data-name specified with a preceding START statement. A preceding random read of the realm or relation can also be used to establish the key of reference.

REWRITE Statement

The REWRITE statement, shown in figure 2-23, replaces an existing record in a realm. The realm must be opened for I-O.

```
REWRITE record-name [FROM identifer]
      [; INVALID KEY imperative-statement]
```

Figure 2-23. REWRITE Statement Format

Before a record is rewritten, the record must be locked either with a record lock or with a realm lock. A protected record lock is established when a READ statement is issued for a realm opened for I-O. A protected record lock can be established either within a transaction or outside of a transaction. An exclusive record lock is established when the rewrite operation is performed within a transaction. A realm lock is established by entering either the C.LOK routine or the DB$LKAR routine. The realm lock overrides the record lock.

The recommended procedure for rewriting a record is to precede the REWRITE statement with a READ statement. The value of each data item being changed can then be set to the new value. When the REWRITE statement executes, the record is rewritten. The value of the primary key should not be changed between the read and the rewrite, because a record is identified by the primary key value.

This procedure must be followed when rewriting a record that is locked by a protected record lock outside of a transaction. In this situation when an attempt is made to rewrite a record with a change in the value of the primary key, CDCS issues a diagnostic message. If the primary key value of a record with a protected record lock must be changed, the program must delete the record with the old primary key value and write the record with the new primary key value.

This procedure should be followed when rewriting a record with an exclusive record lock within a transaction or when writing a record with a realm lock. If this procedure is not followed in these circumstances, the record being rewritten might inadvertently overwrite another record.

Refer to section 5 for information about both the exclusive record lock and the realm lock.

In the REWRITE statement, record-name must be a record name associated with a realm. Record-name must be included in the subschema.

The FROM phrase of a REWRITE statement causes the statement to have the same effect as execution of the following:

```
MOVE identifier TO record-name
REWRITE record-name
```

The contents of the record prior to the implicit move has no effect on REWRITE execution.

The INVALID KEY phrase specifies the imperative statement that is executed whenever an invalid key condition exists. When an invalid key condition occurs, the update operation does not take place and the data in the existing record is not affected.

START Statement

The START statement, shown in figure 2-24, logically positions a realm or a relation for subsequent retrieval of records. This statement can also be used to reposition a realm when end-of-information has been reached. Before the START statement is executed, the realm or relation must have been opened for INPUT or I-O.

Successful execution of the START statement establishes a key of reference for purposes of future access to the realm. The key of reference can be either the primary key or the alternate key of the record. Once the key of reference has been established, it can only be changed by execution of another START, READ, or OPEN statement, or an INVALID KEY phrase.

The START realm statement (format 1) establishes the key of reference for a realm by positioning the realm to the first record whose key value meets the specified condition. The search begins either from the current key of reference or from the beginning of the realm. The key that satisfies the condition becomes the new key of reference.

The START relation statement (format 2) positions a relation for subsequent sequential retrieval of records through relational reads. The START relation statement positions the root realm as described in the preceding paragraph. This statement establishes the key of reference for the root realm of the relation. A sequential relation read following a START relation statement retrieves from the root realm the record with the key that satisfies the start condition and a record from each realm in the associated relational hierarchy.

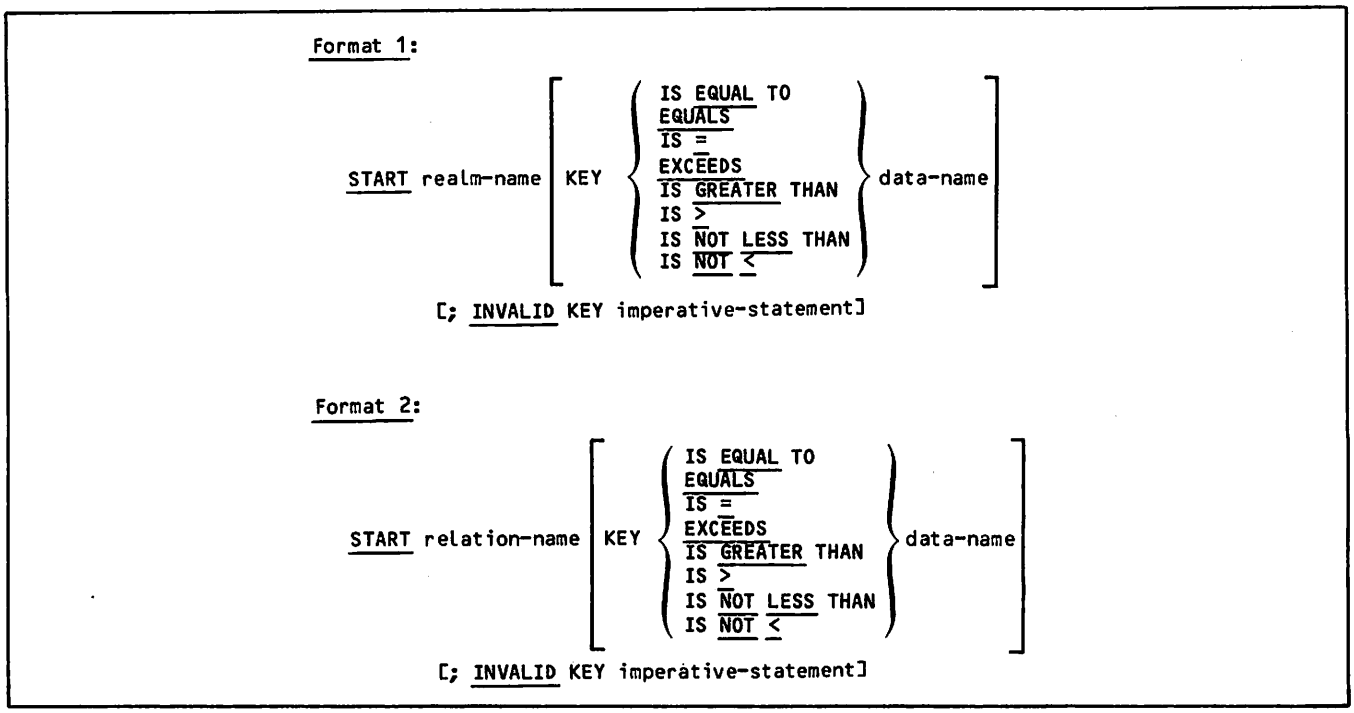


Figure 2-24. START Statement Format

The KEY phrase specifies the data item to be compared with the keys of records existing in the realm (the root realm for a relation). The data-name in the KEY phrase must be set to the appropriate value of the primary, alternate, or major key of the realm (root realm for a relation) before execution of the START statement. The key of reference is determined by the KEY phrase as follows:

If the KEY phrase is omitted, the key of reference is the primary key. (However, the phrase cannot be omitted for realms which are direct access and actual key files.)

If data-name is a primary key, the key of reference is that primary key. (However, the primary key can be specified only for indexed sequential files.)

If data-name is an alternate record key, the key of reference is that alternate record key.

If data-name is a major key (the leading portion of a group item that is a primary or alternate record key), the key of reference is that record key.

The type of comparison is specified by the relational operator in the KEY phrase. When the key is alphanumeric and the operands are of unequal size, comparison proceeds as though the longer operand were truncated on the right, making its length equal to that of the shorter operand. All other COBOL numeric or nonnumeric comparison rules apply.

The values used for comparison are determined by the collating sequence specified for the realm (the root realm of a relation) in the schema. (The data administrator can provide this information.) The collating sequences are shown in appendix I.

The INVALID KEY phrase specifies the imperative statement that is executed if the comparison is not satisfied by any record in the realm or relation. In this instance, the key of reference is undefined.

USE FOR ACCESS CONTROL Declarative Statement

The USE FOR ACCESS CONTROL statement, shown in figure 2-25, supplies the access control key required to gain access to the specified realms. Access control locks can be defined by the data administrator to provide privacy at the realm level. If the realm is defined with an access control lock in the schema, this statement must supply the access control key before the realm can be opened. The USE FOR ACCESS CONTROL statement must be the first sentence after a section header within the Declaratives portion of the Procedure Division. This statement is not allowed within a transaction.

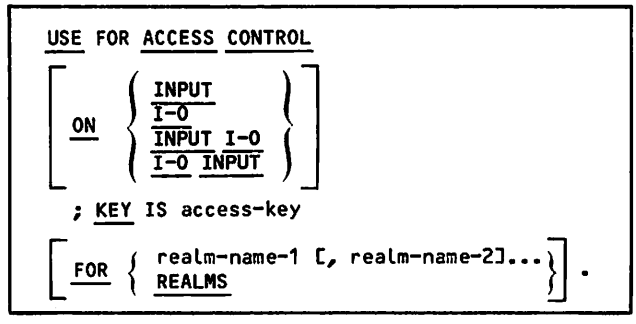


Figure 2-25. USE FOR ACCESS CONTROL Declarative Statement

Each USE FOR ACCESS CONTROL procedure is executed once: at the start of program execution and before any realm is opened. CDCS compares the key supplied by the KEY IS phrase with the lock specified for the realm in the schema. If a program attempts to open a realm without supplying the correct key, the program is terminated. The type of access allowed when the correct key is supplied depends on the option chosen for the ON phrase. (The option chosen must agree with the type of access provided by the data administrator.)

The program must include as many access control procedures as required to provide the necessary access control keys. To access several realms, each requiring a separate access control key, the program must contain a USE FOR ACCESS CONTROL procedure for each realm. If the data administrator has defined separate access control keys for reading and updating of the records in a realm, the COBOL program must specify two USE FOR ACCESS CONTROL procedures to permit the realm to be opened for input/output. One statement must specify ON INPUT; the other statement must specify ON I-O. Table 2-2 shows the ON phrase option that satisfies the access control locks specified by the data administrator in the schema. The USE FOR ACCESS CONTROL statement must be compatible with the mode option of the OPEN statement.

TABLE 2-2. SCHEMA ACCESS CONTROL LOCK AND CORRESPONDING ON PHRASE USAGE OPTION

Schema Access Control Lock	ON Phrase Option
Update	I-O
Retrieval†	INPUT
Update/Retrieval†	INPUT I-O I-O INPUT

†Retrieval refers to read only access.

Options for the ON phrase are as follows:

INPUT

Read only access

I-O

Update only access

INPUT I-O or
I-O INPUT

Read and update access

INPUT I-O and I-O INPUT are synonymous and interchangeable. If the ON phrase is not specified, ON INPUT I-O is assumed.

In the KEY phrase, access-key identifies the data item containing the access control key required to gain the specified access to the realm. Access-key must be set to the value of the access control key. (The access control key required to obtain the specified access is defined in the schema, can be 1 to 30 characters in length, and should be provided by the data administrator.) Access-key must be specified as a 30-character word-aligned (synchronized left) alphanumeric item described in its Data Description entry as PICTURE X(30). When the USE FOR ACCESS CONTROL procedure completes execution, the value of access-key is passed to CDCS.

The FOR phrase specifies the realm to which the access control key applies. Any realm-name specified must also appear in the subschema. If REALMS is specified in the FOR phrase, CDCS grants access to all realms named in the subschema that have lock values equal to the access control key value. If the FOR phrase is omitted, FOR REALMS is assumed.

Figure 2-26 shows two USE FOR ACCESS CONTROL procedures. The procedures establish the right of the program to access PART-REALM for read only operations, and to access ORDER-REALM for both read and update operations. The access control key is the same for both realms.

```
WORKING-STORAGE SECTION.
77 ACCESS-KEY            PICTURE X(30).
      .
      .
      .
PROCEDURE DIVISION.
DECLARATIVES.
ACCESS-CONTROL-1 SECTION.
   USE FOR ACCESS CONTROL ON INPUT
      KEY IS ACCESS-KEY FOR PART-REALM.
PAR-M.       MOVE "AX12345" TO ACCESS-KEY.
ACCESS-CONTROL-2 SECTION.
   USE FOR ACCESS CONTROL ON INPUT I-O
      KEY IS ACCESS-KEY FOR ORDER-REALM,
         ACCOUNT REALM.
PAR-N.       MOVE "R-W1000" TO ACCESS-KEY.
END DECLARATIVES.
      .
      .
      .
```

Figure 2-26. USE FOR ACCESS CONTROL Procedure Example

USE FOR DEADLOCK Declarative Statement

The USE FOR DEADLOCK statement, shown in figure 2-27, identifies the procedure to be executed when a deadlock situation occurs. A deadlock situation occurs when two or more run-units, each holding locks on records or realms, try to access a locked realm or a locked record in such a way that no run-unit is able to proceed unless another run-unit's locks are released. The USE FOR DEADLOCK statement must be the first sentence after a section header within the Declaratives portion of the Procedure Division.

```
USE FOR DEADLOCK

   ON { realm-name-1 [, realm-name-2]... } .
      { REALMS                           }
```

Figure 2-27. USE FOR DEADLOCK Declarative Statement

The USE FOR DEADLOCK procedure should be written so that the points in the program where deadlock might occur are recognized. See section 5 for more information on locking and deadlock. Normally, this procedure should cause the input/output statements that locked the records or realms to be repeated.

If the USE FOR DEADLOCK statement is omitted for a given realm and a deadlock situation occurs for that realm, the run is aborted.

In the statement, realm-name specifies the name of the realm for which the deadlock situation occurs. Any realm-name specified must also appear in the subschema.

If REALMS is specified, the USE FOR DEADLOCK procedure is executed when a deadlock situation occurs involving any realm named in the subschema.

Figure 2-28 illustrates a possible coding sequence for the USE FOR DEADLOCK procedure. The C.LOK, DB$LKAR, and C.UNLOK routines cannot be called from the USE procedure directly because using these routines could result in another deadlock situation.

```
IDENTIFICATION DIVISION.
      .
      .
      .
DATA DIVISION.
FILE SECTION.
      .
      .
      .
01  DEAD-FLAG                    PIC 9.
      .
      .
      .
PROCEDURE DIVISION.
DECLARATIVES.
ADEADLOCK SECTION.
    USE FOR DEADLOCK ON REALMS.
BEGIN-DEADLOCK.
    MOVE 1 TO DEAD-FLAG.
      .
      .
      .
MAIN-LOGIC-SECTION.
START-UP.
    PERFORM START-UP.
    PERFORM READ-PROJ WITH TEST AFTER
       UNTIL DEAD-FLAG IS EQUAL TO ZERO.
    PERFORM READ-PROD WITH TEST AFTER
       UNTIL DEADFLAG IS EQUAL TO ZERO.

READ-PROJ SECTION.
READ-PROJECT.
    MOVE ZERO TO DEAD-FLAG.
    MOVE "M130001560" TO PROJECT-ID IN
       PROJECT.
    READ PROJECT KEY IS EQUAL TO PROJECT-ID,
       INVALID KEY PERFORM ERRPROC-1.
      .
      .
      .
READ-PROD.
READ-PRODUCTS.
    MOVE ZERO TO DEAD-FLAG.
    MOVE "826NAMW019" TO PRODUCT-ID.
    READ PRODUCTS KEY IS EQUAL TO PRODUCT-ID,
    INVALID KEY PERFORM ERRPROC-1.
```

Figure 2-28. USE FOR DEADLOCK Procedure Example

WRITE Statement

The WRITE statement, shown in figure 2-29, uses the current value of all the data items defined for the record in the subschema to construct a record, and then writes the record to the data base file associated with the realm. The realm must be opened in OUTPUT or I-O mode. All primary and alternate key values must be set appropriately before the record is written. Any data items in the schema record that are not defined in the subschema are given null values before the record is written to the data base. Format 1 of the WRITE statement, which appears in the COBOL reference manual, does not apply to data base processing.

In the statement, record-name specifies the record to be written. The record must be specified in the subschema.

The FROM phrase of the WRITE statement is equivalent to execution of the following two statements:

```
MOVE identifier TO record-name
WRITE record-name
```

Information in identifier remains available to the program.

The INVALID key phrase specifies the imperative statement that executes when an invalid key condition occurs.

```
Format 2:

    WRITE record-name [FROM identifier]

        [; INVALID KEY imperative-statement]
```

Figure 2-29. WRITE Statement Format

COBOL SUBPROGRAMS

COBOL subprograms can access data base realms. Both the main program and the subprogram must include the same SUB-SCHEMA clause in the SPECIAL NAMES paragraph of the ENVIRONMENT Division. CDCS automatically attaches all files (realms) named in the subschema specified in the SUB-SCHEMA clause.

Neither the main program nor the subprogram need specify File-Control entries for realms referenced in the run-unit unless the FILE STATUS clause is used to obtain non-CDCS error information associated with the realms. If the FILE STATUS clause is used, it must be declared in the Common-Storage Section of the main program and any subprograms that use FILE STATUS for detection of errors.

When updated status information is desired for operations performed in a subprogram on a realm or a relation, the data base status block must be declared in the Common-Storage Section of both the main program and the subprogram. The DB$DBST routine, however, need only be entered once from the main program. For detailed information about Data Description entries in the Common-Storage Section, refer to the COBOL reference manual.

COMPILATION AND EXECUTION

Compilation of a COBOL application program for processing with CDCS requires that the subschema be attached. The D parameter on the compiler call must specify the local file name of the file (usually the subschema library) on which the subschema directory resides. Data base information is incorporated in the application program during compilation. During compilation, the LO=M parameter of the COBOL5 control statement can be used to obtain a listing of the subschema.

Compilation and execution of a COBOL application program is initiated by control statements and is performed in sequential steps. To compile and execute a COBOL program that utilizes CDCS, the following steps must be performed:

To compile

Attach the subschema library.

Specify the COBOL5 control statement and include the D parameter to indicate the local file name of the file on which the subschema directory resides.

To execute

Specify the file containing the relocatable binary program; LGO is the system default file.

A COBOL program that utilizes CDCS could be compiled with the following COBOL5 control statement:

```
COBOL5,D=SSLIBRY.
```

The D parameter of the control statement specifies SSLIBRY, the file on which the subschema directory resides.

Refer to section 5 for more information about execution of a COBOL program that utilizes CDCS. Refer to appendix G for information about compilation and execution of an application program using the CDCS Batch Test Facility (CDCSBTF).

RECOMPILATION GUIDELINES

Recompilation of an application program using a subschema is governed by the types of changes made to the schema. The checksum facility is the mechanism used to determine whether the changes made to the schema require the recompilation of a subschema or an application program or both. The DDL compiler produces a one-word identifying bit string, called a checksum, for each realm and area in the schema. A checksum is also generated for each subschema. The checksums are stored in the schema and subschema directories and are recorded in the master directory. The checksums are used to verify the consistency of the schema and its associated subschemas, as well as the consistency of the subschema and the application program referencing it.

When the schema is recompiled, the data administrator compares the checksums generated with the corresponding checksums in the previously generated image for the schema. If the checksums do not match, all subschemas that reference the areas or relations whose checksums have changed must be recompiled by the data administrator.

If the subschema has been recompiled and the checksum of the recompiled subschema differs from its previous checksum, a COBOL program referencing that subschema must be recompiled. If a COBOL program references an invalid checksum, CDCS aborts the program and issues a diagnostic message. The COBOL programmer can prevent the abnormal termination of a program by ensuring, prior to program execution, that the subschema checksum in the master directory matches the checksum of the subschema used to compile the program.

STATUS CHECKING

CDCS provides various mechanisms by which extensive status information can be returned to a COBOL program. CDCS can return status information and error codes in the data base status block if the DB$DBST routine is entered. In order to provide compatibility with earlier versions of CDCS, the COBOL program can still call C.DMRST and C.IOST to determine the status of CDCS operations.

DATA BASE STATUS BLOCK

The COBOL application program can provide CDCS with a group item or a table to which data base status information can be returned. The data base status block is updated automatically after every CDCS operation. The following information is returned to the data base status block:

CDCS or CYBER Record Manager error codes

Subschema item ordinal for item level error

CRM code indicating the file position of a realm

Function being executed when an error occurred

Rank in a relation of the realm on which either a CRM or CDCS error occurred, or a special relation condition (control break, or a null record) occurred

Name of the realm in which an error occurred

The COBOL application program communicates the location and length of the data base status block to CDCS by a call to the CDCS routine DB$DBST, explained earlier in this section.

Data base status block error codes and file position codes are returned to the user as decimal values. The user must convert to octal those values returned that are CRM codes, in order to correlate the code with those listed in the CRM reference manual. Error codes 384 through 447 and codes 472 through 511 indicate CDCS errors; these codes and corresponding message are listed in appendix B. Refer to the CRM Advanced Access Methods reference manual for CRM codes and messages.

The length of the data base status block is specified in words. One word provides space for either a 10-character DISPLAY item or a COMPUTATIONAL-1 item. The data base status block must be from 1 to 11 words in length. CDCS returns as much information as possible in the given length.

The information returned in the data base status block is shown in table 2-3.

The length of the data base status block is variable; the length can range from 1 to 11 words. As a minimum, one word must be provided for the error code. The other items are optional.

An example of a COBOL description of the data base status block is shown in figure 2-30. Each elementary item of DATA-BASE-STATUS-BLOCK corresponds to a word of memory in which particular information is returned, except DB-REALM which corresponds to three words.

The specification of the data base status block must adhere to the following list of rules (the rules reference the description of the data base status block shown in figure 2-30):

The items must be defined with the length shown and in the order shown.

For any particular item that is defined, all items that precede it must also be defined.

COMPUTATIONAL-1 must be used to define the numeric items as indicated.

For information to be returned to any elementary item of the group items AUXILIARY-STATUS or RELATION-RANK-STATUS, length must be provided for all three items of the particular group.

The database status block must be declared in common storage if it is used in a subprogram.

CDCS only updates those words contained within the given length of the status block. If, for instance, the COBOL program wants the data base status block to include only the items DATABASE-STATUS and DB-ITEM-ORDINAL, the user must specify a length of four words.

ADDITIONAL STATUS CHECKING ELEMENTS

Additional status checking elements are available to the COBOL program. CRM error codes can be obtained by entering the C.IOST routine described earlier in this section. The C.DMRST routine, also described earlier in this section, can be entered after a relation read to determine the status of the realms in the relation.

In addition to these routines, the data item specified in the FILE STATUS clause in a COBOL program can be used to check for CRM errors and end-of-file status relating to a realm. The USE AFTER STANDARD ERROR procedure can be used for checking the status of input/output operations involving realms. Refer to the COBOL reference manual for further details on COBOL error processing.

PROGRAM DEBUGGING

The automatic generation by COBOL of SELECT statements for subschema realms poses special debugging problems. The local file name from the subschema FIT is used for implementor-name-1 and implementor-name-2 in the generated SELECT statements. Implementor-name-1 is the first 7 characters of the realm name. Implementor-name-2 is the file-name specified on the XN parameter of the FILE control statement used in describing the realm during schema compilation. The data administrator can provide this name. These local file names must not appear as user-defined words elsewhere in the COBOL program or subschema description. If they do, fatal errors are generated.

Errors occurring on the code for the generated SELECT statements have line and column numbers pointing to the subschema name in the SUB-SCHEMA clause.

```
                                                              Word
01  DATABASE-STATUS-BLOCK.
    02 DATABASE-STATUS              PICTURE 9(5) USAGE IS COMP-1.    1
    02 AUXILIARY-STATUS.
       03 DB-ITEM-ORDINAL           PICTURE 9(5) USAGE IS COMP-1.    2
       03 DB-FILE-POSITION          PICTURE 9(3) USAGE IS COMP-1.    3
       03 DB-SEV-CODE               PICTURE 9(3) USAGE IS COMP-1.    4
    02 DB-FUNCTION                  PICTURE A(10).                   5
    02 RELATION-RANK-STATUS.
       03 DB-REL-RANK-ERROR         PICTURE 9(3) USAGE IS COMP-1.    6
       03 DB-REL-RANK-CTLBK         PICTURE 9(3) USAGE IS COMP-1.    7
       03 DB-REL-RANK-NULL          PICTURE 9(3) USAGE IS COMP-1.    8
    02 DB-REALM                     PICTURE X(30).                   9,10,11
```

Figure 2-30. Example of a COBOL Description of a Data Base Status Block

TABLE 2-3. CONTENTS OF DATA BASE STATUS BLOCK

Word Number	Contents
1	CRM or CDCS error code in decimal for the last data base operation on a realm or relation; value is zero if no error has occurred. Values 384 through 447 and values 472 through 511 are CDCS errors; all others are CRM errors. Word 1 should be specified as a COMPUTATIONAL-1 data item. Note that only error codes are returned. Null occurrence and control break conditions often arise in the course of normal processing. The status codes for these conditions are not returned.
2	Subschema item ordinal for CDCS item-level errors. Item-level errors include data validation errors, record mapping errors, and item-level data base procedure errors. Value is zero if no error has occurred. The item ordinal assigned by the DDL compiler is identified on the subschema compilation listing. Word 2 should be specified as a COMPUTATIONAL-1 data item.
3	CRM code in decimal indicating file position of the realm when the last data base operation was performed. A file position code is returned when open, close, read, and start operations are performed. For a relation operation, the file position code indicates the position of the root realm when the last operation was performed. Word 3 should be specified as a COMPUTATIONAL-1 data item. The following list includes the file position codes that most commonly occur during data base processing: 8 End-of-key-list, which occurs when the last primary key value associated with a given alternate key (for which duplicate values exist) has been returned during a read operation using an alternate key value. 16 End-of-record, which occurs when a record has been returned during a read operation. 64 End-of-information, which occurs when a sequential read operation is attempted after the previous read operation returned the last record in the file.
4	Severity of error that occurred during execution of the last data base operation: the value is zero if no error or a nonfatal error has occurred; the value is one if a fatal error has occurred. Word 4 should be specified as a COMPUTATIONAL-1 data item.
5	Function being performed when an error or relation condition occurred; one of the following character strings: ASK, BEGIN, CLOSE, COMMIT, DELETE, DROP, END, GET-ID, LOCK, LOCK-AREA, OPEN, PRIVACY, RAN-READ, RECVR-PNT, REL-NEXT, REL-READ, REL-START, REWRITE, SEQ-READ, START, UNDEFINED, UNLOCK, VERSION, WRITE Value is undefined if no error has occurred. Word 5 should be specified as PICTURE X(10) or PICTURE A(10).
6	For a relation operation, the rank of the realm on which a CRM or CDCS error occurred; zero if no error has occurred. (The root realm of the relation has a rank of one.) An error on a realm during a relation read terminates the operation. Consequently, there is never more than one rank in the relation which has a CRM or CDCS error. Word 6 should be defined as a COMPUTATIONAL-1 data item.

TABLE 2-3. CONTENTS OF DATA BASE STATUS BLOCK (Contd)

Word Number	Contents
7	For a relation operation, the lowest rank on which a control break occurred; value is zero if no control break has occurred. The control break condition signifies that a new record occurrence was read for the realm's parent realm in a relation. If the null occurrence condition is set for a realm, the control break condition can be assumed for that realm even though it is not set. If a control break occurs, all realms in the relation with a rank greater than the rank recorded in this word also have control break status (or a null record occurrence, since null record occurrences override control break status). See section 5 for more information about control break conditions. Word 7 should be specified as a COMPUTATIONAL-1 data item.
8	For a relation operation, the lowest rank for which there was a null record; value is zero if no null record. A null record occurrence for a specific rank means either that at this rank no child record occurrences of the parent record passed the record qualification criteria or that no child record occurrences at this rank exist for the parent record. If a null record occurs, all realms in the relation with a rank greater than the rank recorded in this word have null record occurrences. See section 5 for more information about null record occurrences. Word 8 should be specified as a COMPUTATIONAL-1 data item.
9,10,11	The display code name of the realm in which an error has occurred. Contains blanks if no error has occurred, or if the error has occurred on a non-I/O (input/output) operation or an I/O operation not explicitly requested by the COBOL program. Words 9, 10, and 11 should be defined as PICTURE X(30).

FORTRAN 5 application programs can be used to access and manipulate a data base controlled by CYBER Database Control System (CDCS). FORTRAN Data Manipulation Language (DML) statements that are coded within a FORTRAN program provide for data base access. The DML statements are translated by the DML preprocessor, which generates and inserts into the program the appropriate calls to CDCS to request input/output processing at execution time.

The FORTRAN interface with CDCS consists of the FORTRAN subschema, the DML statements and routines used to access the data base, and the status checking elements available to a FORTRAN program that utilizes CDCS to access a data base.

In this section a basic knowledge of FORTRAN 5 is assumed. Detailed information regarding the use of the FORTRAN language is contained in the appropriate FORTRAN reference manual.

FORTRAN SUBSCHEMA

The DMS-170 data base files that are to be accessed by a FORTRAN application program must be described in a directory called a FORTRAN subschema. The data administrator, working with application programmers, is responsible for creating the subschema.

When the subschema is compiled, a listing is produced. The listing provides information required by the application programmer to code the FORTRAN application program. The data administrator should provide a compilation listing of the subschema for the application programmer.

Some information required to access data base files is not included in the subschema listing. The data administrator should provide any necessary additional information to the application programmer.

This subsection documents the information provided by the subschema listing and the additional information that the data administrator must provide when the information is necessary for data base processing.

SUBSCHEMA LISTING

The information provided by the subschema is indicated by the following alphabetic list of items. Not all items are included in every subschema; those items that always appear in the subschema listing are so indicated. An example of each item is pointed out in the sample subschema shown in figure 3-1.

Alias names

The ALIAS statement identifies the name of a data item, record, or realm used in the subschema in place of the name used in the schema. The name on the left of the equal sign (=) in the ALIAS statement is the name that must be referenced in a FORTRAN application program.

Alternate key

The notation ALTERNATE KEY indicates a data item that is an alternate record key. The realm for which the data item is an alternate key is also indicated.

Checksum

Always present. A checksum is a one-word attribute generated by the DDLF compiler for the subschema. The checksum of the subschema referenced by the FORTRAN application program is incorporated in the object program at compilation time and must agree at execution time with the checksum associated with the subschema in the master directory. Refer to the subsection Recompilation Guidelines for more information.

Concatenated key

Present if one is defined. A concatenated key is a primary or alternate key made up of several data items. The listing in the subschema shows the data items that make up a concatenated key. The concatenated key name and the list of data items that make up the key appear with the notation PRIMARY KEY or ALTERNATE KEY. The realm for which the key is defined is also indicated.

When a concatenated key is defined, major key processsing is possible. A major key is the leading data item or items of a concatenated key.

A concatenated key name can contain up to 30 characters and can include hyphens. The concatenated key name is referenced in the FORTRAN applications program by either the FORTRAN DML READ statement or START statement. The START statement can also reference leading data items of the concatenated key for major key processing. Major key processing is not allowed in a READ statement. Using a concatenated key and major key processing are discussed in the description of the READ and START statements later in this section.

```
1      COMPARE                * SOURCE LISTING *   (83207)  DDLF       83/09/14. 15.07.43.     PAGE     1

                                                                                    Subschema name
              00001                SUBSCHEMA COMPARE,SCHEMA=UNIVERSITY
              00002                                                                 Alias name
              00003                ALIAS(REALM)  PFILE=PROFESSOR
              00004                ALIAS(RECORD) PRECORD=PROF-REC
              00005                ALIAS(ITEM)   STDID=STUDENT-ID.CURR-REC
              00006                ALIAS(ITEM)   PROFID=PROF-ID.PROF-REC
              00007                ALIAS(ITEM)   PNAME=PROF-NAME
              00008
              00009                ALIAS(REALM)  CRSFILE=COURSE
              00010                ALIAS(RECORD) CRSREC=COURSE-REC
              00011                ALIAS(ITEM)   CRSID=COURSE-ID.COURSE-REC
              00012                ALIAS(ITEM)   CRSNAME=COURSE-NAME
              00013                ALIAS(ITEM)   PROF=PROF-ID.COURSE-REC
              00014                ALIAS(ITEM)   FIELD=ACADEMIC-FIELD
              00015
              00016                ALIAS(REALM)  CFILE=CURRICULUM
              00017                ALIAS(RECORD) CRECORD=CURR-REC
              00018                ALIAS(ITEM)   COURSE=COURSE-ID.CURR-REC
              00019                ALIAS(ITEM)   CODE=COMPLETE-CODE
              00020                ALIAS(ITEM)   DATE=COMPLETE-DATE
              00021
              00022                REALM PFILE                                   Realm name
              00023                REALM CRSFILE
              00024                REALM CFILE
              00025
              00026                RECORD PRECORD                                Record name
** WITHIN PFILE                                                                 Associated realm
              00027                CHARACTER*8 PROFID
** ORDINAL    1                                                                  Data items
              00028                CHARACTER*30 PNAME                            comprising
** ORDINAL    2                                                                  record PRECORD
              00029                CHARACTER*20 FIELD
              00030
** ORDINAL    3
              00031                RECORD CRSREC
** WITHIN CRSFILE
              00032                CHARACTER*6 CRSID                             Data item
** ORDINAL    1                                                                  Schema item ordinal
              00033                CHARACTER*20 CRSNAME                          for CRSID
** ORDINAL    2
              00034                CHARACTER*8 PROF
              00035
** ORDINAL    3
              00036                RECORD CRECORD
** WITHIN CFILE                                                                  Major key
              00037                CHARACTER*3 IDENT
** ORDINAL    1                                                                  Concatenated key
              00038                CHARACTER*11 STDID                            data items
** ORDINAL    2
              00039                CHARACTER*6 COURSE
** ORDINAL    3
              00040                CHARACTER*1 CODE
** ORDINAL    4
              00041                CHARACTER*8 DATE
** ORDINAL    5
              00042                REAL GRADE
              00043
```

Figure 3-1. Sample FORTRAN Subschema (Sheet 1 of 2)

```
      COMPARE              * SOURCE LISTING *   (83207)  DDLF       83/09/14. 15.07.43.     PAGE     2

** ORDINAL    6
               00044              RELATION REL3 <------------------------------------- Relation name
PRIMARY KEY    00027         PROFID FOR AREA PFILE <---------------------------------- Primary key
ALTERNATE KEY  00029         FIELD FOR AREA PFILE <---------------------------------- Alternate key
PRIMARY KEY    00032         CRSID FOR AREA CRSFILE
ALTERNATE KEY  00034         PROF FOR AREA CRSFILE
PRIMARY KEY    *****         CATKEY(IDENT,STDID) FOR AREA CFILE <-------------------- Concatenated key
ALTERNATE KEY  00038         STDID FOR AREA CFILE <---------------------------------- Alternate key
ALTERNATE KEY  00039         COURSE FOR AREA CFILE
ALTERNATE KEY  00042         GRADE FOR AREA CFILE
               *****         RECORD MAPPING IS NOT NEEDED FOR REALM -   PFILE
               *****         RECORD MAPPING IS NEEDED FOR REALM -   CRSFILE
               *****         RECORD MAPPING IS NEEDED FOR REALM -   CFILE
               00045               RESTRICT CRECORD (CODE .EQ. 'C') <---------------- Restriction
               00046               END                                                (relation)
               00047
               *****         END OF SUB-SCHEMA SOURCE INPUT

                                                                                      Relation
               *****               RELATION   STATISTICS                *****
RELATION 001                 REL3 JOINS          AREA - PFILE <---------------------- Rank 1
                                                 AREA - CRSFILE <------------------- Rank 2
                                                 AREA - CFILE <--------------------- Rank 3



               -----         BEGIN SUB-SCHEMA FILE MAINTENANCE                -----

          SUBSCHEMA                                CHECKSUM
          COMPARE                                  6675544551611453173O <------------ Checksum

               -----         END OF FILE MAINTENANCE                  -----
DDLF COMPLETE.          0 DIAGNOSTICS.
  51200B CM USED.     0.262  CP SECS.
```

Figure 3-1. Sample FORTRAN Subschema (Sheet 2 of 2)

Data item (name and description)

Always present. A type statement identifies the name and description of a data item. A type statement is included in the subschema listing for each data item that is available to the application program.

A data item of type CHARACTER has a default length of 1 character if no length is specified in the type statement.

Type statements in the subschema follow the same rules as type statements for FORTRAN programs; refer to the FORTRAN reference manual for more information about type statements. Data received in the working storage area of an application program is mapped according to the subschema description.

Major key

Present if a concatenated key is defined. Refer to concatenated key.

Primary key

Always present for each realm included in the subschema. The notation PRIMARY KEY indicates each data item that is a primary record key. The realm for which the data item is a primary record key is also indicated.

Ranks of the relation

The realms joined in a relation are listed by realm name in order of rank. The first realm listed has a rank of 1; the second, a rank of 2; and so forth.

Realm (file) names

Always present. A REALM statement specifies the realms available to the application program. Unless ALL is specified, only those realms included in the realm division are available to the application program.

The realm name that follows the notation WITHIN designates the realm associated with the record description that follows in the listing.

Record names, associated realms and data items

Always present. A RECORD statement identifies the record name. A RECORD statement is included in the subschema listing for each record available to the application program.

The notation WITHIN realm name identifies the particular realm associated with a record.

The type statements for the data items that make up the record always follow the RECORD statement in the subschema listing. Another RECORD statement, a RELATION statement, or an END statement in the subschema listing terminates the list of data items within the record.

Relation names

A RELATION statement identifies the name of a relation. A RELATION statement is included in the subschema listing for each relation available to the application program.

Restriction

A RESTRICT statement in the subschema listing identifies a restriction. If a restriction is placed on a relation, only records that meet all the qualification criteria specified in the RESTRICT statement in the subschema are returned to the application program when the relation is read.

If the RESTRICT statement contains a data item that is not included in the subschema, the data item must be defined in the program with the same type and length as the subschema item used for comparision. The data item must be set to a value before any read that uses the relation.

Schema name

Always present. The schema is identified in the subschema.

Subschema item ordinal

Always present. The subschema item ordinal is a unique identifier within a record that is assigned to each data item in a subschema when the subschema is compiled. A subschema item ordinal is used in conjunction with the data base status block.

Subschema name

Always present. A SUBSCHEMA statement identifies the subschema name. An application program must reference the subschema by using the subschema name.

INFORMATION PROVIDED BY DATA ADMINISTRATOR

It is the responsibility of the data administrator to provide the following information when it is required for data base processing:

Constraints

If constraints are defined in the schema and updates are likely to violate them, the application programmer should be provided with information about the constraints.

Join items of the relation

Files are joined in a relation by identical data items that exist in two files. The information about the join items is contained in the schema. Usually, an application programmer does not need to know the join items to use a relation. Often the programmer can determine the join items from the subschema listing. (Refer to section 5 for more information on relations.) However, in some situations, the data administrator should provide the application programmer with the join items.

Permanent file information for the subschema library

The subschema directory must be available for preprocessing of DML statements; therefore, the application programmer must be provided with the information required to attach the subschema library file that contains the subschema directory.

Privacy keys

If an area has been defined with an access control lock in the schema, a PRIVACY statement must be included in the FORTRAN application program to access the realm. The application programmer must be provided with all privacy keys required for data base access. (Refer to the PRIVACY statement subsection for further information.)

Requirements imposed by any data base procedure

If data base procedures are defined in the schema, the application programmer must be provided with information required by the data base procedures.

Transaction update limits

If limits have been imposed on the number of transactions allowed for all users of the schema, the application programmer should be provided with this information.

Version name

If alternate data base versions are defined for the data base, the application programmer must be provided with information about the name and use of the alternate data base versions.

SUBSCHEMA DIRECTORY

When the subschema source input is compiled, the object subschema, called the subschema directory, is generated. The subschema directory is usually included in a subschema library. The subschema directory must be available to the DML preprocessor for preprocessing the FORTRAN DML program; therefore, the subschema library must be attached for preprocessing.

During preprocessing of the FORTRAN DML program, the DML preprocessor inserts into the FORTRAN program the descriptions of all files, data items, and relations that are included in the subschema.

FORTRAN DML

FORTRAN DML is the execution time facility enabling data base access from a FORTRAN program. DML consists of a series of statements similar to FORTRAN statements that are included in a FORTRAN program and processed by the DML preprocessor prior to compilation of the program. DML translates the statements into FORTRAN specification statements and CALL statements, which can then be compiled like other FORTRAN statements. Data descriptions are obtained from the FORTRAN subschema directory. Once the program is preprocessed using the subschema, it can be compiled and executed later without reattaching the subschema. At execution time, CDCS is called to access the data base.

LANGUAGE ELEMENTS

FORTRAN DML statements consist of keywords, constants, variables, and operators. The following subsections explain the elements of DML statements.

Keywords

DML keywords are shown in upper case in the following subsections. These words identify statements and options within statements. Each statement begins with a specific keyword, and other keywords are used within statements. When a keyword is used, it must be specified exactly as shown in the particular statement format illustrated later in this section.

Constants

Data items can be specified as constants in DML statements. For FORTRAN 5, a character constant is used.

Variables

With a few exceptions, variables appearing in DML statements follow the rules for FORTRAN variables. These rules are defined in the FORTRAN reference manual corresponding to the version of FORTRAN specified on the DML control statement.

A number of variables are generated in the FORTRAN program by the DML preprocessor; these variables are reserved for use by DML, and should not be defined or referenced by the FORTRAN program. Refer to appendix F for a list of these reserved names.

Specification of Variables in DML Statements

For FORTRAN 5 programs, a data item specified as a variable in a DML statement must be declared type CHARACTER * 10, with the following exceptions:

In the PRIVACY statement, the variable used as the data item for the PRIVACY option must be declared CHARACTER * 30.

In the LOCK statement, the variable used as the data item for the TYPE option must be declared CHARACTER * 9.

If a FORTRAN 5 program uses a variable to specify version name for the INVOKE or NEWVERSION statement, the variable must be declared type CHARACTER * 7.

DML STATEMENTS

The realms referenced in DML statements should not be referenced elsewhere by conventional input/output statements, including the PROGRAM statement. These files should be referenced exclusively by DML statements. DML statements access files defined by the FORTRAN subschema. The subschema library file must be available to the DML preprocessor.

DML statements can appear both in the main program and in subprograms. Certain DML statements cannot appear within a data base transaction. (Any reference to a transaction in this section refers to a data base transation.) A transaction consists of a BEGINTRAN statement that marks the beginning of the transaction, the DML statements used for update operations within the transaction, and a COMMITTRAN or a DROPTRAN statement, either of which marks the end of the transaction.

Table 3-1 summarizes the DML statements and routines, describes where they are allowed to appear within the executable or nonexecutable portions of a program, and indicates whether they can appear within a transaction.

SYNTAX REQUIREMENTS

The syntax requirements for DML statements and routines are basically the same as for FORTRAN statements. The syntax requirements are outlined in the FORTRAN 5 reference manual. The exceptions to standard syntax are:

A DML statement or routine cannot be the object of a logical IF.

A DML statement or routine cannot appear on the same line as another statement or routine.

DML statements and routines are described in the following subsections in alphabetic order.

TABLE 3-1. FORTRAN DML STATEMENTS AND ROUTINES

Statement	Function	Position in Program
	Establishing and Terminating Data Base Access	
SUBSCHEMA	Identifies the subschema to be used by the program.	Must appear after specification statements and before DATA or NAMELIST statements, statement function definitions, or executable statements of every program unit containing DML statements.
INVOKE	Establishes the connection between the executing program and CDCS.	Must be executed before any other DML statement (except SUBSCHEMA which is nonexecutable); must appear in every program unit containing DML statements, but should not appear within a transaction.
NEWVERSION	Change the data base version used by an application program.	Anywhere after the CLOSE for a realm, except with a transaction.
TERMINATE	Disconnects the executing program from CDCS.	Must be the last DML statement to be executed (until a subsequent INVOKE statement), must precede an INVOKE statement used to change version names. If a TERMINATE statement occurs within a transaction, the transaction is automatically dropped.
	Opening and Closing a Realm or Relation	
OPEN realm	Initiates processing of a realm.	Anywhere between INVOKE and TERMINATE, except within a transaction; can only be executed when the specified realm is closed.
OPEN relation	Initiates processing of the realms joined in a relation.	Anywhere between INVOKE and TERMINATE, except within a transaction; can only be executed if the realms in the specified relation are closed.
CLOSE realm	Ends processing of a realm.	Anywhere after OPEN and before TERMINATE, except within a transaction; a realm can be opened and closed any number of times within a given program. The CLOSE statement can only be executed if the specified realm is open.
CLOSE relation	Ends processing of the realms joined in a relation.	Anywhere after OPEN and before TERMINATE, except within a transaction; realms in a relation can be opened and closed any number of times within a given program.
	Manipulating a Data Base	
START realm	Logically positions a realm for subsequent retrieval of records through a sequential DML READ statement.	Anywhere between OPEN and CLOSE for a realm.
START relation	Logically positions the root realm of the specified relation for subsequent retrieval of records through a DML READ relation statement.	Anywhere between OPEN and CLOSE for a relation.
READ realm	Transfers data from a record in the specified realm to the variables included in the subschema description of the record.	Anywhere between OPEN and CLOSE for a realm.

TABLE 3-1. FORTRAN DML STATEMENTS AND ROUTINES (Contd)

Statement	Function	Position in Program
	Manipulating a Data Base (Contd)	
READ relation	Transfer data from a record in each of the realms joined in the relation to the variables included in the sub-schema descriptions of the records.	Anywhere between OPEN and CLOSE for a relation.
DELETE	Logically removes a record from a realm.	Anywhere between OPEN and CLOSE for a realm; should be preceded by either READ or LOCK.
LOCK	Establishes a lock on a realm that prevents other jobs from reading and/ or updating the realm.	Anywhere between OPEN and CLOSE for a realm, except within a transaction.
PRIVACY	Establishes the right of a program to access a realm.	Must be executed before the first execution of OPEN for a realm with controlled access; must not appear within a transaction.
REWRITE	Logically replaces a record in a realm.	Anywhere between OPEN and CLOSE for a realm; should be preceded by either READ or LOCK.
UNLOCK	Releases a lock on a specified realm; releases any record locks the program holds on records in that realm.	Anywhere between OPEN and CLOSE for a realm, except within a transaction.
WRITE	Causes a record to be stored in a realm consisting of the current values of the variables included in the subschema description of the record.	Anywhere between OPEN and CLOSE for a realm.
	Processing a Data Base Transaction	
ASSIGNID	Obtains the restart identifier assigned by CDCS.	Normally specified before a transaction is initiated; must not be specified within a transaction.
BEGINTRAN	Identifies and begins a transaction.	Anywhere after OPEN; transaction processing must be allowed for the schema.
COMMITTRAN	Causes all updates of a successful transaction to be made permanent.	Anywhere after the associated BEGINTRAN; a transaction must have been initiated.
DROPTRAN	Cancels an active transaction; the records within the transaction are restored to the states that they were before the transaction began.	Anywhere after a BEGINTRAN, but before a COMMITTRAN.
FINDTRAN	Determines the appropriate place to restart transaction processing when a program is recovering from a system failure.	Anywhere within the program, usually in the program logic used to determine the point at which transaction processing is to resume after a system failure.
	Miscellaneous	
DMLDBST	Communicates the location and length of the data base status block to CDCS.	Anywhere after the INVOKE statement; need be specified only once.
DMLRPT	Defines a recovery point to CDCS.	Anywhere after INVOKE; except within a transaction.

ASSIGNID Statement

The ASSIGNID statement, shown in figure 3-2, obtains the restart identifier assigned by CDCS. This identifier can subsequently be used by the FINDTRAN statement to determine the status of a transaction when a system failure occurs. The restart identifier should not be saved on a data base file because it is used for data base file recovery. The application should contain the logic neccessary to save the identifier outside of the program.

FORTRAN 5 Format:

```
ASSIGNID (restart-id [,ERR=s])
```

Figure 3-2. ASSIGNID Statement Format

Normally, ASSIGNID should be specified before any updates are attempted within a transaction (although this is not required.) ASSIGNID must not be specified within a transaction. If the data base administrator has not assigned a restart identifier file to the data base, this statement cannot be used; if it is used, an error results.

Restart-id receives the 1- to 10-character restart identifier. Restart-id must be specified as a variable. (Refer to the Variable subsection for information about specification of variables in DML statements.)

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

BEGINTRAN Statement

The BEGINTRAN statement, shown in figure 3-3, indicates the beginning of a transaction to CDCS. Records which are subsequently updated remain exclusively locked until the transaction is either completed or dropped. Updates (write, rewrite, and delete operations) are considered temporary until the transaction is successfully completed. If an attempt is made to execute the BEGINTRAN statement and transaction processing is not allowed for the schema, a fatal error occurs. Refer to section 5 for a description of transaction processing.

FORTRAN 5 Format:

```
BEGINTRAN (tran-id [,ERR=s])
```

Figure 3-3. BEGINTRAN Statement Format

Tran-id identifies the 1- to 10-character transaction identifier supplied by the user. Tran-id can be specified either as a constant or as a variable. (Refer to either the Constant or the Variable subsection for more information.)

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

CLOSE Statement

The CLOSE statement, shown in figure 3-4, ends processing of the specified realm or of the realms joined in the specified relation. The only DML statements that can be executed on a realm when the realm is closed are either an OPEN or a PRIVACY statement. The CLOSE statement is not allowed within a transaction.

FORTRAN 5 Format:

```
CLOSE ({realm-name   }[,ERR=s])
       {relation-name}
```

Figure 3-4. CLOSE Statement Format

A CLOSE realm statement closes the specified realm. The realm specified in this statement must have been included in the subschema.

A CLOSE relation statement is executed as if a separate CLOSE were issued for each realm, in the order of rank of the realms in the relation. Realms closed by a CLOSE relation statement should not be explicitly closed by a CLOSE realm statement. If a realm closed by a CLOSE relation statement is already closed, no action is taken for that realm. The relation specified in the CLOSE statement must have been included in the subschema.

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

COMMITTRAN Statement

The COMMITTRAN statement, shown in figure 3-5, indicates the completion of a transaction to CDCS. Execution of this statement causes all updates of the present transaction to become permanent; all record locks are released so that the records become available for access by other application programs (unless a realm lock applies). A fatal error occurs if an attempt is made to execute the COMMITTRAN statement when a transaction has not been initiated (refer to the BEGINTRAN statement).

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

FORTRAN 5 Format:

```
COMMITTRAN [(ERR=s)]
```

Figure 3-5. COMMITTRAN Statement Format

FORTRAN 5 Format:

```
DELETE (realm-name [,ERR=s])
```

Figure 3-6. DELETE Statement Format

DELETE Statement

The DELETE statement, shown in figure 3-6, removes a record from a realm. The realm specified in this statement must have been included in the subschema.

Before an attempt is made to delete a record, the record must be locked either with a record lock or with a realm lock. A protected record lock is established when a DML READ statement is issued for a realm opened for input/output. A protected record lock can be established either outside of a transaction or within a transaction. An exclusive record lock is established when the delete request is issued within a transaction. A realm lock is established by issuing a DML LOCK statement. The realm lock overrides the record lock.

The recommended procedure for deleting a record is to issue a DML read request and delete the record. When the delete request executes, the record is deleted. The value of the primary key should not be changed between the read request and the delete, because a record is identified by its primary key value. Following this procedure ensures that the correct record is deleted.

This procedure must be followed when deleting a record with a protected record lock outside of a transaction. Before the DELETE statement executes, CDCS checks the value of the primary key to ensure that it has not changed since the read. If the value of the primary key has changed, CDCS issues a diagnostic message and ignores the delete request.

This procedure should be followed when deleting a record with an exclusive record lock within a transaction or when deleting a record with a realm lock. If the value of the primary key is changed, the wrong record could inadvertently be deleted.

Refer to section 5 for more information about both the exclusive record lock and the realm lock.

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

Figure 3-7 shows an example of the DELETE statement. Sample keys on the realm REBLOCHON, both before and after execution of the DELETE statement, are also shown. In the example, the item PKEY has been defined in the schema as the primary key. The READ statement reads the record with the primary key that is greater than 300 (in this case the record with the primary key of 550). Execution of the DELETE statement removes this record from the realm.

```
Keys on REBLOCHON before the delete operation:

      .
      .
      .
      70
      120
      190
      550
      663
      664
      .
      .
      .

Statements required to delete the record with
the primary key 550:

          PKEY = 300
          READ (REBLOCHON, KEY .GT. PKEY)
      C   THIS WILL DELETE RECORD WITH
             PRIMARY KEY OF 550
          DELETE (REBLOCHON)

Keys on REBLOCHON after the delete operation:

      .
      .
      .
      70
      120
      190
      663
      664
      .
      .
```

Figure 3-7. Example of Use of the DELETE Statement

DMLDBST Routine

The DMLDBST routine, shown in figure 3-8, communicates the location and length of the data base status block to CDCS. The routine DMLDBST can be called at any point after the INVOKE statement. It need be called only once. The data base status block specified in the call is updated for any data base operation performed after the call. Each time DMLDBST is called, the data base status block is initialized to zero or blanks, so it should not be called after execution of a DML statement if the status of that statement is desired. Only one data

```
CALL DMLDBST (status-block, length)
```

Figure 3-8. DMLDBST Routine Format

base status block can exist at a time for a FORTRAN program. If DMLDBST is called more than once in a program, the data base status block defined in the last call is the one that is updated by CDCS.

If DMLDBST is not called, the FORTRAN program still can reference the variable DBSTAT and the status words for the realm in the common blocks set up by DML (described under Additional Status Checking Elements later in this section).

Status-block identifies the data base status block. Status-block must be specified as an integer array.

Length defines the length in words of the data base status block. Length can be specified as either a variable or a constant.

Refer to the Data Base Status Block subsection later in this section for more information.

DMLRPT Routine

The DMLRPT routine, shown in figure 3-9, defines a recovery point to CDCS. This is a point to which the data base would be restored for easy restarting of the program should recovery of the data base be necessary. Recovery point definition is not allowed within a transaction. For recovery purposes, the use of transaction processing is recommended rather than the use of this routine.

```
CALL DMLRPT (rpt-num, comment)
```

Figure 3-9. DMLRPT Routine Format

Execution of the subroutine DMLRPT causes the following events to occur in the order given. The user program is suspended until these events have taken place

All I/O buffers for data base realms are flushed.

A recovery point log record is force written to the log file for the data base.

The quick recovery file for the data base is emptied.

On return from this subroutine, the user is assured that the data base can be recovered to its current state (barring such disasters as simultaneous destruction of both the data base and the journal log file).

The creation of a recovery point does incur overhead, since CDCS activity halts for all users until the preceding three events have been completed. To reduce this overhead, an application might choose to create a recovery point every fourth update. Judicious use of recovery points can aid in recovery, but misuse can severely impact processing time. The data administrator should be consulted to determine if the use of a recovery point is applicable to a particular file.

Rpt-num receives the unique recovery point number from CDCS; the recovery point number can be retained by the user for reference purposes. Rpt-num must be specified as an integer variable.

Comment specifies a 1- to 30-character user-supplied explanatory message for the recovery point; the explanatory message is written to the CDCS journal log file along with the recovery point number. For FORTRAN 5, comment can be specified as a variable that has been defined as type CHARACTER * 30.

DMLSIR Routine

The DMLSIR routine, shown in figure 3-10, enables and disables the immediate return feature. The DMLSIR routine cannot be specified before the INVOKE statment executes.

```
CALL DMLSIR (item-name)
```

Figure 3-10. DMLSIR Routine

If the immediate return feature is enabled, a FORTRAN application program can receive an immediate response from CDCS when resource conflicts or fatal errors occur. If the immediate return feature is disabled, a FORTRAN application program cannot provide program logic to determine the action taken when resource conflicts or fatal errors occur.

The value specified for item-name determines whether the immediate return feature is enabled or disabled. If item-name is not equal to zero, immediate return is enabled. If item-name is equal to zero, immediate return is disabled. Item-name must be specified as an integer variable or constant. Refer to section 5 for more information about the immediate return feature.

A data base status block must be defined if a FORTRAN program is to be able to detect a fatal error or resource conflict. Refer to the Data Base Status Block subsection later in this section for more information.

DROPTRAN Statement

The DROPTRAN statement, shown in figure 3-11, cancels the current transaction. Execution of the DROPTRAN statement causes CDCS to restore the records updated within the transaction to their original states which existed just before the transaction was initiated, and also causes CDCS to release all record locks. A fatal error occurs if an attempt is made to execute a DROPTRAN statement if no transaction has been initiated (refer to the BEGINTRAN statement).

FORTRAN 5 Format:

```
DROPTRAN [(ERR=s)]
```

Figure 3-11. DROPTRAN Statement Format

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

FINDTRAN Statement

The FINDTRAN statement, shown in figure 3-12, obtains information for a program restart operation after a system failure. This statement is normally issued in the restart unit of the program.

FORTRAN 5 Format:

```
FINDTRAN (restart-id, tran-id [,ERR=s])
```

Figure 3-12. FINDTRAN Statement Format

Restart-id identifies the 1- to 10-character restart identifer that was assigned to the program before the system failure. Restart-id can be specified either as a constant or as a variable. (Refer to the Constant or Variable subsection for more information.)

Tran-id receives the transaction identifier of the last completed transaction; this identifier is returned only if the application program had begun, but not committed or dropped, a CDCS transaction prior to a system failure. Tran-id must be specified as a variable. (Refer to the variable subsection earlier in this section for more information).

Tran-id receives the characters ********** (10 asterisks) if the restart identifier is unknown to CDCS. The restart identifier is unknown to CDCS if the wrong value is specified for restart-id or if the run-unit terminated normally. If the run-unit terminated normally, a new restart identifer must be obtained.

Tran-id receives a value of ten blanks if the restart identifier is known to CDCS, but no transaction had been completed prior to the system failure. The FINDTRAN statement executes normally and no new restart identifier need be obtained; the restart identifier specified as restart-id is reassigned to the program.

If a transaction identifier is returned, no new restart identifer need be obtained; the restart identifer specified as restart-id is reassigned to the program.

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

INVOKE Statement

The INVOKE statement, shown in figure 3-13, must be specified before any executable DML statement (except SUBSCHEMA, which is nonexecutable). It establishes communication between the application program and CDCS. INVOKE must be executed in every run-unit (the main program and any subprograms) in which DML statements are executed.

```
INVOKE [(VERSION=version-name)]
```

Figure 3-13. INVOKE Statement Format

The INVOKE statement is not normally allowed within a transaction; however, if an INVOKE statement is specified within a TAF task (or task chain) the statement is ignored unless it is associated with a different subschema or version name. If INVOKE is associated with a different subschema or version name within a transaction, an error results.

The application program can change the version it is currently using by first executing a TERMINATE statement and then re-invoking CDCS with an INVOKE statement that specifies an alternate version name. Even though the privacy keys are the same for each version of the subschema, the application must repeat PRIVACY statements after each INVOKE statement if access control locks are in use.

In an application program consisting of more than one program unit, multiple INVOKE statements can occur without an intervening TERMINATE statement. In this case the version name appearing in any subsequent INVOKE statement must be the same as the version name specified for the first INVOKE statement. If the version name is omitted in subsequent INVOKE statements, the version name specified in the first INVOKE statement is assumed.

Version-name specifies the 1- to 7-character version name described in the master directory for the schema being used. Version-name can be specified as either a constant or a variable. (Refer to either the Constant or Variable subsection for more information.) If the VERSION option is omitted, version MASTER is assumed.

LOCK Statement

The LOCK statement, shown in figure 3-14, identifies a realm and a lock type that either restricts or prevents access to the realm by other jobs. Although CDCS always assigns a protected lock to a record read with intent to update (the record can

FORTRAN 5 Format:

```
LOCK (realm-name [,TYPE= lock-type [,ERR=s]])
```

Figure 3-14. LOCK Statement Format

be read but not updated by another concurrently executing program), the LOCK statement locks the whole realm. The realm lock is in effect until cancelled by an UNLOCK or CLOSE statement or unless a deadlock situation occurs. When using transaction processing, use of the LOCK statement is not recommended. The LOCK statement is not allowed within a transaction.

When a realm has been locked by the LOCK statement, the recommended procedure for deleting or rewriting a record in that realm is to read the record, and then perform the delete or rewrite operation. This procedure offers protection for the data base, because a program can check for an error on the read before proceeding with the delete or rewrite operation. Another procedure is also available; that is, the value of the primary key can be set to the value of the key of the record to be deleted or rewritten, and then the delete or rewrite operation can be performed. This procedure does not offer as much protection to the data base. A realm lock should be used judiciously when deleting or rewriting a record, because the realm lock overrides the record lock and the checking capabilities through the record lock.

Two types of locking are permitted: exclusive or protected. Exclusive locking prohibits concurrent access to the realm for read or update operations; protected locking allows concurrent access to the realm for read operations only.

Realm-name specifies the realm to be locked. Realm-name must be described in the subschema or a nonfatal error occurs.

Lock-type identifies the type of locking desired. Lock-type can be specified as either a constant or a variable. The lock-type option must specify one of the values mentioned previously, either the value EXCLUSIVE or the value PROTECTED. (Refer to the Constant or Variable subsection earlier in this section for more information.)

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

NEWVERSION Statement

The NEWVERSION statement, shown in figure 3-15, changes the data base version that is being used by a program. Use of this statement provides the application program with the capability to change version names without having to terminate and re-invoke CDCS. Therefore, when privacy locks are in effect, versions can be changed without repeating PRIVACY statements. The NEWVERSION statement is not allowed within a transaction.

FORTRAN 5 Format:

```
NEWVERSION (version-name [,ERR=s])
```

Figure 3-15. NEWVERSION Statement Format

If a NEWVERSION statement is used to change versions, a subsequent INVOKE statement (one that is specified without an intervening TERMINATE or NEWVERSION statement) must conform to the following rules:

If a version name appears in the INVOKE statement, it must be the same as the version specified in the NEWVERSION statement.

If the version name is omitted in the INVOKE statement, the version specified in the NEWVERSION statement is assumed.

All the subschema realms for the program must be closed before the NEWVERSION statement executes or a fatal error results. A fatal error also occurs if the version name specified does not exist before the NEWVERSION statement is executed.

Version-name identifies the 1- to 7-character version name described in the master directory for the schema being used. Version-name can be specified as either a constant or a variable. (Refer to either the Constant or Variable subsection for more information.)

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

Figure 3-16 shows an example of the use of the NEWVERSION statement. The INVOKE statement specifies that version TEST17 is to be used. The NEWVERSION statement is used to change the version name. The first iteration of the DO loop causes version BRANCHA to be used, the second iteration causes version BRANCHB to be used, and so on.

```
      INTEGER STATBLK(11)
      CHARACTER * 7 VERS(5)
      SUBSCHEMA (PARTORD)
      DATA VERS/'BRANCHA','BRANCHB','BRANCHC','BRANCHD','BRANCHE'/
      INVOKE (VERSION='TEST17')
C     VERSION IS TEST17
      OPEN (PRODUCTS, MODE=IO)
      .
      .
      .
      CLOSE (PRODUCTS)
      DO 25 I=1,N
      NEWVERSION (VERS(I))
C     VERSION FOR FIRST ITERATION OF THE DO LOOP IS BRANCHA, THEN
C     BRANCHB, AND SO FORTH.
      OPEN (PRODUCTS, MODE=IO)
      .
      .
      .
      CLOSE (PRODUCTS)
25    CONTINUE
      TERMINATE
```

Figure 3-16. Example of Use of the NEWVERSION Statement

OPEN Statement

The OPEN statement, shown in figure 3-17, prepares a realm or a relation for processing. No other statement (except PRIVACY) related to the realm or relation can be executed when the realm or the relation is not open. The OPEN statement is not allowed in a transaction.

FORTRAN 5 Format:

```
OPEN{ realm-name    } [,MODE={I }] [,ERR=s])
    { relation-name }    {IO}
                         {O }
```

Figure 3-17. OPEN Statement Format

Successful execution of an OPEN statement sets the key of reference to the primary key of the first record in the realm (or the root realm of a relation). Execution of this statement makes the record associated with the primary key value available to the program; it does not obtain or release the record.

The OPEN realm statement makes the records in a realm available to the FORTRAN program and positions the realm at beginning of information. The realm specified in the OPEN statement must be among those included in the subschema.

The OPEN relation statement makes the records in the realms joined in the specified relation available to the FORTRAN program and positions each realm in the relation at beginning of information. The relation specified in the OPEN statment must be included in the subschema.

Relations are normally opened for input (MODE=I). The relation can be opened for input and output (MODE=IO) if the user wishes to have locking of records read for the relation occurrence, or if individual realms in the relation are to be updated. If any of the realms included in the relation are already open, no action occurs for that realm. Any previous mode setting remains in effect.

The option chosen for MODE determines the type of processing allowed on a realm or a relation, as follows:

When a realm is opened for input (MODE=I), only the READ, LOCK, UNLOCK, and CLOSE statements can be executed.

When a realm is opened for input/output (MODE=IO), the READ, WRITE, DELETE, REWRITE, LOCK, UNLOCK, and CLOSE statements can be executed.

When a realm is opened for output (MODE=O), only the WRITE, LOCK, UNLOCK, and CLOSE statements can be executed.

MODE=O must be specified for creation of a new file; it is not valid for an existing file. If MODE is omitted, the default is MODE=IO.

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

An example of the OPEN statement is shown in figure 3-18. The realm named CUL-DE-SAC is opened in read only mode.

```
OPEN (CUL-DE-SAC, MODE = I)
```

Figure 3-18. Example of Use of OPEN Statement

PRIVACY Statement

The PRIVACY statement, shown in figure 3-19, establishes the right of a program to access a realm. It has no effect unless the realm was defined with controlled access in the schema. If the realm was defined with controlled access in the schema, the PRIVACY statement must supply the privacy key before the realm can be opened. To access several realms, each requiring a privacy key, the FORTRAN program must contain a PRIVACY statement for each realm. Similarly, to open a relation that joins realms with each realm requiring a privacy key for access, a PRIVACY statement for each realm must be specified before the OPEN relation statement. If an INVOKE statement is used to change version names, PRIVACY statements must be repeated for each realm. The PRIVACY statement is not allowed within a transaction.

At execution time, the key specified in the PRIVACY statement is compared with the lock specified in the schema. If a program attempts to open a realm without supplying the correct privacy key, the program is terminated.

Realm-name specifies the realm which requires the privacy key. The realm specified in the PRIVACY statement must be included in the subschema.

Privacy-key specifies the 1- to 30-character privacy key for the PRIVACY option. The language elements used to specify privacy-key depend on the version of FORTRAN used. Privacy-key can be specified in FORTRAN 5 as a character constant, a variable, or the unsubscripted name of a three-word array. If either a variable or an array is used, the data assigned must be character data.

The value of the MODE option determines the type of access allowed when the character string specified by privacy-key matches the privacy key defined in the schema. Specification of the value I restricts access to read operations only; the value 0 restricts access to update operations only. If the value IO is specified, access is allowed for both read and update operations (the default if the MODE option is omitted). The value specified for the MODE option of the PRIVACY statement must be compatible with the value specified for the MODE option of the OPEN statement.

If the data administrator has defined separate privacy keys for update and retrieval (read access) for a realm, the FORTRAN program must specify two PRIVACY statements to open the realm for input/output. One statement must specify MODE=I; the other must specify MODE=0. Table 3-2 shows the mode option that is used to satisfy the types of access control locks specified by the data administrator in the schema.

TABLE 3-2. SCHEMA ACCESS CONTROL LOCK AND CORRESPONDING MODE OPTION USAGE

Schema Access Control Lock	Corresponding FORTRAN MODE Option
Update	0
Retrieval†	I
Update/retrieval†	I and 0 IO

†Retrieval refers to read access.

Examples of the PRIVACY statement are shown in figure 3-20. In the FORTRAN 5 example, two PRIVACY statements are specified since separate access control locks were declared by the data administrator: one for update access and the other for retrieval (read) access. If STUDENT-ID is specified as the privacy key for retrieval (read) operations, and CLASS-ID is specified as the privacy key for update operations, then the realm can be opened for input/output.

FORTRAN 5 Example:

```
CHARACTER*30 READKY,RITEKY * 30
DATA READKY/'STUDENT-ID'/
DATA RITEKY/'CLASS-ID'/
  .
PRIVACY(AVERAGE,MODE=I,PRIVACY=READKY)
PRIVACY(AVERAGE,MODE=0,PRIVACY=RITEKY)
```

Figure 3-20. PRIVACY Statement Examples

READ Statement

The READ statement, shown in figure 3-21, causes CDCS to read a record or relation occurrence from the specified realm or from the realms in the specified relation. The record (or records in the relation occurrence) must be described in the subschema.

```
PRIVACY (realm-name [,MODE={I|IO|0}], PRIVACY = privacy-key)
```

Figure 3-19. PRIVACY Statement Format

FORTRAN 5 Format:

```
READ ({realm-name     }  [ ,KEY { =   }  { item-name            }  ]  [,ERR=s] [,END=s])
      {relation-name  }        { .EQ. }  { concatenated-key-name }
                               { .GT. }
                               { .GE. }
```

Figure 3-21. READ Statement Format

Syntax Requirements

Table 3-2.1 shows how the READ syntax corresponds to read characteristics.

The KEY option of the READ statement establishes the key of reference. The key of reference is the primary key or alternate key of the record or relation read. Once the key of reference has been established, it can only be changed by another READ statement (with the KEY option specified), a START statement, or an OPEN statement.

Refer to the ERR and END Specifiers subsection later in this section for information about these parameter.

Realm Read

A realm read causes a record to be read from the specified realm and disassembled into the variables and arrays included in the subschema description of the record. These variables and arrays in the FORTRAN program are set to their present values from the record. If any type conversion is implied by the correspondence between the schema and subschema descriptions of a data item, it is performed at this time; the program receives data according to its description in the subschema.

Relation Read

A relation read causes CDCS to read a relation occurrence; the relation specified in the READ statement must be included in the subschema. A relation occurrence consists of one record from each of the realms comprising the relation. The FORTRAN variables and arrays included in the subschema description of each record are set to their current values from the relation occurrence. Refer to section 5 for detailed information about reading a relation.

Random Read

If the KEY option of a READ statement is specified, the read is random; the record or relation occurrence read is the record or relation occurrence with the key that satisfies the specified comparison. In the KEY option, item-name must be set before the read to the primary or alternate key value of the record occurrence or relation occurrence desired. For a realm read, item-name must have been defined in the subschema, and must refer to the primary or alternate key for the realm. For a relation read, item-name must have been defined in the subschema and must refer to the primary or alternate key of the root realm of the relation. (The root realm is the first realm listed for the

TABLE 3-2.1 SYNTAX CORRESPONDENCE TO READ CHARACTERISTICS

Syntax	Read Characteristics					
	Random	Sequential	Realm	Relation	Concatenated Key	Single-item Key
realm-name			X			
relation-name				X		
KEY option used	X					
KEY option omitted		X				
item-name						X
concatenated-key-name					X	

relation in the subschema listing as shown in the Subschema Listing subsection earlier in this section.)

Item-name can be of any data type except logical. If it is complex or double precision, it is treated as real. If it is complex, the imaginary part is discarded and it is treated as a real value. If it is double precision, the least significant part is discarded and it is treated as real.

When the comparison specified by the relational operator of the KEY option is performed, the record returned is the first record in the realm (or in the root realm of the relation) that satisfies the specified comparison. If the comparison is .EQ. or =, the key of the record must exactly match the value specified in item-name. (No conversion is performed, except as described for complex and double precision items.) If the comparison is .GE., the key of the record must be greater than or equal to the value specified in item-name.

The value of the primary or alternate key and the value specified in item-name are compared by numerical magnitude. The values are determined by the collating sequence defined for the realm in the schema. (The data administrator can provide this information.) Collating sequences are shown in appendix I.

Read With a Concatenated Key

If a concatenated key is used for a read, concatenated-key-name is specified in the KEY option. The constituent items of the concatenated key must be individually set before the READ statement. These items must also have been defined in the subschema.

Sequential Read

If the KEY option is omitted, the read is sequential; that is, the record or relation occurrence located is the next record or relation occurrence from the present location of the realm or relation. The next record or relation occurrence is determined by the collating sequence defined for the realm (the root realm of a relation) in the subschema. (The data administrator can provide this information.) The collating sequences are shown in appendix I.

Examples

Figure 3-22 shows the keys on the realm ADMIN. In the first set of statements, item-name STDENT is set to the primary key value 31. Since the comparison specified is =, the record read is the record whose primary key is 31. In the second set of statements, item-name STDENT is set to the primary key value 35. Since the comparison specified is .GE., the record read is the record whose primary key is greater than or equal to 35; in this case, the record whose key is 36.

```
Keys on ADMIN:

      .
      .
      .
      21
      31
      34
      36
      37
      .
      .
      .

 Statements that read the record whose key is 31:

      INTEGER STDENT
           .
           .
      STDENT = 31
      READ (ADMIN, KEY = STDENT)


Statements that read the record whose key is 36:

      STDENT = 35
      READ (ADMIN, KEY .GE. STDENT)
```

Figure 3-22. Example of Use of READ Statement

REWRITE Statement

The REWRITE statement, shown in figure 3-23, replaces an existing record in a realm with a new record based on the current values of all the variables defined for the record in the subschema.

```
FORTRAN 5 Format:

     REWRITE (realm-name [,ERR=s])
```

Figure 3-23. REWRITE Statement Format

Before rewriting a record, the record must be locked, with either a record lock or with a realm lock. A protected record lock is established when a DML READ statement is issued for a realm opened for input/output. A protected record lock can be established either within a transaction or outside of a transaction. An exclusive record lock is established when the rewrite request is issued within a transaction. A realm lock is established by issuing a DML LOCK statement.

The recommended procedure for rewriting a record is to precede the REWRITE statement with a DML READ statement. The value of each data item being rewritten can then be changed. When the REWRITE statement executes, the record is rewritten. The value of the primary key should not be changed between the read and the rewrite of the record, because a record is identified by its primary key

value. Following this procedure insures that the correct record is rewritten.

This procedure must be followed when rewriting a record with a protected record lock outside of a transaction. If the value of the primary key is changed before the rewrite is attempted, CDCS issues a diagnostic message. If the value of the primary key of a record with a protected record lock must be changed outside of a transaction, the program must delete the record with the old primary key value and write the record with the new primary key value.

This procedure should be followed when rewriting a record with an exclusive record lock within a transaction or when rewriting a record with a realm lock. If this procedure is not followed, the wrong record might be inadvertently rewritten.

Refer to section 5 for information about the exclusive record lock and the realm lock.

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

START Statement

The START statement, shown in figure 3-24, positions a realm or relation for subsequent retrieval of records; however, this statement does not cause a record to be transferred to the program. The START statement can be specified any number of times. Before the START statement is executed, the realm or relation must have been opened with MODE=I or MODE=IO specified.

Syntax Requirements

realm-name

Specified for a realm start.

relation-name

Specified for a relation start.

KEY

Specifies the positioning of the realm or relation. If KEY option is omitted, the realm (the root realm of a relation) is positioned at the record whose primary key value equals the current value of the primary key (item-name or concatenated-key-name, whichever applies). If the KEY option is specified, comparison occurs as described later.

item-name

Used to specify a single-item key (as opposed to a concatenated key).

concatenated-key-name

Concatenated key name as listed on the FORTRAN subschema output. It can be specified when retrieval is by concatenated key.

item-name-list

List of item-names separated by commas: item-name-1, item-name-2, etc. It can be used when a concatenated key is used and also when major key processing is done. Item-names can be all the constituent items or the leading items of the concatenated key. Refer to examples later.

ERR=s

Refer to the ERR and END Specifiers subsection later in this section for information about this parameter.

Realm START

The realm START statement establishes the key of reference for a realm by positioning the realm to the first record occurrence whose key value meets the specified condition. The key that satisfies the condition becomes the new key of reference. The statement is normally followed by a sequential realm READ statement, which performs the read from the position established by the execution of the START statement.

Relation START

The relation START statement positions a relation for subsequent sequential retrieval of records through relational reads. The root realm of the relation is positioned as described in the preceding paragraph. This statement establishes the key of reference for the root realm of the relation. A relation READ statement following a relation START

FORTRAN 5 Format:

```
START ({realm-name   } [,KEY{=    } {item-name          } ] [,ERR=s])
       {relation-name}     {.EQ.} {concatenated-key-name}
                           {.GT.} {item-name-list       }
                           {.GE.}
```

Figure 3-24. START Statment Format

statement retrieves from the root realm both the record with the key that satisfies the start condition and a record from each realm in the associated relational hierarchy.

KEY Comparisons

If the KEY option is specified, the comparison specified by the relational operator is performed. As a result of this comparison, the realm or relation is positioned at the first record occurrence or relation occurrence that satisfies the comparison. Comparison is performed by numerical value; for example, if the comparison is .GT., the value of the key in the record must be strictly greater than the value of item-name.

The values used for comparison are determined by the collating sequence specified for the realm (the root realm of a relation) in the subschema. The data administrator can provide this information. Collating sequences are shown in appendix I.

Using a Single-Item Key

In the KEY option, item-name specifies a single-item primary or alternate key. Item-name must match a data item described in the subschema as a primary or alternate key for the realm (root realm in the case of a relation). Also, item-name must be set to the primary or alternate key value of the record or relation occurrence desired before the START is executed.

For a FORTRAN 4 program, item-name can be a special long variable. (Refer to the Special Long Variable subsection earlier in this section for more information.)

Using a Concatenated Key

In the KEY option, concatenated-key-name specifies a concatenated primary or alternate key. It must match the concatenated key name as shown in the subschema output listing. Also, each constituent item of the concatenated key must be individually set to the primary or alternate key value of the record or relation occurrence desired. Once the constituent items are set, the concatenated key name can be used in the START statement.

For example, if ITEMA, ITEMB, and ITEMC are the contiguous data items making up the concatenated key, the START statement might appear as follows:

```
ITEMA=M130
ITEMB=123
ITEMC=77
START(FILE1, KEY .EQ. CONCKEY)
```

where CONCKEY is the concatenated key name as listed in the subschema. The list ITEMA,ITEMB,ITEMC could have been specified instead of CONCKEY in the START statement.

Using Major Key Processing

Major key processing can only be specified in the START statement.

If concatenated keys have been defined in the subschema, major key processing can be done. For major key processing, the item-name-list must match the leading contiguous elementary items that make up the concatenated key as listed in the subschema. Each data item in the list must be individually set to the primary or alternate key value of the record or relation occurrence desired. The item-name-list can include as many of the leading contiguous items that make up the concatenated key as defined in the subschema.

Item-name-list consists of a list of item-names separated by commas. For example, if ITEMA, ITEMB, and ITEMC are the contiguous data items making up the concatenated key, the START statement specifying major key processing might appear as follows:

```
ITEMA=X85
ITEMB=600
START(FILE2, KEY .EQ. ITEMA, ITEMB)
```

Leading items ITEMA and ITEMB are set to the respective values before the START.

START or READ

Under most circumstances, either a START statement or a READ statement that specifies the KEY option can be used to position a realm or relation. However, when a user wants to position a relation for subsequent sequential read operations and that particular relation is qualified by a RESTRICT statement, the START relation statement must be used. Under this one circumstance, the result of positioning the relation with the READ statement (READ relation-name KEY) would be unpredictable.

SUBSCHEMA Statement

The SUBSCHEMA statement, shown in figure 3-25, is required in every program or subprogram unit accessing the data base defined by the subschema. It need not appear in a program unit that contains no DML statements. It is nonexecutable, and must appear after the specification statements and before the first DATA or NAMELIST statement, statement function, or executable statement.

```
SUBSCHEMA (subschema-name)
```

Figure 3-25. SUBSCHEMA Statement Format

Only one subschema can be used by a FORTRAN program. The subschema to be referenced must have been previously compiled with the same language version (FORTRAN 5) as specified in the DML control statement. The subschema must be available to the DML preprocessor.

At the point in a program unit where the SUBSCHEMA statement appears, the DML preprocessor copies the text of the type declarations and DATA statements resulting from the subschema compilation into the program. The variables and arrays appear in common blocks. In addition, several other variables and arrays are declared by DML. These variable and array names are reserved for use by DML and should not be defined by the user. Refer to the subsection Additional Status Checking Elements later in this section for a list of these arrays and variables.

TERMINATE Statement

The TERMINATE statement, shown in figure 3-26, disconnects the application program from CDCS. After TERMINATE has been executed, no other DML statements can be executed until another INVOKE has been executed. A TERMINATE statement must precede an INVOKE statement that is used to change a version name. The TERMINATE statement must be executed before the FORTRAN STOP or END statement.

```
TERMINATE
```

Figure 3-26. TERMINATE Statement Format

TERMINATE permits CDCS to close files and return resources used by the job. If the job terminates abnormally before TERMINATE is executed, CDCS automatically performs the same functions.

UNLOCK Statement

The UNLOCK statement, shown in figure 3-27, releases the lock on the specified realm; any record locks for the realm that are held by a run-unit are also released. The realm specified in this statement must have been included in the subschema. The UNLOCK statement is not allowed within a transaction.

```
FORTRAN 5 Format:

UNLOCK (realm-name [,ERR=s])
```

Figure 3-27. UNLOCK Statement Format

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

WRITE Statement

The WRITE statement, shown in figure 3-28, uses the current value of all the variables defined for the record in the subschema to construct a record, and then writes the record to the data base file associated with the named realm. All primary and alternate keys must be set appropriately before the record is written. Any data items in the schema record that are not defined in the subschema are given null values before the record is written to the database.

Refer to the ERR and END Specifiers subsection later in this section for information about the ERR=s parameter.

```
FORTRAN 5 Format:

WRITE (realm-name [,ERR=s])
```

Figure 3-28. WRITE Statement Format

An example of the use of the WRITE statement is shown in figure 3-29. The keys on the realm WRBLOCHON are also shown, both before and after execution of the WRITE statement. In the example, if the item PKEY was defined in the schema as the primary key, execution of the WRITE statement creates a new record with the primary key 662.

```
Keys on WRBLOCHON before the write operation:

      .
      .
      .
      60
      120
      190
      550
      663
      664
      .
      .
      .

Statements that write a new record for WRBLOCHON:

          PKEY = 662
      C   CREATE A NEW RECORD
          WRITE (WRBLOCHON)

Keys on WRBLOCHON after the write operation:

      .
      .
      .
      70
      120
      190
      550
      662 (new record)
      663
      664
      .
      .
      .
```

Figure 3-29. Example of Use of the WRITE Statement

LISTING CONTROL DIRECTIVES

The DML preprocessor automatically generates listing control directives as part of the translated FORTRAN program. These directives have the form:

```
FORTRAN 5

    C$  LIST(ALL=0)
    C$  LIST,ALL
```

The first directive inhibits the listing of all succeeding FORTRAN statements. The second directive resumes listing of FORTRAN statements. The DML preprocessor inserts these directives immediately after the SUBSCHEMA and INVOKE statements, as shown in figure 3-30.

```
      .
      .
      .
**    SUBSCHEMA(BIRD)
C$    LIST,(ALL=0)
C$    LIST,ALL
C
**    INVOKE
C$    LIST,(ALL=0)
C$    LIST,ALL
      .
      .
      .
```

Figure 3-30. Example of Listing Control Statements in a FORTRAN Program

The effect of the listing control directives is to inhibit the listing of the FORTRAN statements generated by DML. These directives can be suppressed by the DS parameter on the DML control statement, in which case all FORTRAN statements generated by DML appear on the FORTRAN source listing. Note that the CALL statements generated as a result of executable DML statements are not suppressed by the listing control directives.

DML CONTROL STATEMENT

The DML control statement causes the preprocessing of DML statements. The format of the DML control statement is shown in figure 3-31.

DML Control Statement Parameters

The DML control statement parameters are interpreted in the following paragraphs. All the parameters are optional and can appear in any order.

The SB parameter specifies the name of the file containing the subschema library. The SB parameter is interpreted as follows:

omitted

Same as SB; the local file SBLFN is assumed to contain the subschema library.

SB

The local file SBLFN is assumed to contain the subschema library.

SB=0

Not allowed.

SB=lfn

The specified local file name identifies the file containing the subschema library.

The LV parameter specifies the version of FORTRAN for which the DML preprocessor generates statements. The LV parameter is interpreted as follows:

omitted

Same as LV=F5; this option can be changed at installation time.

LV

Same as LV=F5.

LV=0

Not allowed.

LV=F5

The DML preprocessor is to generate statements for processing by the FORTRAN 5 compiler.

The I parameter specifies the file which contains the FORTRAN source program and the added DML statements that are to be preprocessed by DML. The I parameter is interpreted as follows:

omitted

The local file INPUT is assumed to contain the source input for the DML preprocessor.

I

The local file COMPILE is assumed to contain the source input for the DML preprocessor.

I=0

Not allowed.

I=lfn

The specified local file name identifies the sequential file containing the source input for the DML preprocessor.

The O parameter specifies the file to which the translated version of the FORTRAN source program is to be written. DML statements appearing in the FORTRAN program are translated into FORTRAN statements before being written to this file. The O parameter is interpreted as follows:

omitted

Same as O; the local file DMLOUT receives the translated output from the DML preprocessor.

O

The local file DMLOUT receives the translated output from the DML preprocessor.

O=0

No translated output file is produced.

O=lfn

The specified local file name identifies the file which is to receive the translated output from the DML preprocessor.

```
DML[,SB=lfn][,LV=op][,I=lfn][,O=lfn][,E=lfn][ET=op][,DS].
```

Figure 3-31. DML Control Statement Format

The E parameter specifies the name of the file to which error diagnostics are written. The E parameter is interpreted as follows:

omitted

The local file OUTPUT receives the diagnostics generated by DML.

E

The local file ERR receives the diagnostics generated by DML.

E=lfn

The specified local file name identifies the file which is to receive the diagnostics generated by DML.

The ET parameter specifies the error termination code. Four levels of errors are defined: trivial (T), warning (W), fatal (F), and catastrophic (C). The error levels are explained with the corresponding option. If an error of the specified level or higher occurs, the job is aborted to an EXIT control statement (NOS) or EXIT(S) control statement (NOS/BE). The abort does not take place until DML is finished. The ET parameter is interpreted as follows:

omitted

Same as ET=0; the job step is not to be aborted even if errors occur (except for control statement errors).

ET=0

The job step is not to be aborted even if errors occur (except for control statement errors).

ET

Same as ET=F; the job is to abort if an error of level F or C occurs. The translated output file cannot be successfully compiled by FORTRAN.

ET=T

The job is to abort if an error of level T, W, F or C occurs. When a level T error occurs, the syntax of statement usage is correct but questionable. The translated output file can be compiled by FORTRAN, but the results might not be those expected.

ET=W

The job is to abort if an error of level W, F, or C occurs. When a level W error occurs, the syntax of statement usage is incorrect, but the preprocessor has been able to recover by making an assumption about what was intended. The translated output file can be compiled by FORTRAN, but the results might not be those expected.

ET=F

The job is to abort if an error of level F or C occurs. When a level F error occurs, the DML preprocessor is prevented from processing the statement in which the error occurs. Unresolvable semantic errors also fall into this category. The translated output file cannot be successfully compiled by FORTRAN.

ET=C

The job is to abort if an error of level C occurs. When a level C error occurs, compilation cannot continue. DML advances to the end of the current program unit and attempts to process the next program unit. The translated output file cannot be successfully compiled by FORTRAN.

The DS parameter specifies listing control directive suppression. Refer to the Listing Control Directive subsection earlier in this section for more information. If this parameter is specified, all FORTRAN statements generated by DML preprocessing appear on the FORTRAN source listing. The DS parameter is interpreted as follows:

omitted

Listing control directives are generated.

DS

Listing control directives are not generated.

DML Control Statement Example

An example of the DML control statement is as follows:

```
DML (LV=F5,SB=BRIE,I,E=ERRFILE,ET=W)
```

This control statement specifies that the DML preprocessor is to generate statements for the FORTRAN 5 compiler, that the subschema library is on the file BRIE, that input is on the file COMPILE, that output is to be written to DMLOUT, that error messages are to be written to ERRFILE, and that the job step aborts if any errors of warning level or higher occur.

COMPILATION/EXECUTION

Compilation of a FORTRAN application program for processing with CDCS requires that the subschema be attached. Data base information from the subschema is incorporated in the application program during compilation. To compile and execute a FORTRAN program containing DML statements, the user must perform the following steps:

To compile

Attach the subschema.

Specify the DML control statement to execute the DML preprocessor.

Execute the FORTRAN compiler, specifying that the output file produced by the DML preprocessor is the input file to the FORTRAN compiler.

To execute

Specify either the LDSET(LIB=DMSLIB) or the LIBRARY(DMSLIB) control statement.

Specify the name of the file containing the relocatable binary program; LGO is the system default file.

When the compiled program is executed, CDCS monitors and interprets all requests for action on relations and on data base files. All data base files and index files that are referenced in the subschema used by the application program are attached automatically by CDCS. If any one of these files does not exist when the program is executed, the program is aborted.

Refer to section 5 for information about execution of a FORTRAN program using CDCS. Refer to appendix G for information about compilation and execution of an application program using the CDCS Batch Test Facility (CDCSBTF).

SAMPLE JOB STRUCTURES

Figure 3-32 illustrates the control statements needed to execute the DML preprocessor and to compile and execute a FORTRAN applications program. The ATTACH statement references the subschema library containing the compiled FORTRAN subschema referenced in the FORTRAN program. The referenced subschema must be included in the master directory and CDCS must be at a system control point. The LDSET(LIB=DMSLIB) control statement causes the DMS-170 library to be made available for execution of the program. The LGO control statement initiates execution of the relocatable binary program contained on that file.

The control statements necessary to compile and execute a program are indicated in figure 3-32. To just compile a program, the necessary control statements include all those indicated except the LDSET(LIB=DMSLIB) and LGO statements. By removing these two control statements, the user can check compilation of a program before specifying execution.

RECOMPILATION GUIDELINES

Recompilation of an application program using a subschema is governed by the types of changes made to the schema. The checksum facility is the mechanism used in determining whether the changes made to the schema require the recompilation of a subschema or an application program or both. The DDL compiler produces a one-word identifying bit string, called a checksum, for each realm and area in the schema. A checksum is also generated for each subschema. The checksums are stored in the schema and subschema directories and are recorded in the master directory. The checksums are used to verify the consistency of the schema and its associated subschemas, as well as the consistency of the subschema and the application program referencing it.

When the schema is recompiled due to a modification to a data base realm or relation, the data administrator compares the checksums generated with the corresponding checksums in the previous version of the schema. If the checksums do not match, all subschemas that reference the realms or relations whose checksums have changed must be recompiled by the data administrator.

If a subschema has been recompiled and the checksum of the recompiled subschema differs from its previous checksum, an application program referencing that subschema must be processed again by the DML preprocessor and recompiled. If a FORTRAN program references an invalid checksum, CDCS aborts the program and issues a diagnostic. The application programmer can prevent the abnormal termination of a program by ensuring, prior to executing the program, that the subschema checksum in the master directory matches the checksum of the subschema used to compile the program.

STATUS CHECKING

CDCS provides various mechanisms by which extensive status information can be returned to an application program. CDCS can return status information and error codes in the data base status block if DMLDBST is called in a FORTRAN program. FORTRAN 5 programs can use ERR and END specifiers in DML

```
Job Statement
USER Statement <-------------------------- NOS Operating System only
CHARGE Statement <------------------------ NOS Operating System only
ACCOUNT Statement <----------------------- NOS/BE Operating System only
ATTACH,SSLIB/UN=num. <-------------------- NOS Operating System only
ATTACH,SSLIB,ID=DDL. <-------------------- NOS/BE Operating System only
DML,LV=F5,SB=SSLIB.
FTN5,I=DMLOUT.
LDSET,LIB=DMSLIB.
LGO.
end-of-record <--------------------------- 7/8/9 multipunch in column 1

    FORTRAN 5 Source Program Containing
    DML Statements

end-of-information <---------------------- 6/7/8/9 multipunch in column 1
```

Figure 3-32. Program Compilation and Execution With CDCS at a System Control Point

statements to specify special processing when an error or end-of-file condition occurs. CDCS provides additional status checking elements through variables and arrays that are available for user access so that information can be obtained when an error occurs. Some CDCS and CRM diagnostic codes returned in DBSTAT or in the data base status block do not necessarily represent error conditions; rather, they are informative diagnostics. The following subsections contain more information about status checking.

DATA BASE STATUS BLOCK

The FORTRAN program can include an array, called a data base status block, in the FORTRAN program to which CDCS returns data base status information. If a data base status block is provided, CDCS returns updated status information after every CDCS operation on a realm. Information returned to the data base status block includes the following:

CRM or CDCS error code

Subschema item ordinal for item-level errors

CRM code indicating file position of a realm

Function being executed when an error occurred

Rank in a relation of the realm on which either a CRM or CDCS error occurred or a special relation condition (null record or control break) occurred

Name of the realm on which an error occurred

The FORTRAN program communicates the location and length of the data base status block to CDCS by calling the DMLDBST routine described earlier in this section. The length of the data base status block must be at least 1 word and should not be greater than 11 words. CDCS returns as much information as possible in the given length. The data base status block should be of type INTEGER.

The information returned in the data base status block is shown in table 3-3.

An example of the declarations for the data base status block is shown in figure 3-33, where the data base status block is used to check for a deadlock situation. When CDCS has detected a deadlock situation and has released the locked resources of an application program, CDCS issues the deadlock error status code 663 octal to the first word of the data base status block. In the example, the files joined in relation ACCT3 are opened for input/output. The program presumably is reading a record prior to updating it, and CDCS has locked all records in the relation occurrence. A test of word one of the data base status block is included, which causes the program to enter a loop in case of deadlock. In the loop, the program attempts to reestablish the locks and checks for deadlock.

```
      PROGRAM STATBLK
      INTEGER STATUS(11),FUNCTN,RELSTAT(3)
      INTEGER REALM(3),IER,IORD,ISEV
      EQUIVALENCE (STATUS(1),IER),
     *(STATUS(2),IORD),
     *(STATUS(3),IFP),
     *(STATUS(4),ISEV),
     *(STATUS(5),FUNCTN),
     *(STATUS(6),RELSTAT(1)),
     *(STATUS(9),REALM)

      SUBSCHEMA (COMPARE)
      INVOKE
      CALL DMLDBST (STATUS,11)
      PRIVACY (CUSTFIL,MODE=IO,PRIVACY='XX999')
      OPEN (ACCT3,MODE=IO,ERR=100)
      CUSTID='CD000123456'
   30 READ (ACCT3,KEY=CUSTID)
      IF (IER .EQ. O"663")THEN
      GO TO 30
      ELSE
      .
      .
      .
  900 CLOSE (ACCT3)
      TERMINATE
      END
```

Figure 3-33. Example of a Data Base Status Block Used For Deadlock Processing

Note that the length of the data base status block is variable; CDCS updates only those words contained within the given length. If, for instance, a user program is not interested in the realm name, the last three words of the status block can be omitted. Only the eight remaining words are updated by CDCS after every data base operation.

Although the length of the data base status block is variable, the length provided must be sufficient to allow complete specification of each unit of status information, or that information will not be updated. Thus, three words are required for both the relation status (words 6, 7, and 8) and the realm name (words 9, 10, and 11). Additionally, CDCS updates words 2, 3 and 4 as a single unit that returns auxiliary status; for information to be returned in any one of these words, length must be provided for all three words. If, for example, the block length is six words, there is not enough space for CDCS to return all three words of the relation status, so no relation status is returned.

The data base status block consists of character and numeric information. Octal codes are returned in words 1 and 3. Decimal integers are returned in words 2, 6, 7, and 8. Character strings are returned as follows: a 10-character string in word 5 and a 30-character string in words 9, 10, and 11. A FORMAT statement that is used in printing words of the data base status block should appropriately specify the contents of each word that is being printed.

TABLE 3-3. CONTENTS OF THE DATA BASE STATUS BLOCK

Word Number	Contents
1	CRM or CDCS error code in octal for the last data base operation on a realm: value is zero if no error has occurred. Values 600 through 677 octal and values 730 through 777 octal are CDCS errors; all others are CRM errors. See appendix B of this manual for CDCS errors and the CYBER Record Manager Advanced Access Methods Reference Manual for CRM errors. Note that only error codes are returned. Null occurrence and control break conditions often arise in the course of normal processing. The status codes for these conditions are not returned.
2	Subschema item ordinal for CDCS item-level errors. Item-level errors include data validation errors, record mapping errors, and item-level data base procedure errors. Value is zero if no errors have occurred. The item ordinal assigned by the FORTRAN DDL compiler is identified on the subschema compilation listing.
3	CRM code in octal indicating file position of the realm when the last data base operation was performed. A file position code is returned when open, close, read, and start operations are performed. For a relation operation, the file position code indicates the position of the root realm when the last operation was performed. The following list includes the file position codes that most commonly occur during data base processing. 10 octal — End-of-key-list, which occurs when the last primary key value associated with a given alternate key has been returned during a read operation using an alternate key value. 20 octal — End-of-record, which occurs when a record has been returned during a read operation. 100 octal — End-of-information, which occurs when a sequential read operation is attempted after the previous read operation returned the last record in the file. This file position code can be used in a FORTRAN program to determine end-of-file. If a FORTRAN program uses information returned in the data base status block to determine end-of-file, the file position code should be used. The file position codes are CRM codes; further information on file position (the FP field of the file information table) is in the CYBER Record Manager Advanced Access Methods reference manual.
4	Severity of error that occurred during the last data base operation: the value is zero if no error or a nonfatal error has occurred; the value is one if a fatal error has occurred.
5	Function being performed when an error or relation condition occurred; one of the following character strings (left-justified and blank filled): ASK, BEGIN, CLOSE, COMMIT, DELETE, DROP, END, GET-ID, LOCK, LOCK-AREA, OPEN, PRIVACY, RAN-READ, RECVR-PNT, REL-NEXT, REL-READ, REL-START, REWRITE, SEQ-READ, START, UNDEFINED, UNLOCK, VERSION, WRITE Value is undefined if no error has occurred.
6	For a relation operation, the rank of the realm on which a CRM or CDCS error occurred; zero if no error has occurred. (The root realm of the relation has a rank of one.) An error on a realm during a relation read terminates the operation. Consequently, there is never more than one rank in the relation which has a CRM or CDCS error.

TABLE 3-3. CONTENTS OF THE DATA BASE STATUS BLOCK (Contd)

Word Number	Contents
7	For a relation operation, the lowest rank on which a control break occurred; value is zero if no control break has occurred. The control break condition signifies that a new record occurrence was read for the realm's parent realm in the relation. If the null occurrence condition is set for a realm, the control break condition can be assumed for that realm even though it is not set. If a control break occurs, all realms in the relation with a rank greater than the rank recorded in this word also have control break status (or a null record occurrence, since null record occurrences override control break status). See section 5 for more information about control break conditions.
8	For a relation operation, the lowest rank for which there was a null record; value is zero if no null record. A null record occurrence for a specific rank means either that at this rank no child record currences of the parent record passed the record qualification criteria, or that no child record occurrences at this rank exist for the parent record. If a null record occurs, all realms in the relation with a rank greater than the rank recorded in this word have null record occurrences. See section 5 for more information about null record occurrences.
9,10,11	Name of the realm on which an error has occurred; stored as a left-justified, blank-filled character string. Contains blanks if no error has occurred, or if the error has occurred on a non-I/O operation or an I/O operation not explicitly requested by the user.

ERR AND END SPECIFIERS

The optional error and end-of-file specifiers in the form ERR=s and END=s are allowed only in FORTRAN 5 programs. The s refers to the statement label of an executable FORTRAN or DML statement where execution is to continue if an error or end-of-file condition occurs during execution of the DML statement containing the specifier.

The ERR=s specifier can be added to the following DML statements: ASSIGNID, BEGINTRAN, COMMITTRAN, CLOSE, DELETE, DROPTRAN, FINDTRAN, LOCK, NEWVERSION, OPEN, READ, REWRITE, WRITE, and UNLOCK. If a CRM or CDCS error condition occurs during the execution of a DML statement containing this specifier, the following steps occur:

1. Execution of the DML statement terminates.
2. DML status variables are set to the CDCS or CRM error code.
3. The appropriate fields in the data base status block are set if DMLDBST has been called.
4. Execution continues with the statement labeled s.

Control is not transferred to the statement labeled s in the error specifier when a CDCS relation condition indicating a null record occurrence or control break (that is, a status code value of 627 octal and 632 octal, respectively) occurs. Unlike execution of a FORTRAN 5 statement, execution of the program is not terminated when an error condition occurs on the execution of a DML statement that does not contain the ERR=s specifier; rather, execution continues at the next executable statement.

The END=s specifier can be added only to the DML READ statement and is applicable only for a sequential read. If a DML READ statement contains this specifier and an end-of-file condition is encountered, the following steps occur:

1. Execution of the READ statement terminates.
2. DML status variables are set to 100 octal.
3. Execution continues with the statement labeled s.

For an explanation of the sequence in which the status variables are set, see the Informative Diagnostic Codes subsection later in this section.

Unlike execution of a FORTRAN 5 statement, execution of the program is not terminated when an end-of-file condition occurs on the execution of a DML READ statement that does not contain the END=s specifier; rather, execution continues at the next executable statement.

ADDITIONAL STATUS CHECKING ELEMENTS

The common block DB0000 is declared by DML in every FORTRAN program. This common block contains variables used for error and status processing. The declarations for the block are as follows:

```
INTEGER DBREALM(3), DBTSTAT,...,
    DBRnnnn(3), DBSnnnn,...
COMMON /DB0000/ DBREALM, DBSTAT,...,
    DBRnnnn(3), DBSnnnn,...
```

These variables and arrays are available for user access so that information can be obtained when an error occurs. The variables are defined as follows:

DBREALM(3)

Name of the realm on which the most recent DML operation was performed. The name is a Hollerith constant, left-justified in the 3-word array, with blank fill.

DBSTAT

Error number in octal of the most recent error; either a CDCS error number or a CRM error number. Numbers through 600 and 677 octal and numbers through 730 and 777 octal are CDCS errors; all others are CRM errors. See the appropriate reference manual for a complete list of errors. If the value of DBSTAT is zero, no error occurred. DBSTAT should be printed using an O format.

DBRnnnn(3)

Realm name, in the same format as DBREALM. One variable in this form is reserved for each realm; the number nnnn is the realm ordinal. Realm ordinals are assigned in the same order as realms are declared in the subschema; the first ordinal is 1.

DBSnnnn

Error status for realm DBRnnnn. Same format as DBSTAT. If no error occurred on the last access to realm nnnn, DBSnnnn is zero; otherwise, it is set to the number of the error that occurred.

After each DML operation, the value in DBSTAT reflects the status of the operation. If the statement has been successfully executed, the value in DBSTAT is 0. If an error has prevented successful execution of the statement, DBSTAT contains the octal value of the CRM or CDCS error or status (such as end-of-information). If DBSTAT is used, the status variable should be checked after each DML operation.

For relation processing, several status or error codes might result from execution of a single DML READ. In this case, DBREALM contains the name of the last realm read that produced a nonzero status. DBSTAT contains the value of that status or error code. The status of each realm involved in the relation can be found in the DBSnnnn field for that realm. Refer to the following subsections for a description of the status codes that can result from a READ relation.

With this scheme, the user can either determine the most recent error occurring in any realm (with DBSTAT) or the current status of a particular realm (with DBSnnnn). The DMLDBST routine provides more information than this scheme; use of the DMLDBST routine is recommended for error and status processing.

INFORMATIVE DIAGNOSTIC CODES

Some diagnostic codes returned in DBSTAT and in the data base status block do not represent error conditions; rather, the codes are informative diagnostics reporting a condition that has occurred. These conditions can be handled in the FORTRAN program. Table 3-4 lists some of the informative diagnostics that are returned in the following status elements: the data base status block, DBSTAT, and DBSnnnn. Not all of the codes indicated are returned in all of the elements. The table indicates the component of DMS-170 recognizing the condition, the code returned, the message, and the status element to which the code is returned. Those informative codes that are returned in the data base status block are returned in the first word, with the exception of the code 100 octal; exceptions concerning this code are indicated later in this subsection. Some conditions are recognized by CRM and others by CDCS; the user can refer to the CYBER Record Manager Advanced Access Methods reference manual or to appendix B for detailed information.

TABLE 3-4. INFORMATIVE DIAGNOSTIC CODES

Code (Octal)	Message	Returned to Data Base	Returned to DBSTAT and DBSnnnn	Diagnostic From
100	CANNOT SEQUENTIALLY POSITION BEYOND FILE BOUNDS		X	CRM
445	KEY NOT FOUND - FILE POSITION ALTERED - REQUEST IGNORED	X	X	CRM
506	ALTERNATE KEY NOT FOUND	X	X	CRM
627	No message. Indicates a null record occurrence was encountered on a file during relation processing.		X	CDCS
632	No message. Indicates a control break was encountered on a file during relation processing.		X	CDCS
652	AREA an ALREADY OPEN	X	X	CDCS
654	AREA an NOT OPEN	X	X	CDCS
663	DEADLOCK ON AREA an	X	X	CDCS

The following paragraphs indicate the significance of informative codes to the user.

The 100 octal diagnostic code indicates an attempt to read beyond end-of-information. This is an informative code that can be used by a FORTRAN program in determining end-of-file only when the code is returned to DBSTAT or DBSnnnn. When returned in the first word of the data base status block, this code indicates a real error. However, the data base status block provides a field (the third word) to which file position information is returned; the 100 octal file position code indicates end-of-information and can be used by a FORTRAN program in determining end-of-file. The timing of the writing of the 100 octal code to the status elements (DBSTAT, DBSnnnn, or the third word of the data base status block) can be important to the user who is considering placing count variables or other routines in association with a sequential DML READ statement. The status elements are set to 100 octal when the READ statement has been executed one additional time after the last record was read. For example, if there are 10 records being read, DBSTAT is set to 100 octal after the 11th time the READ statement is executed. Similarly, if a FORTRAN 5 application program contains the READ statement with the END=s specifier, it would be after the 11th time the READ statement is executed that execution is transferred to the statement labeled s.

The 445 octal diagnostic code occurs when no record is found whose key matches the value of the key specified in the DML READ statement.

The 506 octal diagnostic code occurs when an alternate key value is not found. CRM issues this diagnostic when a read is attempted at the end of the alternate key file.

The 627 octal diagnostic code indicates a null record occurrence on a realm. The 632 octal code indicates a control break on a realm. These conditions can occur on a relation read. These codes are not returned in the data base status block. Refer to section 5 for detailed information on relation processing.

The 652 octal diagnostic code indicates that an open was attempted on a realm already opened. The 654 octal diagnostic code indicates that a close was attempted on a realm already closed. These diagnostics can occur on the statements OPEN or CLOSE of a relation or a realm.

The 663 octal code indicates a deadlock condition; that is, a situation arising in concurrent data base access when two or more application programs are contending for a resource that is locked by one of the other programs. The 663 octal code indicates that CDCS has unlocked all locks held by the FORTRAN program. The user should provide appropriate code to handle recovery from a deadlock. Refer to section 5 for information about recovery from a deadlock.

QUERY UPDATE INTERFACE 4

Query Update is an interactive processing language that enables individuals with varying levels of technical expertise to access and manipulate a data base controlled by CYBER Data Base Control System (CDCS) and to produce special-purpose reports. In this section a basic knowledge of the concepts and usage of Query Update is assumed. Detailed information regarding the Query Update language and its capabilities is contained in the Query Update reference manual.

The Query Update interface with CDCS consists of a subschema directory, directives, and particular diagnostic messages. The subschema directory provides for access to data base files. The subschema listing provides information the Query Update user can use when performing operations on data base files.

All directives that perform operations on data base files are monitored by CDCS. Query Update used with CDCS operates in two modes:

CDCS data base access mode

CDCS catalog mode

When a Query Update-CDCS subschema is used in an INVOKE or CREATE directive, Query Update operates in CDCS data base access mode. When a Query Update-CDCS subschema is used in a VERSION directive, Query Update operates in CDCS catalog mode. Mode operations can be combined as follows:

CDCS data base access mode
CDCS catalog mode

The same subschema must be used for both modes.

CDCS data base access mode
CRM catalog mode

For CRM catalog mode, either no subschema is specified in the VERSION directive in effect or the default catalog is being used.

non-data-base mode
CDCS catalog mode

No subschema is used for data base access.

QUERY UPDATE SUBSCHEMA

The DMS-170 data base files that are to be accessed by a Query Update application program must be described in a directory called a Query Update subschema. The data administrator, working with the application programmers, is responsible for creating the subschemas.

When the subschema is compiled, a listing is produced. The listing provides information required by the application programmer to query and to update the data base. The data administrator should provide the application programmer with a compilation listing of the subschema.

Some information required to access data base files is not included in the subschema listing. It is the responsibility of the data administrator to provide the application programmer with any necessary additional information.

This subsection documents the information provided by the subschema listing and the additional information that the data administrator must provide when the information is necessary for data base processing.

If no subschema listing is provided, the Query Update user can obtain information about areas, records, and relations with the EXHIBIT directive. Refer to the EXHIBIT subsection for information about this directive.

SUBSCHEMA LISTING

The information provided by the subschema is indicated by the following alphabetic list of items. Not all items are included in every subschema; those items that always appear in the subschema listing are indicated as always being present. An example of each item is pointed out in the sample subschema shown in figure 4-1.

Alias names

The Alias Division identifies the data names and the names of records and realms that are used in the subschema in place of the name used in the schema. The name on the right of the keyword BECOMES in the AD clause is the name that must be referenced in a Query Update directive.

```
                    * SOURCE LISTING *   (82061)  DDL  3.2+564.

00001          TITLE DIVISION.
00002          SS  ACADEMIC WITHIN UNIVERSITY.
00003
00004          ALIAS DIVISION.
00005          AD  REALM CURRICULUM BECOMES TESTCURRIC.
00006          AD  DATA PROF-ID OF COURSE-REC BECOMES PROF.
00007
00008          REALM DIVISION.
00009          RD  COURSE STUDENT TESTCURRIC QUCATALOG.
00010
00011          RECORD DIVISION.
00012
00013          01  COURSE-REC.                                  ** WITHIN COURSE
00014              03  COURSE-ID       PIC X(6).                ** ORDINAL    1
00015              03  COURSE-NAME     PIC X(20).               ** ORDINAL    2
00016              03  SCHOOL          PIC X(20).               ** ORDINAL    3
00017              03  PROF            PIC X(8).                ** ORDINAL    4
00018              03  PREREQUISITE    PIC X(6).                ** ORDINAL    5
00019              03  UNITS           PIC 9(2).                ** ORDINAL    6
00020
00021          01  STUDENT-REC.                                 ** WITHIN STUDENT
00022              03  STUDENT-ID      PIC X(11).               ** ORDINAL    1
00023              03  STUDENT-NAME    PIC X(30).               ** ORDINAL    2
00024               .
00025               .
00026               .
00027              03  ZIP-CODE        PIC X(5).                ** ORDINAL    6
00028              03  PHONE           PIC X(12).               ** ORDINAL    7
00029              03  MAJOR           PIC X(20).               ** ORDINAL    8
00030
00031          01  CURR-REC.                                    ** WITHIN TESTCUR
00032              03  CONCAT-KEY.
00033                  05  STUDENT-ID  PIC X(11).
00034                  05  IDENT       PIC X(3).
00035              03  COURSE-ID       PIC X(6).                ** ORDINAL    4
00036              03  GRADE           PIC 9V9 USAGE COMP-2.    ** ORDINAL    5
00037              03  COMPLETE-CODE   PIC A.                   ** ORDINAL    6
00038              03  COMPLETE-DATE   PIC X(8).                ** ORDINAL    7
00039              03  UNITS           PIC 9V9.                 ** ORDINAL    8
00040
00041          01  QUCAT-REC.                                   ** WITHIN QUCATAL
00042              03  QUCAT-KEY       PIC X(10).               ** ORDINAL    1
00043              03  QUCAT-ITEM      PIC X(1030).             ** ORDINAL    2
00044
```

Schema name
Subschema name
Alias name
Area (realm) name
Record name
Name of associated realm
Subschema item ordinal
Data description
Data name
Concatenated key, also Group item
Major key
Catalog file

Figure 4-1. Sample Query Update Subschema (Sheet 1 of 2)

```
PRIMARY KEY   00014       COURSE-ID FOR AREA COURSE <-------------------- Primary key and
ALTERNATE KEY 00017       PROF FOR AREA COURSE                            associated realm
PRIMARY KEY   00022       STUDENT-ID FOR AREA STUDENT
ALTERNATE KEY 00029       MAJOR FOR AREA STUDENT
PRIMARY KEY   00032       CONCAT-KEY FOR AREA TESTCURRIC
ALTERNATE KEY 00035       COURSE-ID FOR AREA TESTCURRIC <---------------- Alternate key and
ALTERNATE KEY 00036       GRADE FOR AREA TESTCURRIC                       associated realm
PRIMARY KEY   00042       QUCAT-KEY FOR AREA QUCATALOG
              *****       RECORD MAPPING IS NEEDED FOR REALM -   COURSE
              *****       RECORD MAPPING IS NOT NEEDED FOR REALM -   STUDENT
              *****       RECORD MAPPING IS NEEDED FOR REALM -   TESTCURRIC
              *****       RECORD MAPPING IS NOT NEEDED FOR REALM -   QUCATALOG

              00045               RELATION DIVISION.
              00046               RN IS REL1 <------------------------------ Relation name
              00047                   RESTRICT CURR-REC WHERE (GRADE EQ 0).
                                                  ^
              *****       END OF SUB-SCHEMA SOURCE INPUT <------------------ Restriction

              *****           |       RELATION   STATISTICS          *****  Relation
RELATION 001                  |-->REL1 JOINS          AREA - STUDENT <------ Rank 1
                                                      AREA - TESTCURRIC <--- Rank 2




              -----       BEGIN SUB-SCHEMA FILE MAINTENANCE          -----

        SUBSCHEMA                              CHECKSUM
        ACADEMIC                               4021507473725135657 5 <------ Checksum

              -----       END OF FILE MAINTENANCE          -----
DDL COMPLETE.           0 DIAGNOSTICS.
  50500B CM USED.     0.293  CP SECS.
```

Figure 4-1. Sample Query Update Subschema (Sheet 2 of 2)

Alternate key

The notation ALTERNATE KEY indicates a data name that is an alternate record key. The realm for which the data name is an alternate key is also indicated.

Queries that use alternate keys are more efficient than queries that use only data names that are not keys.

Area (realm) names

Always present. The Realm Division identifies the realms available to the application program. Query Update syntax uses the word area instead of the word realm; therefore, the word area is used in the descriptions of syntax. In this description of the subschema, however, a realm is called a realm. Any area name used in a Query Update directive for processing with CDCS is, indeed, a realm name. The complete realm name is always identified in the listing with the primary key.

Unless ALL is specified, only those realms included in the Realm Division are available to the application program.

Catalog file

A catalog file for use in CDCS catalog mode is a realm. The realm must have a primary key that is 10 characters in length. The primary key is a concatenation of session-id and transmission-id. The length of the data item must be the transmission length used (that is, the length specified by the TL parameter on the QU control statement). In figure 4-1, QUCATALOG is the name of the catalog file. The transmission length is 1030 characters (the default transmission length) and is represented by QUCAT-ITEM.

Checksum

Always present. A checksum is a one-word sum generated by the DDL compiler for the subschema. The checksum of the subschema referenced and used by the Query Update application program must agree with the checksum associated with the subschema in the master directory.

Data description entry (name and description)

Always present. Data description entries are entries identified by a level number 02 or greater. Data description entries identify the name and description of data items. The sample subschema includes only a few of the data description entries that could be in a subschema. For detailed information about data description entries, refer to the CDCS 2 Data Administration reference manual.

Data type in the subschema is determined by a PICTURE clause and an optional USAGE clause. Table 4-1 lists keywords used in the USAGE clause to describe data type and the corresponding Query Update data type. If no USAGE clause is present, data type is determined by the PICTURE clause. Refer to the Query Update reference manual for a description of Query Update data types.

TABLE 4-1. CORRESPONDING SUBSCHEMA AND QUERY UPDATE DATA TYPES

Subschema Data Description	Query Update Data Type
COMPLEX	COMPLEX
COMPUTATIONAL or COMP	NUMERIC
COMPUTATIONAL-1 or COMP-1	INTEGER
COMPUTATIONAL-2 or COMP-2	FLOATING
DISPLAY	CHARACTER
DOUBLE	DOUBLE
INDEX	INTEGER
LOGICAL	LOGICAL
No type specified; PICTURE clause with 9's only	NUMERIC
No type specified; PICTURE clause with mix of X's, A's, and 9's or all X's or A's	CHARACTER

Only data represented by data description entries are available to the application program. Data received in the working storage area of an application program is mapped according to the subschema description.

Group item

A group item is made up of other data items. The lower level number, 03 in the sample subschema, denotes the group item name. The group item name can be used in Query Update directives to qualify the name of a data item that is in the group item.

Major key

A major key is the leading data name of a group item that is defined as a primary or alternate key. In this example, CONCAT-KEY is the group item that is defined as a primary key. STUDENT-ID is the major key and can be used in major key processing.

Primary key

Always present for each realm included in the subschema. The notation PRIMARY KEY indicates each data name that is a primary record key. The realm for which the data name is a primary record key is also indicated.

Queries that use primary keys are more efficient than queries that do not use them.

Ranks of the relation

The realms joined in a relation are listed by realm name in order of rank. The first realm listed has a rank of 1; the second, a rank of 2; and so forth.

Realm names

See area names and catalog file (described earlier in this section).

Record names, associated realms and data names

Always present. An 01 level data description identifies the record name. Each record available to the application program is included in the subschema listing.

The notation WITHIN realm name identifies the particular realm associated with a record by giving the first seven characters of the realm name.

The data descriptions of 02 level or greater that follow the record name identify the data names that comprise the record.

The record name can be used in Query Update directives to qualify any data name within the record.

Relation names

The RN clause in the Relation Division identifies a relation name. An RN clause is included for each relation available to the application program.

Restriction

A RESTRICT clause in the Relation Division identifies a restriction. If a restriction is placed on a relation, only records that meet all the qualification criteria specified in the RESTRICT clause are returned to the application program when the data base is queried via a relation.

If the RESTRICT clause contains a data name that is not defined in the subschema, the data name must be defined in a DEFINE directive with the same size and type as the subschema item used for comparison. The data name must be set to a value in the Query Update program before any query is entered that uses the relation.

Schema name

Always present. The SS clause in the Title Division identifies the schema. The schema name follows the keyword WITHIN.

Subschema item ordinal

Always present. The subschema item ordinal is a unique identifier within a record that is assigned to each data name in a subschema when the subschema is compiled. A subschema item ordinal can be returned in an error message.

Subschema name

Always present. The SS clause in the Title Division identifies the subschema. An application program must reference the subschema by using the subschema name.

INFORMATION PROVIDED BY DATA ADMINISTRATOR

It is the responsibility of the data administrator to provide the following information when it is required for data base processing:

Access keys

If an area has been defined with an access control lock in the schema, the ACCESS directive must be used to gain access to a realm. The application programmer must be provided with all access control keys required for data base access. (Refer to the description of the ACCESS directive for more information.)

Constraints

If constraints are defined in the schema and updates are likely to violate them, the application programmer should be provided with information about the constraints.

Data base version name

If alternate data base versions are defined for the data base, the application programmer must be provided with information about the name and use of the alternate data base versions.

Join items of the relation

Files are joined in a relation by identical data names that exist in two files. The information about the join items is contained in the schema. Usually, an application programmer does not need to know the join items to use a relation. Often the programmer can determine the join items from the subschema listing. (Refer to section 5 for more information on relations.) In some situations, however, the data administrator should provide the application programmer with the join items.

Requirements imposed by any data base procedure

If data base procedures are defined in the schema, the application programmer must be provided with information required by the data base procedures.

SUBSCHEMA DIRECTORY

When the subschema source input is compiled, the object subschema, called the subschema directory, is generated. The subschema directory is usually included in a subschema library.

The subschema directory must be available to Query Update for data base access; therefore, the user must include the file identification for the subschema library in CREATE, INVOKE, and VERSION directives. During processing, Query Update uses the descriptions of areas, data names, and relations that are included in the subschema.

DIRECTIVE SYNTAX

Query Update language elements are grouped together to form directives. The language elements include the following:

Reserved words

Recognized symbols

Punctuation

User-supplied elements, such as names, literals, expressions, conditions, and picture specifications

Refer to the Query Update reference manual for definitions of the language elements.

The Query Update interface to CDCS consists of 18 directives that operate directly on data base files (including CDCS-catalog files) or provide access to data base files. All Query Update directives (but not all formats of each directive) can be used when processing with CDCS.

The entire set of Query Update directives that are available for use with CDCS is shown in table 4-2. The directives that are used in CDCS data base access mode and in CDCS catalog mode are marked in the table.

Only directives that have unique requirements for processing with CDCS are described in this manual. The syntax is shown for all directives that provide for data base access when used in CDCS data base access mode.

Any data name in a directive can be qualified by a group name or record name; if there is confusion about which elementary item a data name references, Query Update requires qualification.

Several format statements shown in this section include an expression. For reference purposes, the format of an expression is shown in figure 4-2. A data name in an expression can be a data base item.

To provide for future system software and to maintain compatibility with earlier releases of Query Update, Query Update supports (for some operations) two directives that perform virtually the same operation. Table 4-3 contains both directives: the recommended directive and the old directive. This manual describes the use of only the recommended directives.

TABLE 4-2. QUERY UPDATE DIRECTIVES

Directive	Directive	Directive
ACCESS†	FORMAT	RETURN
ALTER	HEADING	REWIND
BREAK	HELP	SELECT
COMPILE	IF	SEPARATOR
CREATE†,††	INVOKE†,††	SORT
DATE	MODIFY†	SPECIFY
DEFINE	MOVE	STORE†
DESCRIBE	NOTE	SUMMARY
DETAIL	OS	TABS
DIAGNOSTIC	PAGE-NUMBER	TIME
DISPLAY†	PAGE-SIZE	TITLE
DUPLICATE	PERFORM	UNIVERSAL
END	PREFACE	UPDATE†
ERASE	PREPARE	VERIFY
EVALUATE	PREVIEW	VERSION†,†††
EXECUTE	RECAP	VETO
EXHIBIT†	RECORDING	VIA†
EXTRACT†	RECOVERY†	
FOOTING	REMOVE†	

†Documented in this manual because of particular use in accessing a CDCS data base.

††Establishes or terminates CDCS data base access mode.

†††Establishes or terminates CDCS catalog mode.

TABLE 4-3. RECOMMENDED QUERY UPDATE DIRECTIVES

Recommended Directive	Old Directive
INVOKE	USE
MODIFY	UPDATE
REMOVE	DELETE
STORE	INSERT

ACCESS

The ACCESS directive, shown in figure 4-3, supplies CDCS with an access key for areas and catalog files. If access control locks have been defined in the schema for the areas and catalog files, the ACCESS directive must be used to provide the access key. The literal or data name must not exceed 30 characters.

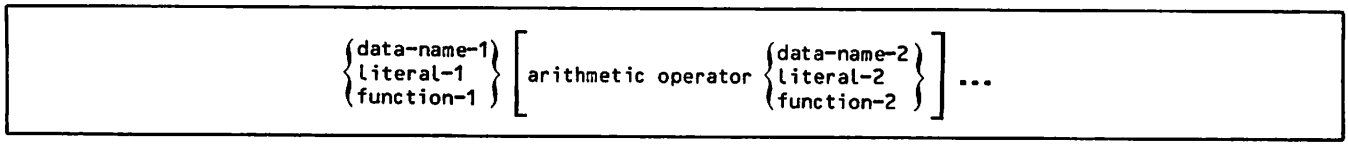


Figure 4-2. Format of an Expression

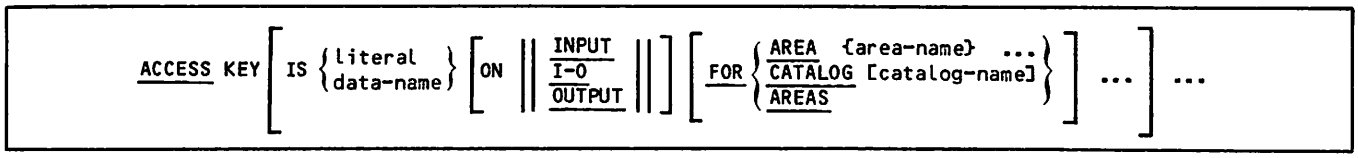


Figure 4-3. ACCESS Directive Format

The ACCESS directive must be entered after a subschema has been specified in an INVOKE, CREATE, or VERSION directive and before any area or catalog file with an access control lock has been accessed.

The ON clause indicates the kind of usage the access key is to provide. The access key is for access locks defined in the schema.

The terms used in the schema to define access are update and retrieval. Access locks in the schema can be defined so that a single access key provides for access as follows: only update, only retrieval, or both update and retrieval. Table 4-4 indicates the usage option that provides the key for a particular schema access lock.

TABLE 4-4. SCHEMA ACCESS CONTROL LOCKS AND CORRESPONDING ACCESS DIRECTIVE USAGE OPTIONS

Schema Access Control Lock	Usage Options
Update	OUTPUT
Retrieval	INPUT
Update/Retrieval†	I-O

†The same access key provides access for either input or output or for both.

If access keys are defined in the schema, Query Update directives require that access keys be specified to provide access as follows:

Retrieval access

DISPLAY and EXTRACT directives used with the INVOKE directive

Update access

STORE directive used with the CREATE directive

Update and retrieval access

MODIFY, REMOVE, and STORE directives used with the INVOKE directive

DUPLICATE, ERASE, EXHIBIT, FORMAT, PERFORM, PREPARE, and RECORDING directives when used with the VERSION directive in CDCS catalog mode

Any program whose access keys provide access to an area for both update and retrieval (either by providing two keys, one ON INPUT and another ON OUTPUT, or by providing one key ON I-O) has met the access requirement for only update or for only retrieval.

If the ON clause is omitted, ON I-O is assumed.

The FOR clause specifies the area names or the catalog file name whose access is controlled by a particular access key. If AREA is used, the specified access key applies only to the specified area. If AREAS is used, the access key specified applies to all areas in the subschema that can be accessed by the specified access key (but not to the catalog file). If CATALOG is used, the access key specified applies only to the catalog file. If the FOR clause is omitted, both the AREAS and CATALOG options are assumed.

The IS clause and subsequent clauses can be omitted. When Query Update encounters a transmission that contains only the word ACCESS or the words ACCESS KEY, terminal input is prompted with the symbols >>>. The user then enters the access key and remaining optional clauses. Query Update continues to prompt after each transmission until the user enters a transmission consisting of *END. By using the prompting feature of this directive, the user protects the security of the access keys; this feature inhibits the printing of access keys on the output listing or trace file.

Examples of the ACCESS directive with access keys being entered after prompts from Query Update are shown in figure 4-4. In the first example, the access key SEEOLDINFO provides access for retrieval; the access key NEWINFO provides access for update. Both access keys must be specified to remove, store, or modify records in the area STUDENT. In the second example, the access key XX99 provides access for both update and retrieval of the area TESTCURRIC.

Example 1:

```
-- ACCESS
>>> $SEEOLDINFO$ ON INPUT FOR AREA STUDENT
>>> $NEWINFO$ ON OUTPUT FOR AREA STUDENT
>>> *END
```

Example 2:

```
-- ACCESS
>>> $XX99$ ON I-O FOR AREA TESTCURRIC
>>> *END
```

Figure 4-4. ACCESS Directive Examples

One specification of a particular access key in an ACCESS directive can provide access to a catalog file and to all areas that can be accessed by that particular access key. If two access keys are required for particular areas or catalog file, both access keys must be specified in access directives before using the areas or catalog file.

CREATE

The CREATE directive, shown in figure 4-5, initiates processing in CDCS data base access mode, identifies the subschema, identifies the data base version, and creates a data base file. The data base file created by the CREATE directive must have been previously cataloged or defined as an empty permanent file by the data administrator. Upon execution of this directive, all the files associated with the subschema and data base version are attached by CDCS. Query Update attaches the subschema file.

```
CREATE area-name OF subschema-name
    [FROM LIBRARY permanent-file-name]
    [(permanent-file-parameter [PW])]
    [FOR DATABASE version-name]
```

Figure 4-5. CREATE Directive Format

Only one CREATE directive can be in effect at a time. When a CREATE directive is entered, the effect of any prior CREATE or INVOKE directive is terminated; that includes terminating the effect of access keys previously entered and releasing the subschema file and tables established to use the subschema specified in the previous INVOKE or CREATE directive. The CREATE directive can be entered any number of times within a single execution of the QU control statement as long as the same area and version are not duplicated. When used in a transmission with other directives, the CREATE directive must be the last directive in a transmission.

The area-name identifies the area to be created. The area-name must be an area included in the subschema referenced by subschema-name.

The subschema-name identifies the subschema being used. The subschema-name must be the name of a Query Update-CDCS subschema. When CDCS catalog mode is active, the subschema must be the same subschema used for the VERSION directive.

The FROM LIBRARY option identifies the subschema library that contains the subschema directory being used. If the FROM LIBRARY option is omitted, the permanent file name of the subschema library is assumed as follows: for NOS, the first seven characters of the subschema name are used; for NOS/BE, the entire subschema name is used.

When permanent file parameters are required to access the subschema, they must be specified in the CREATE directive. The PW option can be specified to protect the security of passwords. When Query Update encounters the characters PW alone, terminal input is prompted with the symbols >>>. The user then enters the appropriate passwords. Query Update continues to prompt after each transmission until the user enters a transmission consisting of *END. The PW option protects the security of the passwords since printing of passwords on the output listing or trace file is inhibited.

Examples of the CREATE directive using the PW option are shown in figure 4-6. Area TESTCURRIC of subschema ACADEMIC from subschema library QULIB is being used. The password PASS must be specified for access to the subschema library (read access only is required).

The FOR DATABASE option identifies the data base version being used. Version MASTER is assumed when the option is omitted with one exception: if a preceding VERSION directive that is in effect specified an alternate data base version, that alternate version is assumed. If a preceding VERSION directive is in effect and the FOR DATABASE option is specified, the same data base must be specified in both the VERSION and CREATE directives. The version name must be a 1- through 7-character name of a data base version included in the master directory as a version of the schema being used; MASTER is the name of the default data base version. The data administrator must provide the user with valid alternate version names when data base versions are being used.

If the files have not been established, they must be established before beginning the Query Update session that creates the files. The data administrator must provide permanent file names and other permanent file information to be used to create the files. The permanent information must be the same as recorded in the master directory for the particular area and data base version.

Under the NOS operating system, the DEFINE control statement must be used to make a data base file permanent. For example, the following statements can be used under NOS:

```
DEFINE,permanent-file-name.
RETURN,permanent-file-name.
QU.
CREATE,area-name...
```

Example 1. For Use on NOS

```
-- CREATE TESTCURRIC OF ACADEMIC FROM LIBRARY QULIB (UN=QUDCDCS2) PW
   >>> QULIB,PASS
   >>> *END
```

Example 2. For Use on NOS/BE

```
-- CREATE TESTCURRIC OF ACADEMIC FROM LIBRARY QULIB (ID=QUCDCS2) PW
   >>> QULIB,PASS
   >>> *END
```

Figure 4-6. CREATE Directive Examples

Under the NOS/BE operating system, the CATALOG control statement is used to make the data base files permanent. For example, the following statements can be used:

```
REQUEST,permanent-file-name,PF.
REWIND,permanent-file-name.
CATALOG,permanent-file-name.
RETURN,permanent-file-name.
QU.
CREATE,area-name...
```

DISPLAY

The DISPLAY directive, shown in figure 4-7, retrieves and displays information from the data base at a terminal or upon a designated file.

The information displayed depends on the subschema and data base version in use. Refer to the Query Update reference manual for detailed information about the DISPLAY directive.

EXHIBIT

The EXHIBIT directive, shown in figure 4-8, displays information that Query Update uses in performing its operations. All options of the EXHIBIT directive are applicable for use with CDCS. Only options that provide information about the data base are described in the following paragraphs.

Information about the data base is exhibited for the subschema specified in the preceding INVOKE or CREATE directive. If a VERSION directive precedes the EXHIBIT directive, the information about the catalog file is exhibited for the catalog specified in the VERSION directive; otherwise, the information about the catalog file is exhibited for the default catalog ZZZZZQ2.

The type of information displayed depends upon the options specified. The options that can cause information to be displayed about the data base are as follows: data name, SESSION, RELATION. Figure 4-9 illustrates the EXHIBIT directive with the following options specified: no option, area name, record name, relation, and a session.

When no options are included in the directive, Query Update exhibits the maximum transmission length; the transmission length of the catalog file; the universal character; the current literal delimiter and ITEM-SIZE if it is in effect; the current subschema, subschema library, and area names; and the current or default maximum number of lines, columns, sections, and images permitted in a report.

The data name option causes a display of information related to the type of data name specified. Data name options cause data base information to be displayed as follows:

Area name

> Record names in the area, primary, alternate, and major keys for each record.

Record name

> Level numbers, item names, and group, elementary, primary key, alternate key, or major key indicators.

Relation name

> The record name and associated area name for each area joined in the relation. The areas are listed in the order of rank the area has in the relation: the first area (root area) has rank 1; the second, rank 2; and so forth.

Group item name

> Number of occurrences and elementary item names within the group.

Elementary item name

> Size, type, and possible qualifiers up to the record name.

The RELATION options lists the names of the relations included in the subschema and, therefore, available to the user. Following the name of each relation, the record name and corresponding area names of each area joined in the relation are listed as described in the preceding data name option, relation name.

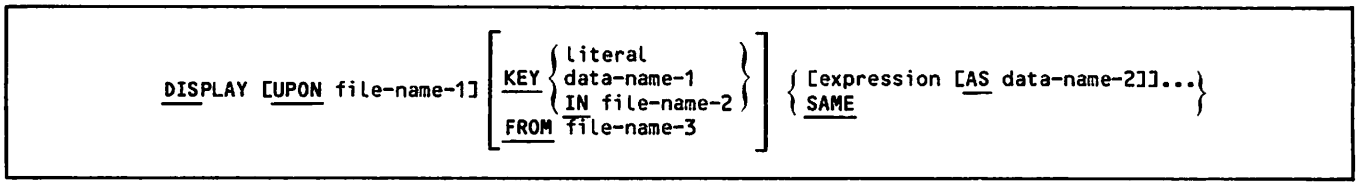


Figure 4-7. DISPLAY Directive Format

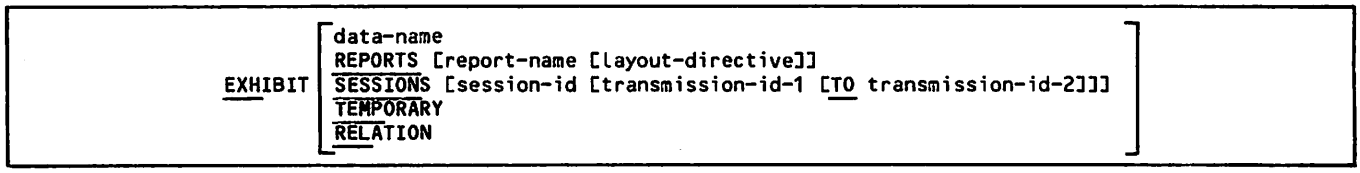


Figure 4-8. EXHIBIT Directive Format

Example 1. Exhibit (No Option Specified)

```
MAXIMUM TRANSMISSION LENGTH    1030
TL OF CATALOG FILE    1030
UNIVERSAL OFF
MAX MUNBER OF LINES  060
MAX NUM. OF COLUMNS  130
MAX NO. OF SECTIONS  010
MAX IMAGER PER PAGE  004
AREA NAME(S):
   COURSE
   STUDENT
   TESTCURRIC
   QUCATALOG
SUBSCHEMA NAME = ACADEMIC
SUBSCHEMA LIBRARY NAME = QULIB
    ID = QUD0461
```

The transmission lengths in use and the area names, subschema name, and subschema library of the current INVOKE directive are shown

Example 2. Exhibit an Area

```
-- EXHIBIT TESTCURRIC
RECORD NAME IS     CURRIC-REC
KEY IS   CONCAT-KEY
MAJ KEY  STUDENT-ID
ALT KEY  COURSE-ID
ALT KEY  GRADE
AREA NAME = TESTCURRIC
```

The area name TESTCURRIC along with the record name, all keys defined for the area (primary, alternate, and major) are shown.

Example 3. Exhibit a Record

```
-- EXHIBIT CURR-REC
03 (KEY) CONCAT-KEY
05 (MAJ) STUDENT-ID
05 (ELM) IDENT
03 (ALT) COURSE-ID
03 (ALT) GRADE
03 (ELM) COMPLETE-CODE
03 (ELM) COMPLETE-DATE
03 (ELM) UNITS
```

The record name is shown with each data name in the record and its level number. All keys defined for the area (primary, alternate, and major) are indicated.

Example 4. Exhibit Relation

```
REL1 RELATES THE RECORDS:
   STUDENT-REC IN STUDENT
   CURR-REC IN TESTCURRIC
```

The relation name is shown together with each record and its associated area name joined in the relation. The areas are shown in order of its rank in the relation. The first area listed has rank 1; the second, rank 2, and so forth.

Example 5. Exhibit a Session

```
-- EXHIBIT SESSION SESS1
  1      INVOKE ACADEMIC FROM LIBRARY QULIB (ID=QUCDCS2)
  6      DISPLAY $ENTER ACCESS KEY FOR AREA TESTCURRIC$
 11      ACCESS
 16      DISPLAY $STUDENT-ID COURSE-ID$
 21      DISPLAY STUDENT-ID OF CURR-REC COURSE-ID OF CURR-REC
```

The session name is listed along with each transmission included in the session.

Figure 4-9. EXHIBIT Directive Examples

EXTRACT

The EXTRACT directive, shown in figure 4-10, creates a subset (a sequential file) of a data base area or data base relation and a directory to the subset. The information extracted depends on the data names specified in the directive and on the subschema and data base version in use. Refer to the Query Update reference manual for detailed information about the EXTRACT directive.

```
EXTRACT UPON file-name {expression [AS data-name]} ...
```

Figure 4-10. EXTRACT Directive Format

INVOKE

The INVOKE directive, shown in figure 4-11, initiates processing in CDCS data base access mode, identifies the subschema, and identifies the data base version. Upon execution of the INVOKE directive, all the data base files associated with the subschema and data base version are attached by CDCS and all areas and relations included in the subschema are made available. Query Update attaches the subschema file.

```
INVOKE subschema-name
    [OF schema-name]
    [FROM LIBRARY permanent-file-name]
    [(permanent-file-parameters [PW])]
    [FOR DATABASE version-name]
```

Figure 4-11. INVOKE Directive Format

An example of the INVOKE directive is shown in figure 4-12. Subschema ACADEMIC from subschema library QULIB is being used. Data base version ALTDB is specified. No passwords are required to read the subschema library file.

Only one INVOKE directive can be in effect at a time. When an INVOKE directive is entered, the effect of any prior CREATE or INVOKE directive is terminated; that includes terminating the effect of access keys previously entered and releasing the subschema file and tables established to use the subschema specified in the previous INVOKE or CREATE directive. The INVOKE directive can be entered any number of times within a single execution of the QU control statement.

The subschema-name identifies the subschema being used. The subschema-name must be the name of a Query Update-CDCS subschema. When CDCS catalog mode is active, the subschema must be the same subschema used for the VERSION directive.

The schema-name identifies the schema. Schema-name is ignored by Query Update. Schema-name can be used by the user for documentary purposes and for compatibility with future versions of Query Update.

The FROM LIBRARY option identifies the subschema library that contains the subschema directory being used. If the FROM LIBRARY option is omitted, the permanent file name of the subschema library is assumed as follows: for NOS, the first seven characters of the subschema name are used; for NOS/BE, the entire subschema name is used.

When permanent file parameters are required to access the subschema, they must be specified in the INVOKE directive. The PW option can be specified to protect the security of passwords. When Query Update encounters the characters PW alone, terminal input is prompted with the symbols >>>. The user then enters the appropriate passwords. Query Update continues to prompt after each transmission until the user enters a transmission consisting of *END. The PW option protects the security of the passwords since printing of passwords on the output listing or trace file is inhibited.

The FOR DATABASE option identifies the data base version being used. Version MASTER is assumed when the option is omitted with one exception: if a preceding VERSION directive that is in effect specified an alternate data base version, that alternate version is assumed. If a preceding VERSION directive is in effect and the FOR DATABASE option is specified, the same data base must be specified in both the VERSION and INVOKE directives. The version-name must be a 1- through 7-character name of a data base version included in the master directory as a version of the schema being used; MASTER is the name of the default data base version. The data administrator must provide the user with valid alternate version names when data base versions are being used.

MODIFY

The MODIFY directive, shown in figure 4-13, modifies the values of data items in existing records. Only one area can be modified at a time; each area joined in a relation must be modified separately. A record is selected for modification either by referencing the record key in the MODIFY directive or as a result of an IF directive with an associated MODIFY directive.

Example 1. For Use on NOS

```
INVOKE ACADEMIC FROM LIBRARY QULIB (UN=QUCDCS2) FOR DATABASE ALTDB
```

Example 2. For Use on NOS/BE

```
INVOKE ACADEMIC FROM LIBRARY QULIB (ID=QUCDCS2) FOR DATABASE ALTDB
```

Figure 4-12. INVOKE Directive Examples

```
MODIFY [record-name]
    || [SETTING {data-name-1}...] [USING key-name] [SETTING {data-name-2}...] [FROM file-name] ||
    || MOVE expression-1 TO {data-name-3}... [AND expression-2 TO {data-name-4}...]... ||
    [VETO]
    [PASS]
```

Figure 4-13. MODIFY Directive Format

The use of the MODIFY directive with CDCS is the same as the use of the directive to modify a data base area with CRM (as described in the Query Update reference manual) with one exception: providing for access requirements to modify the area. If an area is defined in the schema with controlled access, the user must provide the access keys in an ACCESS directive. Refer to the discussion of the ACCESS directive for more information.

The file accessed depends on the subschema and data base version in use. The primary, alternate, or major key must be specified in the USING option if data base items are to be modified. Modifying by major or alternate key can result in several records being updated at one time. All records containing the specified major or alternate key value are updated in the same way. No record in a CDCS controlled file can have a primary key value that duplicates another primary key value within the file.

The user is responsible for maintaining the integrity of the data base. Updating operations must be consistent with the meaning of the relation defined for the areas. Refer to section 5 for guidelines on updating files joined in a relation. Constraints can affect updating operations. Refer to section 5 for guidelines on updating files associated in a constraint.

RECOVERY

The RECOVERY directive, shown in figure 4-14, establishes a recovery point on the journal log file.

```
RECOVERY POINT USING data-name-1 { literal    }
                                 { data-name-2 }
```

Figure 4-14. RECOVERY Directive Format

Upon execution of the RECOVERY directive, data-name-1 contains a unique recovery point number assigned by CDCS. Data-name-1 must be a defined integer item.

Literal or data-name-2 must be a defined 30 character type CHARACTER field that contains a user-supplied explanatory message (up to 30 characters in length) for the recovery point. The message is written on the journal log file at the same time the recovery point number is established.

CDCS responds to a RECOVERY directive by halting all use of the schema and suspending execution of the Query Update program until the following events occur:

All I/O buffers for data base areas are cleared.

A recovery point log entry is written to the log file for the data base.

The quick recovery file for the data base is emptied.

The RECOVERY directive should be used judiciously because considerable overhead is involved. The data administrator should be consulted to determine if the use of recovery points is appropriate for a particular file.

REMOVE

The REMOVE directive, shown in figure 4-15, removes specific records from a data base area. Remove operations can be performed on only one area at a time; a separate remove operation must be performed on each area joined in a relation. The complete record is removed from the data base; the REMOVE directive does not remove part of a record.

```
REMOVE [record-name]
    [SETTING {data-name-1}...]
    USING key-name
    [SETTING {data-name-2}...]
    [FROM file-name]
    VETO
    PASS
```

Figure 4-15. REMOVE Directive Format

The use of the REMOVE directive with CDCS is the same as the use of the directive to remove a record from a data base area with CRM (as described in the Query Update reference manual) with one exception: providing for access requirements to modify the area. If the area is defined in the schema with controlled access, the user must provide the access keys in an ACCESS directive. Refer to the discussion of the ACCESS directive for more information.

The file accessed depends on the subschema and data base version in use. Removing records by major or alternate key can result in several records being removed at one time. All records containing the specified major or alternate key value are removed in the same way.

The user is responsible for maintaining the integrity of the data base. Updating operations must be consistent with the meaning of the relation defined for the areas. Refer to section 5 for guidelines on updating files joined in a relation. Constraints can affect updating operations. Refer to section 5 for guidelines on updating files associated in constraints.

STORE

The STORE directive, shown in figure 4-16, creates a new record and places it in an area of the data base; record values are established. Only one area can be modified at a time; each area joined in a relation must be modified separately. Data can be input from the terminal or from a designated file.

The use of the STORE directive with CDCS is the same as the use of the directive to store records in a data base area with CRM (as described in the Query Update reference manual) with one exception: providing for access requirements to modify the area. If the area is defined in the schema with controlled access, the user must provide the access keys in an ACCESS directive. Refer to the discussion of the ACCESS directive for more information.

The file accessed depends on the subschema and data base version in use. The primary key must be included in the data-name-1, data-name-2, or data-name-3 list if data base items are referenced. The primary key does not have to be specified if the area referenced has actual key file organization. (The data administrator can provide this information.) No record in a CDCS controlled file can have a primary key value that duplicates another primary key value within the file.

The user is responsible for maintaining the integrity of the data base. Storing operations must be consistent with the meaning of the relation defined for the areas. Refer to section 5 for guidelines on updating files joined in a relation. Constraints can affect updating operations. Refer to section 5 for guidelines on creating and updating files associated in constraints.

UPDATE

The UPDATE directive, shown in figure 4-17, permits an area to be identified for updating before a REMOVE, STORE, or MODIFY directive is entered.

```
UPDATE area-name
```

Figure 4-17. UPDATE Directive Format

The area-name must be an area included in the subschema in use.

Refer to the Query Update reference manual for the description of this directive.

VERSION

The VERSION directive determines whether Query Update enters either the CRM catalog mode or the CDCS catalog mode. Format 3 of the VERSION directive, shown in figure 4-18, initiates processing in CDCS catalog mode, identifies the subschema, and identifies the data base version. (Formats 1 and 2 are documented in the Query Update reference manual.)

```
VERSION is catalog-file OF subschema-name
    [FROM LIBRARY permanent-file-name]
    [(permanent-file-parameters [PW])]
    [FOR DATABASE version-name]
```

Figure 4-18. VERSION Directive Format

Either CRM or CDCS catalog mode can be used together with processing in CDCS data base access mode. Refer to the subsection Catalog File for more information and an example of using a catalog file.

When format 3 of the VERSION directive is used, the specified subschema file is attached (if not already attached). Query Update checks that the catalog file is described correctly within the subschema. To be described correctly, the transmission length specified in the QU control statement must be the same as the length of the data item in the catalog file (see the sample subschema in figure 4-1). The specified catalog file is used for all reports and sessions. In addition, the default catalog is available for duplication.

```
STORE [record-name]
    || SETTING {data-name-1}... [FROM file-name]                                  ||
    || MOVE expression-1 TO {data-name-2}... [AND expression-2 TO {data-name-3}...] ... ||
    [VETO]
    [PASS]
```

Figure 4-16. STORE Directive Format

Only one VERSION directive can be in effect at a time. If a format 3 VERSION directive is entered (called the current VERSION directive) after a VERSION directive had been entered, the following situations apply:

Any access keys entered to use the catalog file specified in the first VERSION directive are no longer in effect. An ACCESS directive must be entered if access keys are required to use the catalog file specified in the current VERSION directive.

If the first VERSION directive initiated CRM catalog mode, the CRM catalog file is released upon execution of the current VERSION directive.

If the first VERSION directive initiated CDCS data base access mode, and if the current VERSION directive uses the same subschema and data base version, Query Update sets internal descriptions to indicate which file is the catalog file. The previous INVOKE remains in effect.

Any previous subschema file and all tables describing a previous subschema, areas, and catalog files are released under the following conditions:

CDCS data base access mode was in effect with either a different subschema or different data base version. The previous INVOKE is no longer in effect.

CDCS catalog mode was in effect with either a different subschema or data base version.

CRM data base access mode was in effect. (That is, a Query Update-CRM subschema was being used.) The previous INVOKE is no longer in effect.

The catalog-file identifies the catalog being used. Catalog-file must be an area name included in the subschema identified by subschema-name.

The subschema-name identifies the subschema being used. The subschema name must be the name of a Query Update-CDCS subschema.

The FROM LIBRARY option identifies the subschema library that contains the subschema directory being used. If the FROM LIBRARY option is omitted, the permanent file name of the subschema library is assumed as follows: for NOS, the first seven characters of the subschema name are used; for NOS/BE, the entire subschema name is used.

When permanent file parameters are required to access the subschema, they must be specified in the VERSION directive. The PW option can be specified to protect the security of passwords. When Query Update encounters the characters PW alone, terminal input is prompted with the symbols >>>. The user then enters the appropriate passwords. Query Update continues to prompt after each transmission until the user enters a transmission consisting of *END. The PW option protects the security of the passwords since printing of passwords on the output listing or trace file is inhibited.

The FOR DATABASE option identifies the data base version being used. Version MASTER is assumed when the option is omitted. The version name must be a 1- through 7-character name of a data base version included in the master directory as a version of the schema being used; MASTER is the name of the default data base version. The data administrator must provide the user with valid alternate version names when data base versions are being used.

VIA

The VIA directive, shown in figure 4-19, is used in relation processing to specify to Query Update which relation should be followed when an ambiguity exists.

```
VIA relation-name
```

Figure 4-19. VIA Directive Format

The relation-name must be included in the subschema in use.

Refer to the Query Update reference manual for the description of this directive.

QUERY UPDATE PROCESSING WITH CDCS

Query Update-CDCS interface operates on NOS and NOS/BE in both batch and interactive modes.

CDCS knows the Query Update program by the identification contained in the USER-ID register. CDCS provides error diagnostics, a validation facility for the subschema used, and a record locking mechanism.

EXECUTION

Execution of Query Update is called by the QU control statement. A control statement in the following form can be used:

```
QU
```

Parameters can be included in the QU control statement. Refer to the Query Update reference manual for more information.

The Query Update program initiates the interface with CDCS when a subschema is specified in an INVOKE, CREATE, or VERSION directive. CDCS must be active in the system for processing with CDCS to occur. If CDCS is not active, a message is returned to an interactive program; a batch job is held by the system and is executed when CDCS becomes active.

Refer to section 5 for information about execution time processing with CDCS.

CATALOG FILE

Accessing a catalog file is necessary whenever a Query Update user performs a previously prepared session, generates a report, or changes a report. When Query Update is using a catalog file, Query Update is processing in a catalog mode. Processing a catalog file can be performed in one of two modes: CDCS catalog mode or CRM catalog mode.

The field length required for Query Update for processing in CRM catalog mode is greater than that required for processing in CDCS catalog mode. Query Update must use CYBER Record Manager Advanced Access Methods (AAM) in processing a catalog file. In CRM catalog mode, AAM is loaded and used within the field length of Query Update. In CDCS catalog mode, AAM is loaded and used within the field length of CDCS.

The VERSION directive determines whether Query Update enters CRM catalog mode or CDCS catalog mode. To establish processing in CDCS catalog mode, a subschema must be specified in the VERSION directive.

To make processing in CDCS catalog mode possible, a catalog file must be described as an area in the Query Update subschema and as an area in the corresponding schema. Refer to figure 4-1 for an example of a catalog file described in the subschema.

Figure 4-20 shows creating and using a CDCS catalog file. Example 1 shows creating the catalog file. To create the file, a session (SESS1 in the example) must be recorded on the default catalog, which is always local file ZZZZZQ2. For NOS, the permanent file must be defined and the contents of the default catalog copied to the permanent file. For NOS/BE, the local file can be cataloged with the name of the permanent file. For both operating systems the permanent file name, identification, and passwords must be those recorded for the area (catalog file) and data base version in the master directory. The data administrator should provide the appropriate permanent file information.

Example 1. Creating a CDCS Catalog File

```
COMMAND- QU

QUERY UPDATE 3.4 R2C-82061
-- RECORDING SESS1 BY 5
     200 TRANSMISSION(S) ALLOWED
001-- INVOKE ACADEMIC FROM LIBRARY QULIB (ID=QUCDCS2)
006-- DISPLAY $ENTER ACCESS KEY FOR AREA TESTCURRIC$
011-- ACCESS
016-- DISPLAY $STUDENT-ID COURSE-ID$
021-- DISPLAY STUDENT-ID OF CURR-REC COURSE-ID OF CURR-REC
026-- RECORDING OFF
      END OF SESSION SESS1
-- END
**CAUTION**
DEFAULT CATALOG REMAINS AS LOCAL FILE ZZZZZQ2
```

NOS

```
REWIND,ZZZZZQ2
DEFINE,MCATFIL
COPY,ZZZZZQ2,MCATFIL
```

NOS/BE

```
CATALOG,ZZZZZQ2,MCATFIL,ID=QUCDCS2
```

Example 2. Executing a CDCS Catalog File

```
COMMAND- QU

QUERY UPDATE 3.4 R2C-82061
-- VERSION IS QUCATALOG OF ACADEMIC FROM LIBRARY QULIB (ID=QUCDCS2)
-- ACCESS $RUN$ ON INPUT FOR CATALOG QUCATALOG
-- ACCESS $ADD$ ON OUTPUT FOR CATALOG QUCATALOG
-- PERFORM SESS1
ENTER ACCESS KEY FOR AREA TESTCURRIC
>>> $XX99$ FOR AREA TESTCURRIC
>>> *END
        *END
STUDENT-ID COURSE-ID
100-22-5860 CHM103
100-22-5860 CHM005
100-22-5860 MATH10
122-13-6704 HIS103
122-13-6704 PSY136
```

Figure 4-20. Examples of Using a Catalog File

Example 2 shows using the catalog file. The VERSION directive specifies the catalog file QUCATALOG. The catalog file has controlled access (defined in the schema); therefore, the ACCESS directive must be specified. Two access keys are specified: RUN for retrieval (INPUT) and ADD for update (OUTPUT). When SESS1 is performed, the user is required to enter access keys for the area TESTCURRIC. After the access keys are successfully entered, STUDENT-ID and COURSE-ID are displayed from the area.

Once a CDCS-catalog file has been created and the VERSION directive has established processing CDCS catalog mode, the use of the catalog is as described in the Query Update reference manual.

RECORD LOCKING MECHANISM

Query Update allows update to occur in only one area at a time. Within an area, only one record (the record being updated) is locked at a time. Query Update automatically makes all the requests to CDCS for the locking and unlocking of records. When Query Update requests a record lock, CDCS holds a protected record lock on the record for the Query Update program; therefore, no other user can update the record, but other users can read the record. CDCS releases the record lock as soon as the record is updated.

A Query Update program cannot be in a deadlock situation because the program can have only one record lock at a time.

The Query Update user can be in the situation of attempting to access an area or record that is locked by another application program. If Query Update finds a locked area or record during an interactive session, Query Update issues a message and gives the user the opportunity to either terminate the access request or to specify that Query Update is to reattempt the access request. During batch processing using Query Update, the application is queued until the locked area or record becomes available.

ERROR PROCESSING

Query Update provides space for the data base status block, which is a block of memory to which CDCS returns status and error messages. When CDCS returns an error code to the data base status block, Query Update issues one of the following messages:

```
904 CRM/CDCS ERROR o FILE a FUNCTION a

908 CDCS ERROR o FILE a FUNCTION a ITEM d
```

Through these messages, Query Update informs the user of the octal value of a CRM or CDCS error code, the area being processed, the function being performed when the error occurred, and the decimal value of the sub-schema item ordinal for item-level errors. If an error message is returned, the interactive user is issued one of the above diagnostics and Query Update prompts the user for subsequent action; the batch job is aborted.

Query Update does not issue a message when a null record or control break condition occurs during relation processing.

Refer to section 5 for additional information about CDCS error processing.

USER-ID REGISTER

Query Update maintains a USER-ID register that contains the name by which CDCS knows a Query Update program. To establish a name for a Query Update program, Query Update initializes the USER-ID register to the name QU, which is left-justified and blank-filled in the 10-character register. Query Update communicates the contents of the USER-ID register to CDCS when executing an INVOKE, CREATE, or VERSION directive.

If a user wants a program known by a name other than QU, the user can update the contents of the USER-ID register by issuing a MOVE, EXECUTE USING, or MODIFY directive. For example, the following directive establishes the name TITLE1 for the Query Update program:

```
MOVE $TITLE1$ TO USER-ID
```

When specifying a program name, the user must modify the USER-ID register before specifying the first INVOKE, CREATE, or VERSION directive that establishes processing with CDCS.

RECOMPILATION GUIDELINES

The checksum facility is the mechanism used in determining whether the changes made to the schema require the recompilation of a subschema. The DDL compiler produces a one-word identifying bit string (called a checksum) for each realm and area in the schema. A checksum is also generated for each subschema. The checksums are stored in the schema and subschema directories and are recorded in the master directory. The checksums are used to verify the consistency of the schema and its associated subschemas, as well as the consistency of the subschema and the application referencing it.

When the schema is recompiled due to a modification of a data base realm or relation, the data administrator compares the checksums generated with the corresponding checksums in the previous version of the schema. If the checksums do not match, all subschemas that reference the realms or relations whose checksums have changed are recompiled.

If a subschema has been recompiled and the checksum of the recompiled subschema differs from its previous checksum, a Query Update application can no longer use the previous subschema. If an application references an invalid checksum, CDCS aborts the application and issues a diagnostic. The Query Update user can prevent the abnormal termination of an application by ensuring, prior to executing the program, that the subschema checksum in the master directory matches the checksum of the subschema used in the Query Update application.

Data base files (realms or areas) can be created and accessed through an application program written in COBOL, FORTRAN, or Query Update. The application references data base files and relations with record descriptions that are contained in a separately compiled schema and subschema defined by the data administrator.

The data administrator can impose controls on the update operations allowed on two logically associated files or two data items within the same file. These controls are imposed by the data administrator in the schema. The controls are called constraints.

This section documents features of CDCS and the processing considerations involved in using these features. These features can affect the way a COBOL or FORTRAN application program is coded.

Query Update automatically handles the use of many of these features; therefore, with the exception of the discussion about constraints, the discussions in the following subsections do not apply to the Query Update user. The Query Update user should, however, consider contraints when making updates.

EXECUTION TIME PROCESSING

Before an application program executes, the data administrator must take the steps necessary to bring CDCS to a system control point (unless the CDCS Batch Test Facility described in appendix G is used). When a compiled COBOL or FORTRAN application program is executed, CDCS monitors and interprets all requests for action on relations and on data base files. When an application program invokes CDCS, CDCS attaches all data base files and index files that are both included in the version referenced by the program and referenced either directly or indirectly in the subschema used by the program. A file is referenced directly in the subschema when the realm name occurs in the subschema listing (refer to the Subschema subsection in sections 2, 3, and 4). A file is referenced indirectly when it is associated in a constraint with a realm included in the subschema.

An independent (non-data-base) file can be used by an application program during CDCS processing; however, CDCS does not process non-data-base files. Any non-data-base files used by the program must be attached before CDCS processing begins.

When the data administrator has specified alternate data base versions, CDCS assumes files of version MASTER are to be accessed by an application program if no alternate version is referenced. For a COBOL program only, the files corresponding to the areas in version MASTER are always attached when CDCS is invoked. This occurs even if the program explicitly uses only alternate versions. The data administrator is responsible for insuring that the files of version MASTER exist and are available to CDCS for execution of a COBOL program.

If one or more of the files referenced in the subschema do not exist when an application program is executed, a program executing in batch mode is aborted; but a program executing through TAF is terminated from CDCS. Control is returned to TAF.

ERROR PROCESSING

CDCS diagnostics indicate errors and informative conditions for all phases of CDCS processing. CDCS reports diagnostics to both the user control point and to the CDCS system control point. When CDCS is executing as the CDCS Batch Test Facility, the user control point and CDCS control point are the same; therefore, all CDCS diagnostics go to the user control point in that situation.

When CRM errors occur during processing, CDCS reports the error through a CDCS diagnostic and takes action, depending on the CDCS classification of the CRM error. When a CRM fatal error occurs on a file (area or realm) during execution, CDCS does not allow the application program to access that file for the remainder of the time the program executes.

An application programmer using CDCS can obtain error and status information from the user error file, from one of several output files produced during CDCS processing, or from within the application program itself. Refer to the Status Checking subsection of sections 2 and 3 or the Error Processing subsection of section 4 for information about obtaining error diagnostics within the application program. Appendix B provides a list of CDCS diagnostics with accompanying explanations. The listing and explanation of CRM diagnostics is in the CRM Advanced Access Methods reference manual.

CDCS DIAGNOSTICS

When a CDCS error or particular condition occurs during the execution of an application program, CDCS writes a diagnostic message. CDCS writes the messages to the user error file ZZZZZEG for application programs executing in batch mode and to the CDCS output file at the system control point. Fatal and nonfatal errors and informative conditions are reported in this manner; trivial errors are written only to the file ZZZZZEG. Two informative CDCS codes (627 octal 632 octal) indicate file status on a relation read but are not written to the file ZZZZZEG or to the CDCS output file. (A COBOL or FORTRAN application program can obtain information about these condition through the data base status block described in sections 2 and 3.) A number of other messages reporting fatal and nonfatal errors and informative conditions are sent to the user control point dayfile and to the CDCS output file. Additionally, fatal, nonfatal, and informative messages concerning the CDCS system control point are reported in the system control point dayfile.

Fatal CDCS errors cause CDCS to abort a program executing in batch mode unless the program has enabled the immediate return feature (for information about this feature, refer to the Immediate Return Feature subsection later in this section). Fatal errors that occur when a COBOL or FORTRAN program is executing through the Transaction Facility (TAF) cause CDCS to terminate the program from CDCS; CDCS returns control to TAF.

Refer to appendix B for more information about CDCS diagnostics.

USER ERROR FILE

The user error file ZZZZZEG contains errors that pertain to user program processing. The ZZZZZEG file is not printed as part of the output from a user's job. Refer to the CRM Advanced Access Methods reference manual for details on how to print this file using the CRMEP utility.

The user error file ZZZZZEG is not available for COBOL or FORTRAN tasks executing through TAF.

RELATIONS DEFINED

The relation processing component of CDCS allows users to retrieve data from two or more files (areas or realms) joined together logically. The logical structure created by the joining of several files is termed a relation. CDCS controls the manipulation of the files joined in a relation and controls the selection of record occurrences through the use of information from the schema and subschema directories.

When the data administrator designs and creates a data base, it usually contains several files with common data item fields. These common data item fields are defined in the schema as join terms or identifiers. Join terms or identifiers can be data item fields that contain primary keys, alternate keys, or data items that are not keys. The join terms or identifiers are used to join the areas in a relation. During processing, the join terms and areas are associated with data items and data base files respectively.

The application program can obtain information from several files involved in a relation by referring to the common data item. For example, the EMPLOYEES file and the PRODUCTS file of a data base are related by a common data item, PROJECT-NO. A PRODUCT record type and an EMPLOYEE record type are linked by PROJECT-NO, an alternate key in the EMPLOYEES file.

The relation can be read by a single relation read request. The PRODUCT record is read using the primary key PRODUCT-NO, and the EMPLOYEE record is accessed by alternate key using the value of PROJECT-NO obtained from the PRODUCT record. The application programmer can then use PROJECT-NO as a key to retrieve all the employee record occurrences for that project number. This relationship between project number in one record and project number in another record results in the identification of a number of employees. The diagram of this relationship in figure 5-1 shows it as a one-to-many relationship.

Without the CDCS relation processing component, the application programmer would have to write the procedures to read the PRODUCT record initially and then, using the alternate key value obtained from the PRODUCT record occurrence, read the EMPLOYEE records.

The programming becomes more difficult when three records are participating in a relationship such as the one just described. For example, if a CONTRACT record were introduced into the above relationship and many records were retrieved for each alternate key value, the programming to handle this relationship would be more complicated than for the relationship joining only two files.

A relation joining three files is illustrated in figure 5-2. The defined structure of the relation begins with a contract number. The CONTRACTS file is accessed first. CONTRACT-NO from the CONTRACT record occurrence is used to access the PRODUCTS file via its alternate key CONTRACT-NO. A number of products exist for a given contract; that is, there are duplicate values in the contract number field.

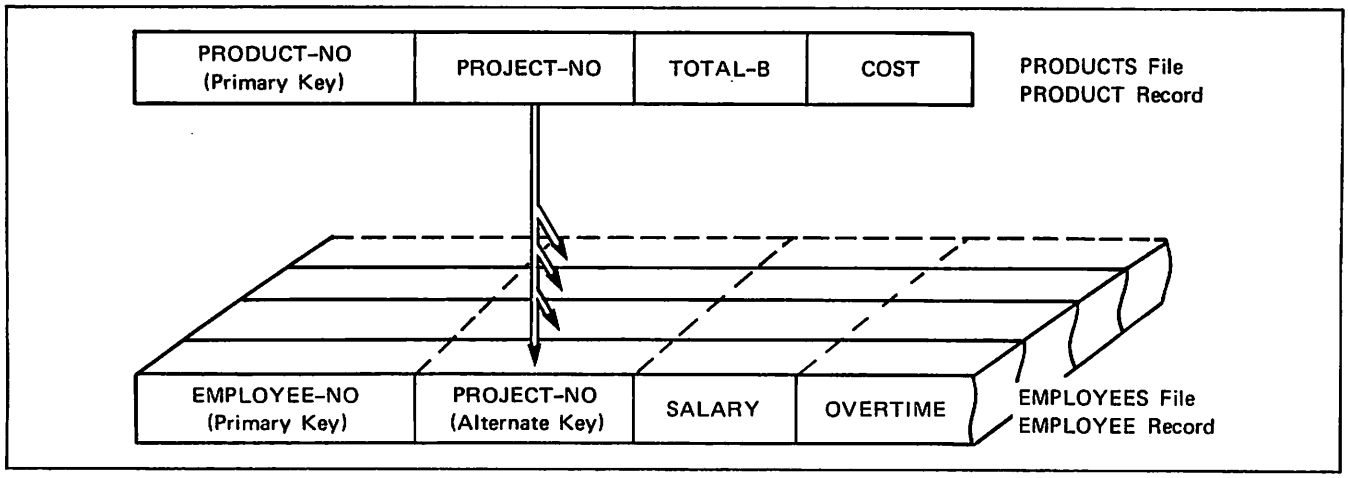


Figure 5-1. Two-File Relationship Example

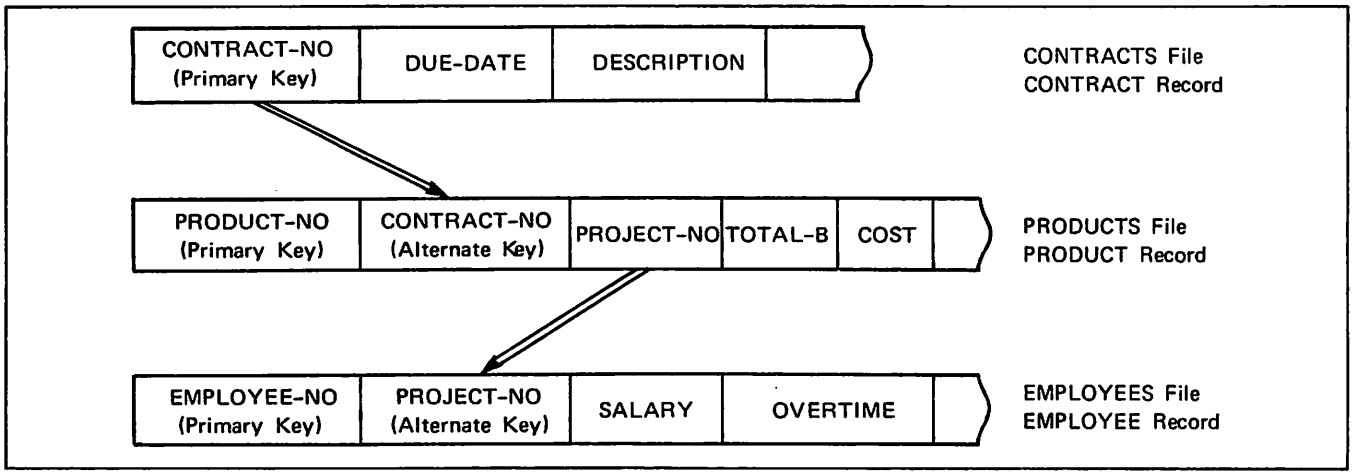


Figure 5-2. Three-File Relationship Example

Each PRODUCT record occurrence retrieved for a given contract supplies the project number value that can in turn be used to retrieve a number of EMPLOYEE record occurrences. PROJECT-NO is an alternate key in the EMPLOYEE record and, again, duplicate values exist.

HIERARCHICAL TREE STRUCTURE

The dependency of the record occurrences within the files joined by a directed relationship can be schematically represented as a hierarchical tree structure like the one shown in figure 5-3. The root of the tree contains the record occurrences of the CONTRACTS file. The CONTRACT record occurrence branches to record occurrences in the PRODUCTS file. Likewise, the PRODUCT record occurrences branch to record occurrences in the EMPLOYEE file. Extraction of record occurrences from each of the three files in the relationship is performed automatically for the application program through CDCS if a relation joining the CONTRACTS, PRODUCTS, and EMPLOYEES files has been defined in the schema and included in the subschema used by the application program.

Ranks of a Relation

A relation entry in the schema links the files together in the relation in a defined order and assigns the files a rank in the relation. The order is determined by the order in which the files are included in the relation in the schema. The first file included in the relation is assigned rank 1; the second rank 2, and so on with subsequent files being assigned a rank equal to the rank of the previous file incremented by 1.

The ranks of a relation can be determined from the relation statistics portion of the subschema listing. The relation name together with each realm joined in the relation are listed according to rank. For example, the files of relation CONTRACTS-PRODUCTS-EMPLOYEES are listed as follows:

```
CONTRACTS-PRODUCTS-EMPLOYEES JOINS
    AREA - CONTRACTS
    AREA - PRODUCTS
    AREA - EMPLOYEES
```

The files are assigned ranks in the relation as follows: CONTRACTS, rank 1; PRODUCTS, rank 2; and EMPLOYEES, rank 3.

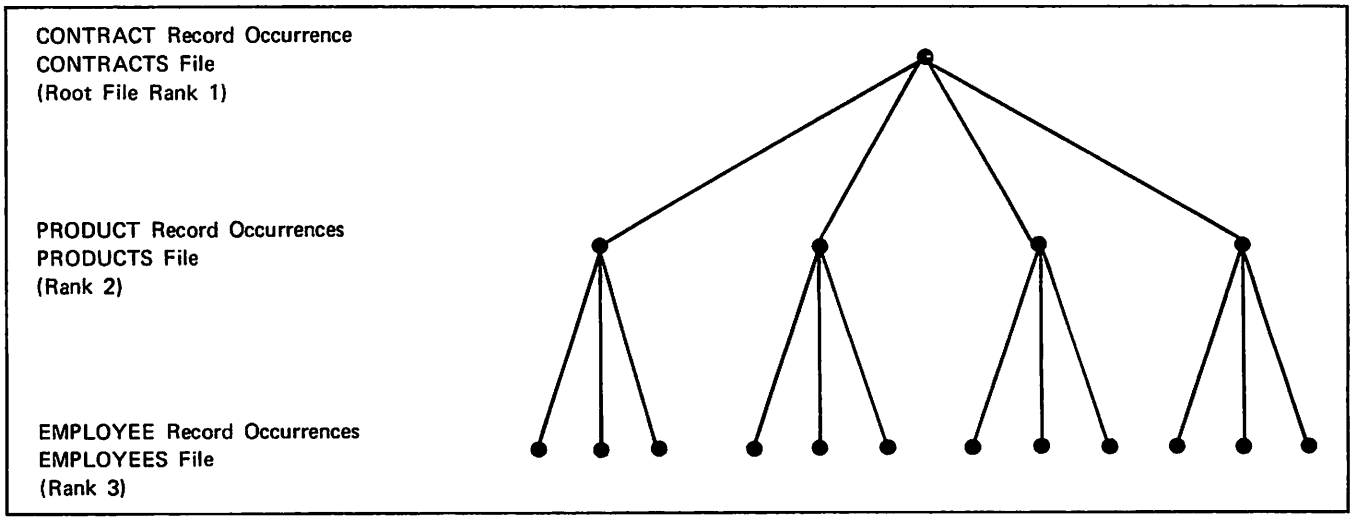


Figure 5-3. Tree Structure for CONTRACTS-PRODUCTS-EMPLOYEES Relationship

The lower rank a file has in a relation, the higher position the file has in the hierarchical tree structure. For example, the file of lowest rank (the root file, which is assigned rank 1) is pictured at the top of the tree structure.

Parent/Child Relationship

The joining of files in a directed relationship results in a dependency condition between record occurrences linked in the files. A record occurrence in the root file is termed the parent record occurrence for all related record occurrences (each termed a child record occurrence) in the file linked to it. A child record occurrence can also be a parent record occurrence when a subsequent file is joined in the relation and related record occurrences exist in that subsequent file.

In the CONTRACTS-PRODUCTS-EMPLOYEES relation, a record occurrence in the CONTRACTS file is the parent record occurrence for related record occurrences in the PRODUCTS file. Likewise, a record occurrence in the PRODUCTS file is a parent of several record occurrences in the EMPLOYEES file. The record occurrences in the PRODUCTS files are the children of a record occurrence in the CONTRACTS file. Likewise, the record occurrences in the EMPLOYEES file are children of a record occurrence in the PRODUCTS file.

A parent record occurrence is one that has another record occurrence at the next numerically higher rank in the relation. A child record occurrence is one that has another record occurrence at a numerically lower rank in the relation.

RECORD QUALIFICATION

Record qualification is the method used to restrict which records in a relation are to be returned to the user. Record qualification is implemented by specifying criteria that must be satisfied by a record occurrence. Qualification is specified in the subschema for records in any file for the relationship. Use of qualification can greatly limit the number of record occurrences returned to the user's work area. For a better understanding of this statement, the CONTRACTS-PRODUCTS-EMPLOYEES relationship is reexamined.

A tree structure of record occurrences in the CONTRACTS-PRODUCTS-EMPLOYEES relation is illustrated in figure 5-4. A contract C1 is composed of four products: P1, P2, P3, and P4. Each product is developed by a number of employees, and each employee works on only one product; for example, employees E1 and E2 develop product P1; employees E3, E4, E5, and E6 develop product P2. Proceeding from left to right following each path up the tree structure, 12 occurrences of the relationship can be identified, namely C1P1E1, C1P1E2, C1P2E3, C1P2E4, and so on.

Qualification criteria are specified in the RESTRICT clause in the COBOL and Query Update subschemas and in the RESTRICT statement in the FORTRAN subschema; the criteria are used by CDCS to determine whether a record occurrence qualifies to be returned to the user's work area as part of the relation occurrence. For example, to retrieve the records of those employees working on product P4, qualification can be specified for the PRODUCT record type to indicate that the record occurrences should be restricted to those in which the value of the product number field is P4. CDCS reads and discards record occurrences P1, P2, and P3 before retrieving record occurrence P4 and its child record occurrences. Since P1, P2, and P3 do not qualify, no input/output operations are performed to retrieve their child record occurrences.

Without the facility to qualify records, each of the record occurrences to the left of the required ones (including P1, P2, and P3 and all their child record occurrences) would have to be extracted and returned to the user's work area (in many cases, after CDCS record mapping is performed). The user would then have to determine if the record occurrence was the one required or not.

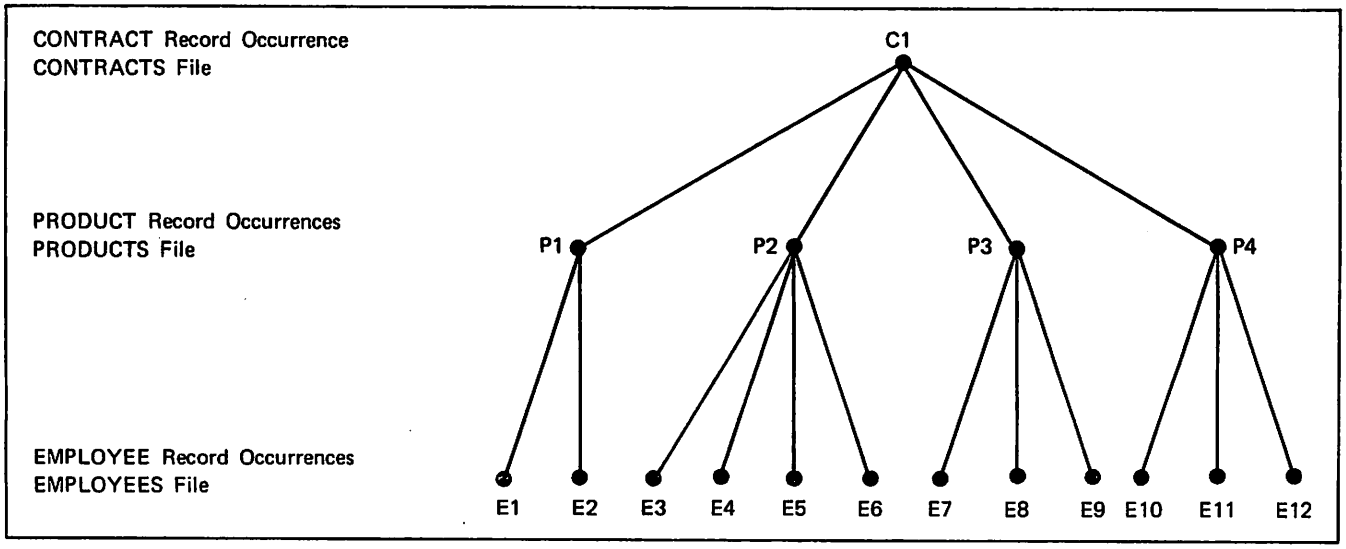


Figure 5-4. Complex Tree Structure for CONTRACTS-PRODUCTS-EMPLOYEES Relationship

Record qualification in the COBOL and Query Update subschemas is defined in the Relation Division. This division contains the relation name and the RESTRICT clauses (which specify record qualification).

Record qualification in the FORTRAN subschema is defined in a relation definition. A relation definition contains the RELATION statement which specifies the relation name) and the RESTRICT statements (which specify record qualification.

CDCS RELATION PROCESSING

A relation defines a directed path joining several files. Join terms, called identifiers, are data items that link the files together. The identifiers are established in the schema. The identifiers must be inspected by CDCS to traverse the relation and return a record occurrence from each file in the relation to the user's work area.

File positions affect CDCS sequential relation processing. The user should exercise caution in performing input/output operations that might alter positions on the files joined in the relation while executing within a sequential read loop. The user can perform input/output operations on a file in a relation if the file is repositioned before continuing the relation read.

Special formats of the OPEN, CLOSE, READ, and START statements allow COBOL or FORTRAN application programs to utilize relation processing while performing input/output operations on the data base. The statements described in the following subsections are those used in COBOL and FORTRAN application programs.

OPENING A RELATION

Application programs using CDCS can open all files joined in a relation with a single relation OPEN statement. The relation OPEN statement is performed as if a separate open was executed for each file, in the order of the rank of the files in the relation. When files in a relation are opened by the relation OPEN statement, they should not be opened by separate OPEN statements.

A relation is normally opened for input processing only. A relation is opened for input/output processing when the user wants to have CDCS lock all records read for the relation occurrence or when the application program is to perform update operations on individual files in the relation.

POSITIONING A RELATION

The root file of a relation is positioned for subsequent read processing by the relation START statement. When a relation START statement executes, the root file of the relation is positioned at the first record occurrence that has a key satisfying the condition specified in the KEY phrase of the relation START statement. The key specified is established as the key of reference. The key of reference is always a key in the root file. After the root file has been positioned and the key of reference has been established, a sequential read of the relation retrieves from the root file the record occurrence with the key that satisfies the condition specified by the start. A sequential relation read loop following a relation start references the root file according to the key established by the start.

The example in figure 5-5 shows two relations, R1 and R2, which have a common file, FILEB. Relation R1 joins FILEA and FILEB, and R2 joins FILEB and FILEC. The application programmer can use the relation START statement with a key value specified to position FILEB in relation R1. FILEB is the root file for relation R2. Relation R2 can then be read using that key value. The key value can be saved to reposition FILEB, if necessary, before performing another read of relation R1. This processing cannot be performed if the key being saved has duplicate values or if the key does not appear in the subschema.

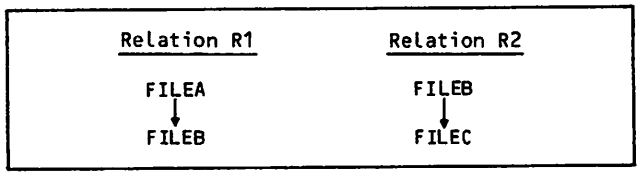


Figure 5-5. File Positioning Example

READING A RELATION

The application program can use the relation READ statement to read a relation. The relation read can be used either for random access of the root file of a relation or for sequential access of the files involved in a relation.

The order in which other files in the relation are accessed depends on the identifier specified for the files. If the identifier is an alternate key for a file containing child record occurrences, that file is read randomly by alternate key.

Reading a Relation Randomly

If a key is specified in the relation READ statement, the root file is accessed according to the primary or alternate key of that file. The record occurrence in the root file that contains the specified key is returned to the user's work area along with a record occurrence from each file at a higher rank in the relation. The root file of a relation can be positioned by a random relation read for subsequent sequential relation reading.

If a key is specified for a file in a relation other than the root file, the COBOL compiler or the DML FORTRAN preprocessor issues a diagnostic.

Reading a Relation Sequentially

If no key is specified in the relation READ statement, the root file is read sequentially by the current key of reference. A relation can be positioned for subsequent sequential reading by either a relation START statement or a relation READ statement that specifies the KEY phrase. Once the relation has been positioned, successive sequential reads of the relation are specified by the user with the READ NEXT AT END statement in COBOL or the READ statement without the KEY option in FORTRAN; these are translated to retrievals on the appropriate files in the relation. Within the sequential read loop, the user should not try to reposition the files involved.

The diagram in figure 5-6 illustrates a group of record occurrences for a relation having three files. If the relation is read sequentially, the order in which record occurrences are returned to the user is A, B, and C first, providing this set of record occurrences meets qualification criteria. Successive sequential reads of this relation by CDCS would return the remaining record occurrences that are child record occurrences of B (namely, record occurrences D and E). FILEA and FILEB are not read again since CDCS expects the user's work area to still contain record occurrences A and B. To retrieve the next record occurrence, CDCS returns to FILEB in the relation and retrieves record occurrence F, if F meets qualification criteria, and in turn retrieves the children of record occurrence F (record occurrences G and H).

The diagram in figure 5-7 identifies the record occurrences that are contained in the user's working storage area after each read of the relation, assuming the record occurrences shown meet qualification criteria. The user's work area contains record occurrences with record descriptions that are defined in the subschema used by the application program.

Reading Relations in Parallel

An application program can read more than one relation in parallel, provided no common files exist between the relations being read. For example, figure 5-8 illustrates three relations, R1, R2, and R3, which have files joined as shown. FILEC is a common file in relations R1 and R2. If R1 and R2 are read in parallel, the results can be unpredictable. R1 and R3, however, or R2 and R3 can be read in parallel without causing any undesirable results.

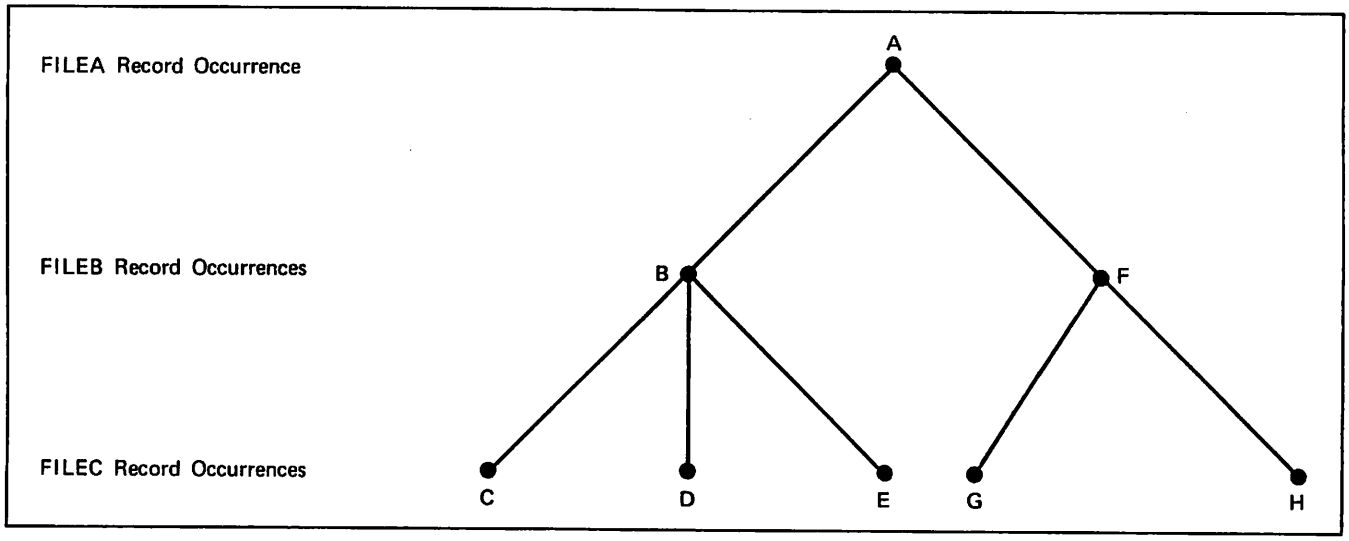


Figure 5-6. Record Occurrences for Three Related Files

User's Work Areas

A	B	C	First read
A	B	D	
A	B	E	Successive sequential reads
A	F	G	
A	F	H	

Figure 5-7. Record Occurrences in User's Work Areas After Reading

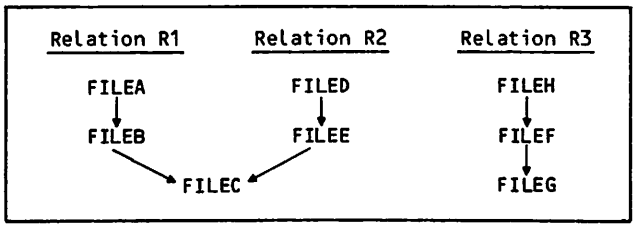


Figure 5-8. Three Relation Example

Reading a Relation Defined With Record Qualifications

Record qualification criteria are specified by the RESTRICT clause in the COBOL subschema or the RESTRICT clause in the FORTRAN subschema. If a data item specified in the RESTRICT clause is not included in the subschema, the data item must be defined in the Data Division of the COBOL application program or defined as variable in the FORTRAN application program before any relation read is performed. The data item must be defined with the same size and type (usage for COBOL) as the subschema item used in the comparison.

CDCS obtains the value of the specified data item from the program when the following operations are performed:

On the first sequential read of the relation after execution of an OPEN relation statement

On a random read of the relation

On the first read following execution of a START relation statement

When a sequential read of the relation follows a random read, CDCS does not change the value of the data item.

Reading a Relation When Data Base Versions Exist

A relation can join different groups of permanent files when data base versions exist. The files joined in a relation depend on the files associated with the version being used. Some files are used by several versions while other files are used by only one version. This means that relational reads can yield different results depending on the version used.

For example, figure 5-9 illustrates the CONTRACTS-PRODUCTS-EMPLOYEES relation being used with two data base versions, MASTER and UNIT1. Contrasted with previous examples of this three-level relation, the names CONTRACTS, PRODUCTS, and EMPLOYEES now refer only to the areas defined in the schema and not to the permanent files; the areas are associated with permanent files with different names.

In the example, versions MASTER and UNIT1 share two files: CMSTR (associated with the area CONTRACTS) and EMSTR (associated with the area EMPLOYEES). Each version uses a separate file associated with the area PRODUCTS.

A relational read using version MASTER could result in the following record occurrences being returned:

Record 1 of permanent file CMSTR
Record 1 of permanent file PMSTR
Record 1 of permanent file EMSTR

The same relational read using version UNIT1 could result in the following record occurrences being returned:

Record 1 of permanent file CMSTR
Record 1 of permanent file PU1
Record 3 of permanent file EMSTR

There are no restrictions on the use of relations with data base versions; however, the possibility of retrieving different records in relational reads that use different data base versions should be recognized.

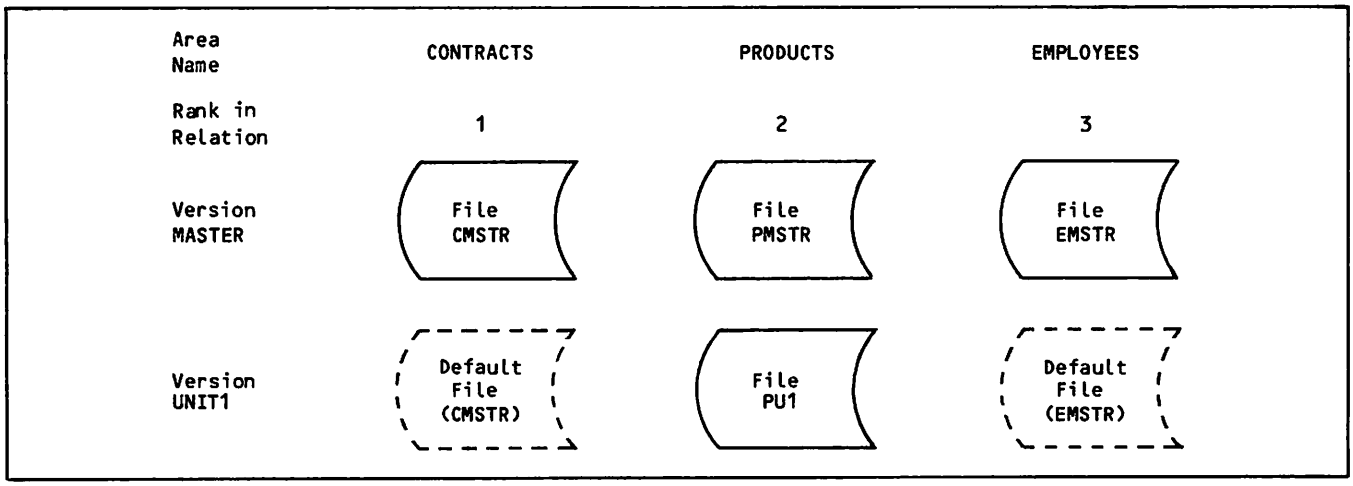


Figure 5-9. Example of Files Joined by a Relation and Grouped by Version

UPDATING FILES JOINED IN A RELATION

CDCS relation processing involves only the retrieval of relation occurrences. No monitoring of delete or update operations occurs on files involved in a relation unless a constraint has been indicated in the schema for the files in the relation. Constraint processing by CDCS occurs independent of relation processing. For details concerning constraint processing, refer to the Constraint subsection later in this section.

The user must take precautions when performing update operations to protect the integrity of the data base. Inadvertent modification of join identifier values can change parent-child relationships. Deletion of parent record occurrences can make all child record occurrences of a deleted parent record occurrence inaccessible when a relation is read.

The integrity of the data base can be maintained in update operations if the user is aware of the following points:

Relations should be designed by the data administrator with the least number of connnections between the relations. When one file is linked to a file in one relation and is also joined to another file in another relation, the update operations on the common file must be monitored in order to preserve the meaning of each of the relations.

The user should delete child record occurrences before deleting the parent record occurrence from a file in a relation.

A parent record occurrence should be written before any child record occurrences are written on a file participating in a relation.

A relation read operation positions all files in the relation hierarchy. Consequently, if an update operation, such as a delete or a rewrite, is performed on a file in the relation after the relation is read, the record occurrence that was just read for the relation occurrence is deleted or rewritten. All child record occurrences of a record occurrence that has been deleted are no longer accessible through a read of the relationship. When a rewrite operation is performed on a record occurrence, it can change the tree structure for the data in the relation if the values of the identifiers (join terms) are changed.

CLOSING A RELATION

All files joined in a relation can be closed with a single relation CLOSE statement. The relation CLOSE statement is performed as if a separate CLOSE was executed for each file in the relation. When files in a relation are closed by the relation CLOSE statement, they should not be closed by separate CLOSE statements.

INFORMATIVE CONDITIONS

During relation processing, CDCS detects the occurrence of the following conditions:

Null record occurrence on a file

Control break on a file

A FORTRAN or COBOL application program can check for these conditions and determine the lowest ranked file on which the condition occurs if appropriate receiving fields are defined and used in the application program. Refer to the Status Checking subsections of sections 2 and 3 for more information.

Null Record Occurrence

If a parent record occurrence does not qualify for retrieval, any child record occurrences automatically do not qualify. In the example shown in figure 5-6, if B does not qualify, record occurrences C, D, and E automatically do not qualify; none of these record occurrences would be returned to the user's work area for FILEB and FILEC. If, however, B does qualify, but C, D, and E do not, one null record occurrence is returned to the user's work area for FILEC to indicate that no child record occurrences of B qualify. The null record occurrence consists of a display code right bracket (]) in each character position; the number of character positions filled depends on the subschema description of the record type.

In general, a null child record occurrence status is returned to the user if all the children of a parent record occurrence that qualified do not qualify, or if no child record occurrences exist. Some examples of null record occurrences returned to the user's work area are illustrated in figure 5-10.

If null record occurrences are returned for all files in a relation except the root realm, another READ statement must be executed to obtain the next set of record occurrences for the relation.

Control Break

The control break condition on a file signifies that a new record occurrence was read for the parent file in the relation. Control break status, however, is returned for the realm of the child. If a file in a relation has control break status after a sequential READ statement has been issued for the relation, the record occurrence read for this file is a child record occurrence for a new parent record occurrence.

Control break status is not set for a file if a null record occurrence status must be set.

Example of Null Record and Control Break Conditions

Figure 5-11 shows the record occurrences A through M in files FILE1, FILE2, and FILE3 and the control break and null occurrence conditions that result from retrieval of each relation occurrence. No qualification criteria have been specified on any of the records for the files.

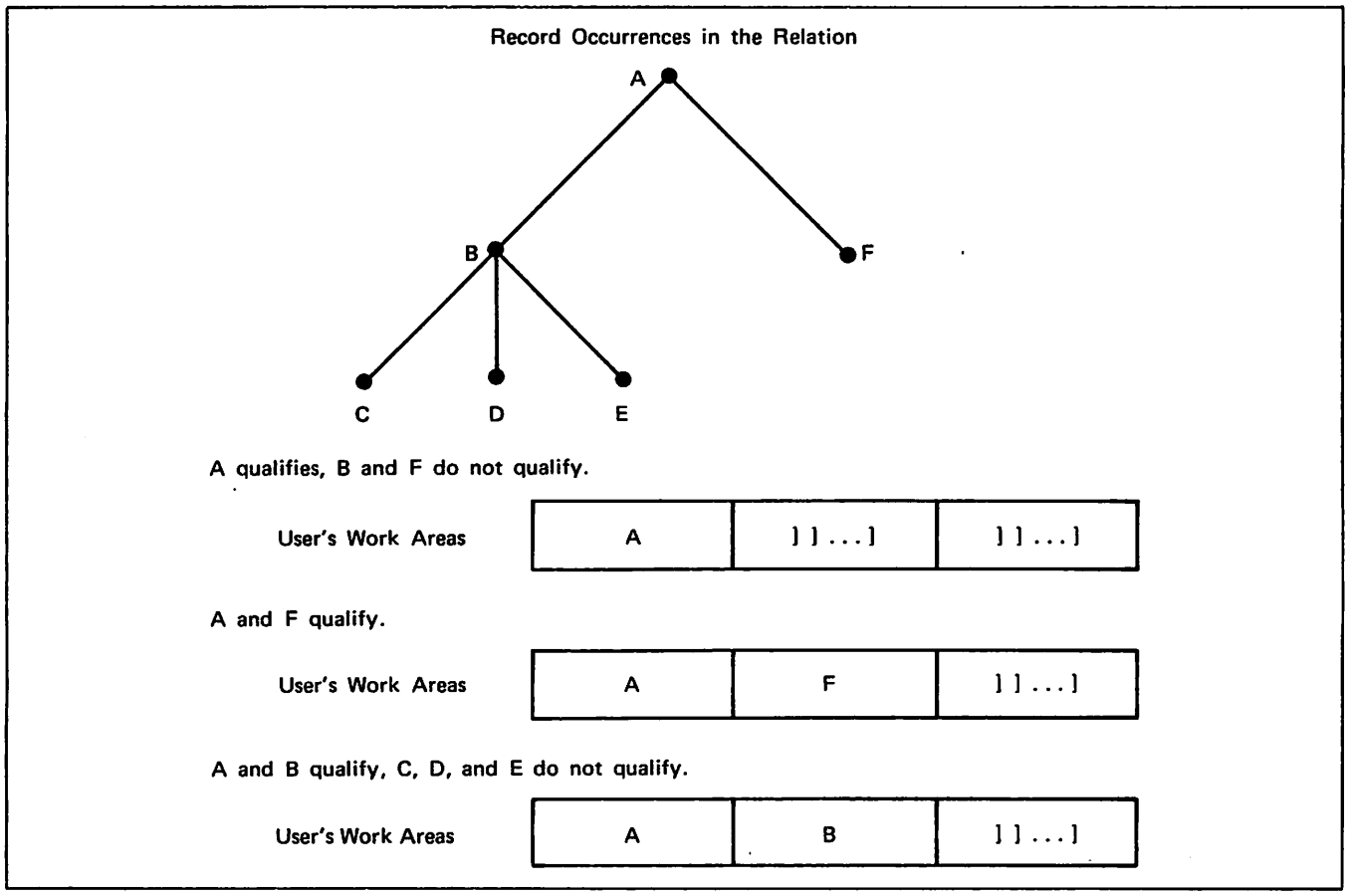
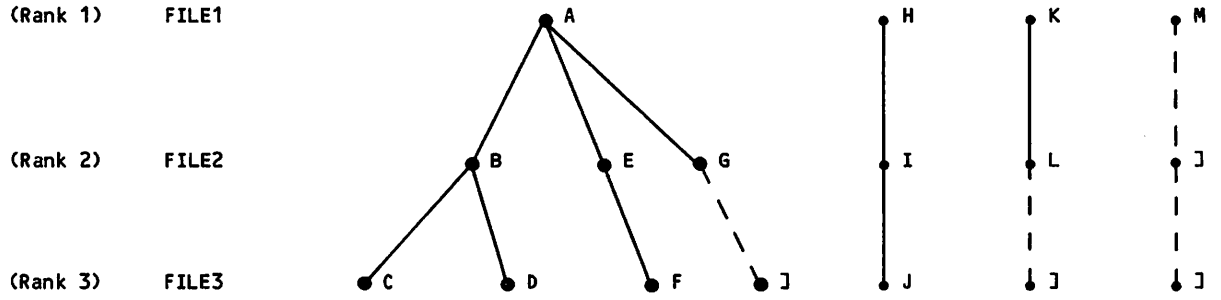


Figure 5-10. Null Record Occurrence Examples



Data Base Status Block

Relation Occurrences			Null Occurrence Rank	Control Break Rank	Condition - File
A	B	C	0	2	Control break - FILE2 Control break - FILE3
A	B	D	0	0	
A	E	F	0	3	Control break - FILE3
A	G	null	3	0	Null occurrence - FILE3
H	I	J	0	2	Control break - FILE3 Control break - FILE2
K	L	null	3	2	Null occurrence - FILE3 Control break - FILE2
M	null	null	2	0	Null occurrence - FILE2

Figure 5-11. Example of Null Occurrence and Control Break Conditions

TRANSACTION PROCESSING

Transaction processing provides for the grouping of updates by FORTRAN and COBOL application programs. Updates include the operations of writing, rewriting, and deleting records. A group of updates for which the application program specifies the beginning and the completion is referred to as a transaction. All updates within a transaction are temporary. The updates within a transaction are made permanent when the application program specifies the completion of the transaction (called committing a transaction).

If an application program does not commit a transaction, each record that was updated within the transaction is restored to the state it was in just before the beginning of the transaction and CDCS issues an informative diagnostic. There are several typical situations in which transactions are not committed. Program logic can determine that the transaction should not be committed and can cancel (drop) the transaction. System failure, program failure, or deadlock can occur during the application program's processing of the transaction. In each of these situations, updates made within the uncommitted transaction are reversed.

If an application program terminates normally but the commit statement was not included in the program logic, CDCS reverses the updates made within the transaction and issues a diagnostic message.

When an application program begins a transaction, CDCS processes subsequent update operations by that program in transaction mode. The exclusive record locking mechanism prevents other users from accessing records updated within an uncommitted transaction.

The restart component of transaction processing allows an application program to determine the point at which to begin processing following a system failure. An application program can determine whether or not a transaction was committed before the system failed. With this information available, the program can determine the point at which to resume transaction processing.

The application program can perform transaction processing only if transaction processing is in effect for the schema. The application program can perform a restart operation only if both transaction processing and the restart mechanism are in effect for the schema. The data administrator, by selecting logging options in the master directory, establishes the facilities that support tansaction processing and the restart mechanism.

PROCESSING OPERATIONS

COBOL routines and FORTRAN Data Manipulation Language (DML) statements provide for the operations involved in transaction processing. A total of five operations can be performed. These operations are described in the following paragraphs. The first three operations are directly involved in main-line code dealing with updates. The last two operations are used for operation restart following system failure.

The operations that an application program performs in transaction processing are as follows:

Begin a transaction

> Designates the beginning of a transaction and communicates a transaction identifier to CDCS. This causes CDCS to begin processing in transaction mode for the application program.

Commit a transaction

> Designates the end of a transaction and indicates that the updates within the transaction are to be committed. This causes all the updates made within the transaction to be made permanent.

Drop a transaction

> Designates the end of a transaction and indicates that the updates already made within the transaction are to be canceled. This causes each record updated within the transaction to be restored to the state it was in just before the beginning of the transaction.

Obtain a restart identifier

> Communicates with CDCS to obtain a restart identifier for the application program. The application program must save the restart identifier for subsequent use in a restart operation. If program restart capability is desired, this operation must be performed before the first transaction is begun.

Inquire about the status of the last transaction

> Communicates to CDCS a restart identifier and obtains from CDCS the transaction identifier for the last completed transaction associated with that restart identifier. CDCS then assigns the restart identifier obtained to the program. This operation provides for restarting an application program. Application program logic then uses the transaction identifier to determine with which transaction to resume processing.

Table 5-1 lists the operations together with the corresponding COBOL routines and FORTRAN DML statements. Refer to the descriptions of the routines and statements in sections 2 and 3 for more information.

TABLE 5-1. OPERATIONS OF TRANSACTION PROCESSING

Description	COBOL Routine	FORTRAN DML Statement
Begin	DB$BEG	BEGINTRAN
Commit	DB$CMT	COMMITTRAN
Drop	DB$DROP	DROPTRAN
Obtain a restart identifier	DB$GTID	ASSIGNID
Inquire about status of last transaction	DB$ASK	FINDTRAN

PROCESSING CONSIDERATIONS

The following rules and considerations apply to transaction processing:

Only one transaction unit can be in progress at a time within a particular application program. In other words, there can be no nesting of transaction units.

The data administrator defines the maximum number of concurrent transactions allowed for all user's of the schema. If this number is exceeded, CDCS issues a diagnostic. The application program can retry the transaction request later.

The following FORTRAN DML statements and routines are not allowed within a transaction: ASSIGNID, CLOSE, DMLRPT, INVOKE (except within a chain of TAF tasks), LOCK, NEWVERSION, OPEN, PRIVACY, and UNLOCK.

The following COBOL statements and routines are not allowed to be used within a transaction: CLOSE, C.LOK, C.UNLOK, DB$GTID, DB$LKAR, DB$RPT, DB$VERS, and OPEN.

File locks are not recommended for use with transaction processing.

It is the responsibility of the application programmer to ensure that the restart identifier is saved in a non-data-base environment so that it can be retrieved if a program restart operation is desired.

If a program performs the inquire operation (DB$ASK for COBOL or FINDTRAN for FORTRAN) and CDCS does not know the restart identifier, CDCS returns 10 asterisks, **********, as the restart identifier and issues a nonfatal error. This situation could occur because the application program using the restart identifier terminated normally. If this does occur, program logic should provide for obtaining a new restart identifier before beginning transaction processing.

EXAMPLES OF TRANSACTION PROCESSING

A transaction processing sequence is illustrated in figure 5-12. Application program PGRMA performs a begin transaction operation and read and update operations within the transaction. CDCS performs the logging and record locking operations required to support transaction processing. The locking operations are described in the following subsection. When program PRGMA performs the commit transaction operation, CDCS will make all the updates permanent and will release all the record locks.

Portions of a COBOL program that illustrate transaction processing are shown in figure 5-13. The Special Names paragraph specifies the subschema being used. The DB$GTID routine is used to obtain a restart identifier. The DB$BEG routine designates the beginning of a transaction. Two realms, PROJECT and PRODUCTS, are read and updated within the transaction. If an error occurs during transaction processing, the procedure ERRPROC-1 executes causing a message to be displayed and the transaction to be dropped. If no error occurs during transaction processing, the transaction is committed by the DB$CMT routine.

Refer to section 7 for an example of a FORTRAN program that performs transaction processing.

CDCS PROCESSING IN TRANSACTION MODE

During processing in transaction mode, an application program holds an exclusive lock on a record or multiple records when update operations are performed. During transaction processing, record locks assigned to a program are not released until the application program commits the transaction, drops the transaction, terminates normally or abnormally, encounters a deadlock situation, or encounters a resource conflict. Refer to the CDCS Locking subsection later in this section for more information.

Transaction processing also affects how CDCS handles deadlock situations. Refer to the Deadlock subsection later in this section for more information.

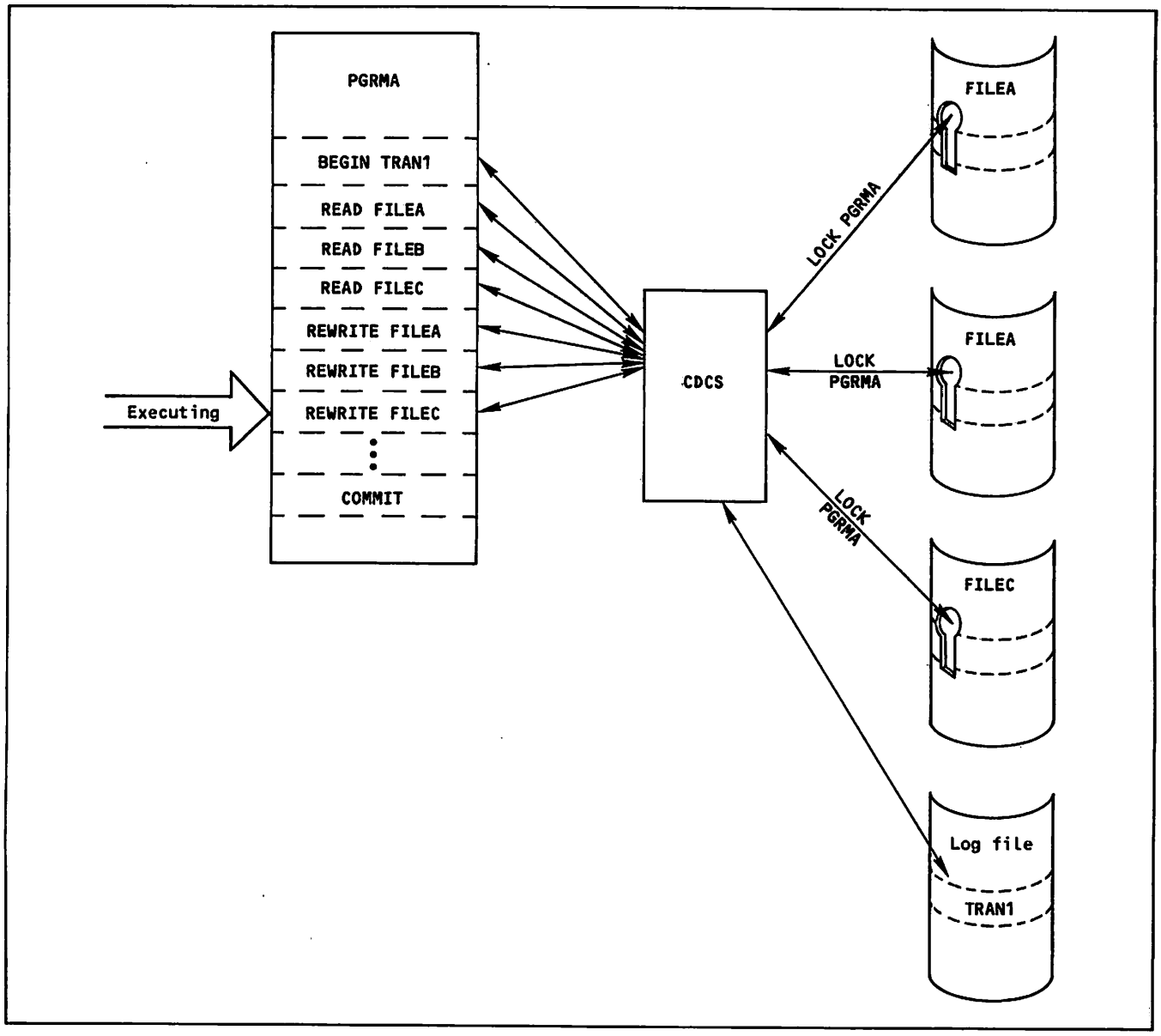


Figure 5-12. Transaction Processing Situation

CONCURRENT ACCESS TO A DATA BASE

CDCS allows concurrent access to a data base. Concurrent access means that two or more programs can access the same file at the same time. Access can be for retrieval or update operations, but care must be taken in concurrent updating. CDCS provides a locking mechanism during concurrent access operations to protect the integrity of the data base. Deadlock can occur when two programs are contending for files or records that have been locked by CDCS or by other programs.

The following paragraphs describe the concepts of locking and deadlock. The word file instead of realm is used in these descriptions; however, in the view of the application program, a file is a realm.

```
IDENTIFICATION DIVISION.
        .
        .
        .
SPECIAL-NAMES.
    SUB-SCHEMA IS C5SS-BUDGET-CHECK.
        .
        .
        .
DATA DIVISION.
FILE SECTION.
FD PRINT-FILE
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS PRINT-LINE.
01  PRINT-LINE                  PIC X(136).
WORKING-STORAGE SECTION.
01  RESTART-ID                  PIC X(10).
01  ACCESS-KEY                  PIC X(30).
01  PROJ-KEY                    PIC X(10).
01  PROD-KEY                    PIC X(10).
01  PRINT-ACCESS                PIC X(30).
01  ERRFLAG                     PIC 9(3)   VALUE ZERO.
        .
        .
        .
PROCEDURE DIVISION.
        .
        .
        .

MAIN-LOGIC-SECTION.
START-UP.
    PERFORM START-UP.
    PERFORM BEGIN-TRANSACTION.
    PERFORM READ-PROJ.
    PERFORM READ-PROD.
    PERFORM UPDATE-PROJ-PROD.
    ENTER "DB$CMT" USING ERR-FLAG.
        .
        .
        .

START-UP SECTION.
OPENING.
    OPEN OUTPUT PRINT-FILE.
    OPEN I-O PROJECT.
    OPEN I-O PRODUCTS.
    PERFORM STATUS-CHECK.
    PERFORM RESTART.

BEGIN-TRANSACTION SECTION.
BEGIN-TRAN.
    ENTER "DB$BEG" USING "MYTRANID1", ERRFLAG.
    IF ERRFLAG NOT EQUAL TO ZERO
        DISPLAY "ERROR BEGINNING TRANSACTION"
        PERFORM FINISHED.

READ-PROJ SECTION.
READ-PROJECT.
    MOVE "M130001560" TO PROJECT-ID IN PROJECT.
    READ PROJECT KEY IS EQUAL TO PROJECT-ID,
        INVALID KEY PERFORM ERRPROC-1.
    PERFORM STATUS-CHECK.

READ-PROD.
READ-PRODUCTS.
    MOVE ZERO TO DEAD-FLAG.
    MOVE "826NAMW019" TO PRODUCT-ID.
    READ PRODUCTS KEY IS EQUAL TO PRODUCT-ID,
        INVALID KEY PERFORM ERRPROC-1.
    PERFORM STATUS-CHECK.
```

Figure 5-13. Transaction Processing Using COBOL (Sheet 1 of 2)

```
UPDATE-PROJ-PROD SECTION.
UPDATE-RECORDS.
    ADD NEW-BUDG TO BUDGET-TOTAL.
    REWRITE PROJREC INVALID KEY PERFORM ERRPROC-1.
    ADD NEW-YTD TO DEV-COST-YTD.
    REWRITE PRODREC INVALID KEY PERFORM ERRPROC-1.

STATUS-CHECK SECTION.
CHECK-STATUS.
    IF DATABASE-STATUS NOT EQUAL TO ZERO
    DISPLAY "ERROR" DATA BASE-STATUS
        "ON" DB-REALM "WHILE PERFORMING" DB-FUNCTION.

RESTART-SECTION.
OBTAIN-RESTART-ID.
    ENTER "DB$GTID" USING RESTART-ID, ERRFLAG.
        IF ERRFLAG NOT EQUAL TO ZERO
        DISPLAY "ERROR OBTAINING RESTART IDENTIFIER"
        PERFORM FINISHED.

ERRPROC-1 SECTION.
ERROR-PROCEDURE.
    DISPLAY "INVALID KEY".
    ENTER "DB$DROP" USING ERRFLAG.
    PERFORM FINISHED.
        .
        .
        .
```

Figure 5-13. Transaction Processing Using COBOL (Sheet 2 of 2)

CDCS LOCKING

To protect the integrity of a data base during concurrent update operations, CDCS performs locking of data base files and records. CDCS holds a lock for an application program. Application programs can either explicitly request a file lock or perform an operation that results in an implicit lock request. Programs can either explicitly request the lock be released or perform an operation that results in an implicit unlock request. CDCS locks have both type and a level of effect.

CDCS maintains two types of locks as follows:

Exclusive lock; prohibits concurrent access.

Protected lock; allows concurrent reading.

When an application program holds an exclusive lock on a file or record, other users can neither read nor update the file or record. When an application program holds a protected lock on a file or record, other users can read but not update the file or record.

CDCS locks are in effect at either of the following levels:

Record

File (realm)

If a file lock is in effect, the record locking mechanism is no longer applicable for the particular file.

Both protected and exclusive locks can be in effect on either the record level or the file level.

Locking Outside of a Transaction

Outside of a transaction, an application program can obtain a protected record or file lock or an exclusive file lock. An application program cannot obtain an exclusive record lock outside of a transaction.

Protected Record Lock

Protected record locks are implicit locks. Protected record locks are the only possible record level locks when a program is updating a record outside of a transaction.

Two requirements must be met for the protected record locking mechanism to be in effect:

The application program must have opened the file for input/output.

The application program must have no explicit file lock in effect for the file.

An application program obtains a protected record lock as a result of a read operation. Under these conditions, CDCS considers a read operation as an indication of the intent to update the record; therefore, CDCS puts an implicit protected record lock on the record for the application program.

Outside of a transaction, an application program can hold a protected record lock on only one record in a particular file (area) at a time. An application can have multiple protected record locks at one time; each locked record must be in a different file.

To release a protected record lock established outside of a transaction, the application program must perform one of the following operations:

Read another record in the file (realm).

Delete or rewrite the record.

Write another record.

Close the file (realm).

Unlock any data base file; that is, execute the C.UNLOK routine (COBOL) or the UNLOCK statement (FORTRAN).

In addition to the preceding operations, a protected record lock is released when a deadlock situation is encountered or when the program terminates, normally or abnormally.

File Lock

An application program obtains an explicit file lock by issuing an explicit lock request on the realm associated with the file. Because the file lock limits other users' access to the file and because the file lock overrides the checking capability of the record lock, the use of a file lock is not recommended unless the data is so sensitive that its use is justified. The following statements and routines issue the file lock request:

C.LOK routine (COBOL)

Obtains a protected lock on the file specified in the routine.

DB$LKAR routine (COBOL)

Obtains the specified lock (exclusive or protected) on the file specified in the routine.

LOCK statement (FORTRAN)

Obtains the specified lock (exclusive or protected) on the file specified in the routine.

When an entire file has been locked by an explicit protected lock, any other program that has opened the file for input only can read, but not update, the file. Any other program that has opened the file for input/output cannot access the file.

When an entire file has been locked by an explicit exclusive lock, no other program can access the file.

To release the explicit file lock, an application program must perform one of the following operations:

Close the file.

Unlock the file; that is, execute the C.UNLOK routine (COBOL) or the UNLOCK statement (FORTRAN).

An application program obtains an implicit protected file lock by opening a file for output only and not explicitly locking the file. As with an explicit protected file lock, other programs that have opened the file for input only can read the file (assuming that records have been written to the file). Other programs that have opened the file for input/output must wait for access until the lock is released. This implicit file lock is released when the file is closed, although it could be released by an unlock request.

Locking Within a Transaction

Within a transaction, an application program can obtain both protected and exclusive record locks. An application program cannot obtain a file lock within a transaction.

Protected Record Lock

A protected record lock is an implicit record lock obtained when an application program reads a record with intent to update. (The application program must have opened the realm for input/output.) Within a transaction, an application program can hold multiple protected record locks on records that are read but not updated. An application program can hold multiple protected record locks on records in each file. The number of protected record locks held depends on the number of records read within the transaction.

Protected record locks are not released within a transaction. Protected record locks are released when an application program commits the transaction, drops the transaction, terminates normally or abnormally, or encounters a deadlock situation. The protected record locks are also released if an application program, utilizing the immediate return feature, encounters a resource conflict. Refer to the Immediate Return subsection and the Deadlock subsection for more information about these situations.

Exclusive Record Lock

An exclusive record lock is an implicit lock that is assigned when an application program updates a record during transaction processing. Exclusive record locks are possible only within a transaction.

CDCS establishes an exlusive record lock when an application program writes a new record. CDCS converts a protected record lock to an exclusive record lock when an application program updates (deletes or rewrites) a record read with intent to update. CDCS prohibits another application program from reading a record with an exclusive record lock. CDCS establishes an exclusive record lock even for a deleted record.

Within a transaction, an application program can hold several exclusive record locks on a particular file (area) or on more than one file (area) at a time. The number of exclusive record locks held depends on the number of records written or updated within the transaction.

Exclusive record locks are not released during a transaction. Exclusive record locks are released by CDCS when the application program commits the transaction, drops the transaction, terminates normally or abnormally, or encounters a deadlock situation. Exclusive record locks are also released when an application program, utilizing the immediate return feature, encounters a resource conflict. Refer to both the Deadlock subsection and the Immediate Return subsection for more information about these situations.

Processing Considerations

REWRITE or DELETE statements must be preceded by a READ statement to lock the record, or the entire file must be locked by the program. CDCS rejects the request if the file or record is not locked. If the program has no record locked in the file, CDCS issues error diagnostic 436 (664 octal).

Coordinated updates involving a record from each of several files require that the user lock all pertinent records before attempting the updates.

For coordinated updates involving records from the same file, the user should either lock the entire file or use transaction processing. Usually, a file should be locked immediately after being opened. CDCS does not allow an explicit file lock if any record lock exists for that program; therefore, if a file is opened for input/output and a read operation is performed, CDCS does not allow an explicit lock to follow the read operation. An explicit file lock must precede a read operation if the file is open for input/output. CDCS does allow an explicit lock to follow a delete, rewrite, or unlock operation, however.

For reading relations, CDCS puts a protected record lock on all records in a given relation occurrence if the files joined in the relation have been opened for input/output. CDCS does not perform locking if files in a relation have been opened for input.

Outside of a transaction, CDCS releases the record locks placed on the records that are retrieved by a relation read when another relation read operation is requested. If a READ relation statement is followed by a DELETE or REWRITE statement, CDCS releases the specific lock involved after completion of the delete or rewrite operation.

Within a transaction, CDCS does not release the record locks placed on the records retrieved by a relation read operation when another relation read operation is requested. CDCS assigns a protected record lock to the records retrieved by the second relation read request. If a READ relation statement is followed by either a DELETE or REWRITE statement, CDCS converts the specific protected record lock to an exclusive record lock.

When a program is reading records from files that are open for input, other jobs can be in the process of modifying or deleting the same records in those files at the same time. The records that are returned can, therefore, contain obsolete or inconsistent data. To avoid obtaining inconsistent results, a program that is reading files can lock the records either by issuing explicit file lock requests or by opening the files for input/output.

The locking facilities provided by CDCS allow for the possibility of a deadlock situation. The next subsection details this concept.

DEADLOCK

Deadlock can occur when two or more application programs contend for files and records that are in a locked state. A deadlock situation can arise as a result of either the automatic locking of records by CDCS or an explicit user lock requests.

For example, the simplest deadlock situation (illustrated in figure 5-14) can occur as follows:

1. Program PGRM1 has a lock on record 100 in FILEA.
2. Program PGRM2 has a lock on record W in FILEB.
3. Both programs are attempting to read with intent to update the record locked by the other program.

In this situation, neither program can continue processing because the other program has a lock on the record required for continued processing.

If deadlock occurs during COBOL or FORTRAN processing, CDCS selects one of the contending programs, releases all locked resources (files or records) held by that program, and issues the deadlock error status code 435 (663 octal). Figure 5-14 shows CDCS releasing the lock for program PGRM2 and issuing the deadlock error code. The code, along with the realm name and version name associated with the file, can be obtained by the application program; refer to the Status Checking subsection in sections 2 and 3.

In the COBOL program, the USE FOR DEADLOCK statement identifies the procedure to be executed when a deadlock situation occurs. In a FORTRAN program, program logic can determine whether or not a deadlock has occurred by testing for the deadlock error status code after a DML statement is executed. When a deadlock situation does occur, the program should ensure that the input/output statements that locked the resources are repeated.

When a deadlock situation occurs involving a program engaged in transaction processing, CDCS releases all record locks held by the program. CDCS reverses the updates made within the uncompleted transaction. The records are restored to their respective states before the transaction began.

In any situation, the programmer is responsible for handling recovery from a deadlock situation by means of appropriate code in the application program. Refer to sections 2 and 3 for examples of handling deadlock.

In order to deal effectively with deadlock, the user must adopt a coding discipline that allows detection of deadlock and recovery from this situation prior to opening any file for input/output. If the user is performing coordinated updates; that is, updating records as a result of reading related files, the user should ensure that all the required records have been locked.

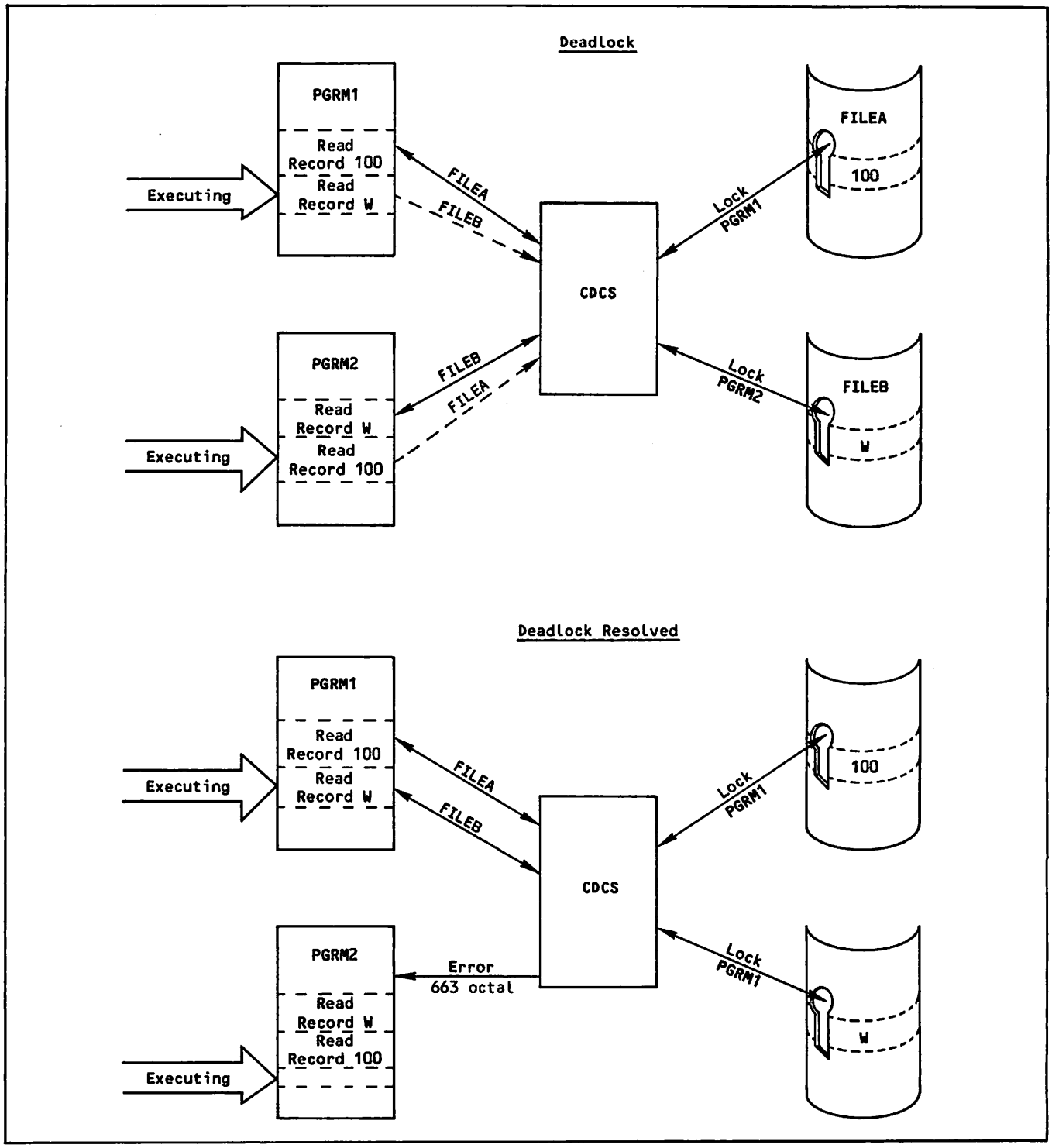


Figure 5-14. Deadlock Situation

IMMEDIATE RETURN FEATURE

The immediate return feature of CDCS provides COBOL and FORTRAN application programs with the ability to receive an immediate response from CDCS when certain resource conflicts (explained in the following subsection) occur. The immediate return feature provides for special error handling so that control returns to the application program when a fatal error occurs. Control returns to the application program when a nonfatal error occurs whether or not the immediate return feature is enabled. When control is returned, the application program should contain logic to determine the action taken.

The immediate return feature can be enabled or disabled by the application program depending on processing circumstances.

In the TAF/CDCS environment the immediate return capability is automatically enabled. The setting of immediate return during TAF processing, therefore, is redundant. Also, the immediate return capability cannot be disabled in TAF processing even though the request to disable immediate return executes without error.

RESOURCE CONFLICTS

If a resource conflict occurs and immediate return is not enabled, an application program must wait until CDCS gains access to the permanent files held by other users. Immediate return offers the application program the ability to provide program logic that determines the action taken when such a conflict occurs.

A resource conflict occurs during a request for version change when a permanent file involved in the version requested cannot be immediately attached. If this situation occurs when immediate return is enabled, CDCS returns control to the program after issuing the following nonfatal error message:

388 (604 octal) - PF WAIT ON AREA an

If immediate return is not enabled, CDCS issues an informative message and the application program must wait until CDCS gains access to the permanent file.

A resource conflict occurs when an application program requests a record or a file that is held (locked) by another user. If this situation occurs when immediate return is enabled, CDCS releases all the locks currently held by the program. If the program is currently involved in transaction processing, CDCS drops the transaction. CDCS returns control to the program after issuing the following nonfatal error message:

387 (603 octal) - LOCKED RECORD/AREA --
REQUEST NOT PROCESSED

If immediate return is not enabled, the program must wait until CDCS gains access to the locked record or file.

Infrequently, a resource conflict occurs when CDCS initiates an internal request to switch the journal log file (a file that contains historical records concerning user operations). The conflict occurs if CDCS cannot immediately attach the new journal log file. If immediate return is enabled, CDCS disconnects the application program, disables immediate return, and returns control to the application program after issuing the following fatal error message:

418 (642 octal) - LOG FILES NOT AVAILABLE
FOR SCHEMA sn

If immediate return is not enabled before the internal request to switch the journal log file is issued by CDCS, the program must wait until CDCS gains access to the new journal log file.

Immediate return cannot be enabled before CDCS is invoked; therefore, any resource conflicts during CDCS invocation result in the user waiting for CDCS to gain access to the files. Possible resource conflicts include the attaching of data base files, logging files, or files required by CDCS during automatic recovery.

USING THE IMMEDIATE RETURN FEATURE

The immediate return feature cannot be enabled before CDCS is invoked. Immediate return is enabled when a FORTRAN program calls the DMLSIR routine, or when a COBOL program enters the DB$SIR routine with correct parameter values specified. If the program has not been disconnected from CDCS (no fatal errors have occurred), a second call with correct parameter values specified to either DMLSIR from a FORTRAN program or DB$SIR from a COBOL program disables the immediate return feature.

A data base status block should be declared in a FORTRAN or COBOL program if the program is to utilize the immediate return feature. The data base status block should be defined to be at least four words in length. If errors occur during execution, word 1 of the data base status block contains the error number; word 4 of the data base status block contains the severity of the error. The program should check both words 1 and 4 since some CDCS errors do not have error numbers. If both word 1 and word 4 are zero, no error has occurred. Action taken by the program should depend on the information obtained from the data base status block. (Refer to the Data Base Status Block subsection in section 2 or 3 for more information.)

PROCESSING CONSIDERATIONS

If a fatal error occurs when the immediate return capability is enabled, the program is disconnected from CDCS and immediate return is automatically disabled. Program logic can determine the action to be taken; for example, a FORTRAN or COBOL program could complete any processing needed before program termination.

CONSTRAINTS DEFINED

The constraint facility of CDCS allows controls to be established and maintained on update operations involving two logically associated files or two data items within the same file. A constraint is a means for imposing an integrity control on associated files or items within a single file.

Constraints are established for the purpose of protecting the integrity of data in a data base during update operations by application programs. Information about constraints that affect the application program should be supplied by the data administrator.

When a program attempts to update a data base file involved in a constraint, CDCS evaluates the effect of the operation on the elements in the constraint before the update operation is performed. If the update operation does not violate the constraint, CDCS allows the update to execute normally. If the update operation would violate the constraint, CDCS does not allow the update to execute.

A constraint is defined in the schema. Common data items within records in two files or within a single file are used to establish a logically dependent condition. In a constraint, one record is defined as the dominant record, and the other record is defined as the dependent record. A dominant record corresponds to a dependent record if both records contain the same value for the item that connects them.

A constraint ensures the following rules are maintained in the data base:

A dependent record occurrence cannot exist in the data base unless a dominant record with the same value for the common data item also exists.

A dominant record cannot be deleted from the data base if a dependent record occurrence exists with the same value for the common data item.

The common data item of a dominant record cannot be changed if a dependent record occurrence exists with the same value for the common data item.

SINGLE-FILE CONSTRAINT

A constraint can be defined for two logically associated items within a single file. In a single-file constraint, one data item in a record participates in the dominant role, and another data item in the same record participates in the dependent role. A typical example is an employee file in which each record contains among other items an employee number and a manager number, where the manager number conforms to the structure of the employee number.

Figure 5-15 illustrates the concept of a single-file constraint. In an EMPLOYEES file, the EMPLOYEE record contains the data item EMP-NO to indicate an employee number and the data item MNGR-NO to indicate the employee number of the manager to whom the employee reports. A constraint is defined for these items. The item MNGR-NO is dependent on the item EMP-NO in the constraint EMPLOYEE-MANAGER. (The arrow in the diagram points from the dependent item to the dominant item.) When the file is involved in the constraint, no employee record can be stored in the file if an employee record for the manager of the employee does not exist. Also, an employee record for a manager cannot be deleted from the file if one or more employee records with the corresponding manager number exist. Two occurrences of the EMPLOYEE record are included in figure 5-15 because a dominant record must exist in the file; that is, a record must exist where an employee has the same value for both EMP-NO and MNGR-NO. This record should be the first record entered into the file and the last record removed from the file.

TWO-FILE CONSTRAINT

When a constraint is defined for two logically associated files, the data administrator establishes a dependent condition between records in the files based on data items common to both records. In a two-file constraint, a record in one file is defined as the dominant record, and a record in the other file is defined as the dependent record.

Figure 5-16 illustrates the dominant-dependent condition in a two-file constraint. A DEPARTMENTS file and an EMPLOYEES file in a data base both contain a data item DEPT-NO that indicates a department number within each file. The DEPARTMENT record is assigned the dominant role and the EMPLOYEE record is assigned the dependent role. The arrow in the figure indicates the dependency of the EMPLOYEE record on the dominant DEPARTMENT record. When the files are protected by the constraint, no record can be stored in the EMPLOYEES file if no corresponding dominant record occurrence exists in the DEPARTMENTS file. Also, a record cannot be deleted in the DEPARTMENTS file if a dependent record occurrence exists in the EMPLOYEES file.

Only two files can be associated in a single constraint. However, a file can be involved in more than one constraint.

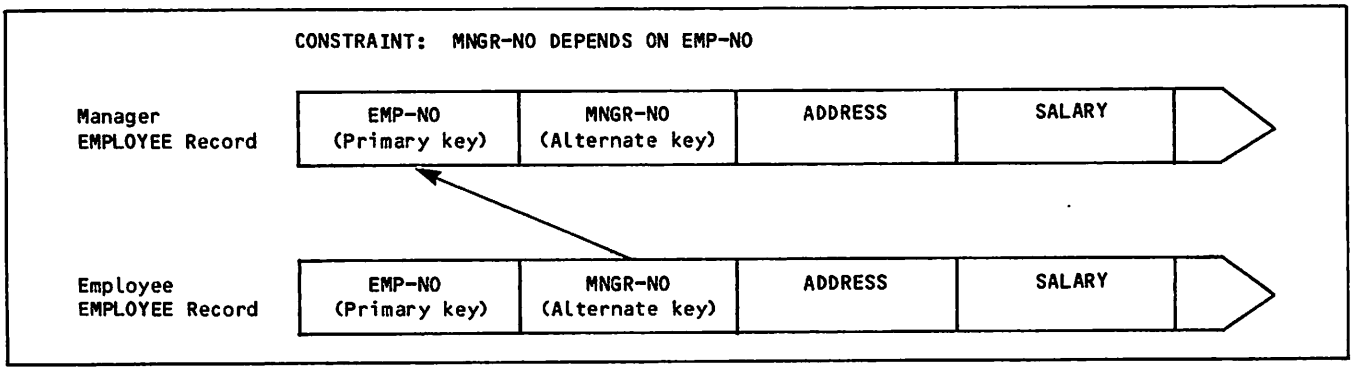


Figure 5-15. Single-File Constraint Example

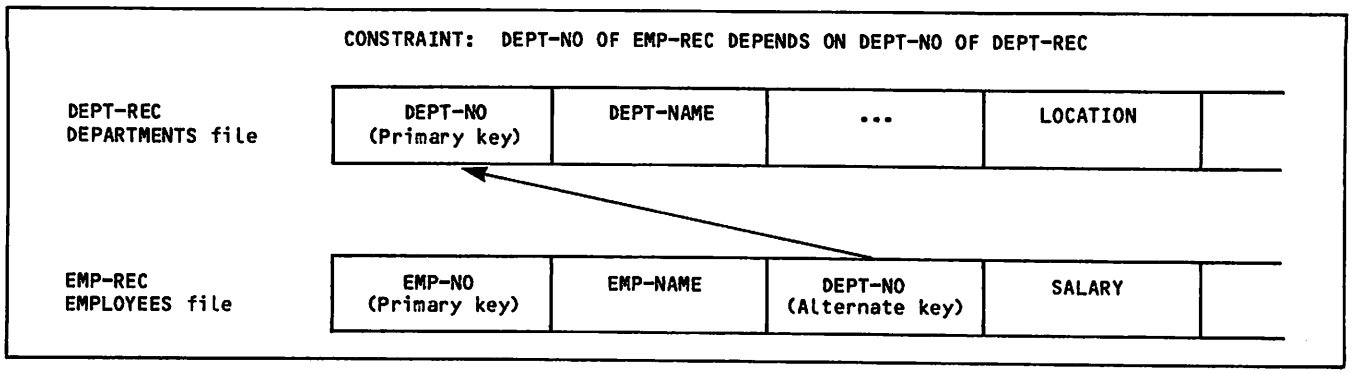


Figure 5-16. Two-File Constraint Example

CDCS CONSTRAINT PROCESSING

CDCS enforces constraints specified in the schema during update processing by application programs. CDCS does not permit a write, delete, or rewrite operation to be performed on records in data base files if the operation would violate the constraint.

When an application program attempts a write, delete, or rewrite that would violate a constraint, CDCS diagnoses the violation as a nonfatal error; the operation is terminated, but the application program is permitted to continue processing. CDCS returns the code (395 octal 601 octal) to the application program if the program defines and uses the status checking fields. The diagnostic message is written to the user error file and to the CDCS output file.

The subschema being used by an application program might specify only one of the two files involved in a constraint in the schema. In order to process the constraint, CDCS must attach both files.

When a constraint is imposed on a file, the method of updating the file differs depending on whether a one-file constraint or a two-file constraint is involved. The following subsections contain guidelines for updating files involved in either a one-file or two-file constraint.

Guidelines for File Creation

Data base files involved in a constraint must be created in a particular order. Dominant records must be created before the dependent records.

Single-File Constraints

In a single-file constraint, at least one record exists that has no dominant record; the dominant record and dependent record must have the same value. This situation occurs in the single-file constraint example (figure 5-15) for the employee who has no manager. The record for this employee must have the same value for both EMP-NO and MNGR-NO.

For creating a file that is controlled by a single-file constraint, the following operations must be followed in the order shown:

1. Open the file for creation.
2. Write the records that have no dominant record to the new file.
3. Close the file.
4. Reopen the file for input/output and add dependent records.

Two-File Constraint

For a two-file constraint the file containing the dominant records must be created first. Files involved in two-file constraints must be created in a particular order as follows:

1. Create the file containing the dominant records (that is, open the file for creation, provide the file with records, and close the file).
2. Create the file containing the dependent records.

Guidelines for Insertion Operations

A write operation on a file with a constraint is permitted by CDCS only if the dominant-dependent condition is not altered by the operation.

To avoid constraint violations, perform write operations as follows:

Write a dominant record occurrence before writing any dependent record occurrences with the same value for the common data item.

Write a dependent record occurrence only if a dominant record occurrence exists with the same value for the common data item.

Single-File Constraints

In the example of the single-file constraint (figure 5-15), the item EMP-NO in the EMPLOYEE record is defined as the dominant item of the constraint and the item MNGR-NO in the EMPLOYEE record is defined as the dependent item. A new EMPLOYEE record for an employee can be added to the EMPLOYEES file only if the value of the MNGR-NO item for the new record

matches the value of the EMP-NO item in a manager's EMPLOYEE record in the file. In other words, a new EMPLOYEE record can be written only if an EMPLOYEE record for the employee's manager exists. If there is no EMPLOYEE record for the manager, CDCS rejects the write request from the application program and issues a nonfatal error message.

Two-File Constraints

In the example of the two-file constraint (figure 5-16), the department number DEPT-NO has been used to assign the DEPARTMENT record the role of dominant record and the EMPLOYEE record the role of dependent. A new EMPLOYEE record can be written to the EMPLOYEES file only if a DEPARTMENT record with the same department number exists in the DEPARTMENTS file. If there is no corresponding DEPARTMENT record, CDCS rejects the write request from the application program.

Guidelines for Deletion Operations

Delete operations on a file on which a constraint has been imposed are permitted by CDCS only if the dominant-dependent condition is not altered by the operation.

To avoid constraint violations, perform delete operations as follows:

Delete dependent record occurrences before deleting the dominant record with the same value for the common data item.

Delete a dominant record only if no dependent record occurrence exists with the same value for the common data item.

Single-File Constraints

For a single-file constraint, the value of the dominant item in the record to be deleted and the value of the dependent item in the remaining records in the file are used in determining whether or not a record can be deleted from a file.

In figure 5-15, an EMPLOYEE record cannot be deleted from the EMPLOYEES file if any EMPLOYEE record has a value for the MNGR-NO item (the dependent item) that matches the EMP-NO item (the dominant item) of the record to be deleted. In general, an EMPLOYEE record for a manager can be deleted only if there are no EMPLOYEE records for the employees of that manager. If one or more EMPLOYEE records with the manager's employee number do exist, the EMPLOYEE record for the manager cannot be deleted. CDCS rejects the delete request and issues a nonfatal diagnostic.

Two-File Constraints

In the example of the two-file constraint (figure 5-16), no DEPARTMENT record (the dominant record) can be deleted from the DEPARTMENTS file if any EMPLOYEE record (the dependent record) in the EMPLOYEES file has a department number matching that of the record to be deleted. If one or more corresponding EMPLOYEE records do exist, CDCS does not permit the DEPARTMENT record to be deleted.

Guidelines for Modification Operations

The guidelines for modification operations on files involved in either a one-file constraint or a two-file constraint are the same. Modification of a record occurrence in a file involved in a constraint is restricted according to the rules for insertion and deletion, when the value of the common data item in the constraint is to be changed. To modify the common data item in a dominant record and to introduce the new value in the dependent records, the application programmer must perform a sequence of update operations. The operations performed depend on whether the data item is a primary key or an alternate key.

If the common data item is a primary key, the following operations must be performed in the order shown:

1. Write the dominant record with the new value in the common data item.
2. Read a dependent record; change the value of the common data item to the new value contained in the dominant record; rewrite the dependent record. (Perform this step for each dependent record of the dominant record.)
3. Delete the dominant record with the old value.

If the common data item to be modified in a dominant record is an alternate key, the following operations must be performed in the order given:

1. Write each dependent record containing the old value of the common data item to a temporary file.
2. Delete each dependent record containing the old value of the common data item from the data base file.
3. Read the dominant record; change the value of the common data item to the new value; rewrite the dominant record.
4. Read a dependent record from the temporary file; change the value of the common data item to the new value contained in the dominant record; write the dependent record to the data base file. (Perform this step for each dependent record of the dominant record.)

Single-File Constraint

Figure 5-15 can be used to show the concept of modifying a file involved in a one-file constraint. The employee number (EMP-NO) of the EMPLOYEE record for a manager cannot be changed unless the dependent records of the employees reporting to that manager are first deleted.

Two-File Constraint

For a two-file constraint, the DEPARTMENTS and EMPLOYEES files, shown in figure 5-16, can be used once again to aid in understanding the concept of controlling file modifications through constraints. A rewrite operation cannot be performed on a DEPARTMENT record (the dominant record) if the operation would change the value of the

common data item containing the department number (DEPT-NO), which is the primary key, and if one or more EMPLOYEE records (the dependent record) within the EMPLOYEES file have the same department number as the DEPARTMENT record to be modified.

A rewrite operation is not permitted on an EMPLOYEE record (the dependent record) if the updated value of the department number does not already exist in a DEPARTMENT record (the dominant record).

TAF-CDCS PROCESSING

TAF tasks coded in the COBOL 5 and FORTRAN Extended 4 programming languages can interface with CDCS. The syntax used to code a task is the same as the syntax used to code a program for execution in batch mode through CDCS, with a few exceptions. The coding for a task must include TAF directives and must provide for a communication block. Also, TAF prohibits a task from making some requests that are allowed to be made by an application program executing in batch mode.

CDCS treats an executing task as a run-unit. Usually, a run-unit is comprised of several tasks that are linked together by TAF as a task chain. For tasks within a task chain, there are coding requirements for the tasks that use CDCS. Each task must invoke CDCS and specify the same subschema and data base version.

If tasks in a run-unit must access realms with controlled access, the first task of the chain must specify the required access control (privacy) keys. The access control privileges given that task remain in effect until the run-unit terminates its connection with CDCS.

CDCS handles processing TAF tasks differently than batch programs in situations where record or file access cannot be provided upon request. When a locked record or file is in contention, CDCS releases all locks held for the tasks in the task chain and returns control to TAF. This procedure ensures that a task is never waiting for record or for file access. This situation can be caused either by deadlock or by a record or file lock by another user program.

These situations must be handled in the tasks by program logic. Program logic can test for the situations. The deadlock situation is indicated by CDCS with the status code 435 (663 octal). The situation of the record or file lock is indicated by CDCS with the status code 387 (603 octal). These codes are returned to the data base status block or to other status checking elements. (Refer to sections 2 and 3 for discussions of status checking.) If these situations occur, program logic should provide processing to reestablish the released locks.

When CDCS cannot immediately attach a file needed for processing, TAF handles the situation differently depending on what stage of processing is involved. If files needed during invoke processing are not immediately available, CDCS issues a diagnostic and returns control to TAF. TAF aborts the executing task. This procedure ensures that a task never waits to invoke CDCS. If CDCS cannot immediately attach the files needed for a version change request, CDCS issues a diagnostic message and returns control to TAF. TAF passes the message to the executing task. Program logic can then determine the processing that is to follow.

Detailed information about using TAF is contained in the TAF reference manual.

COBOL EXAMPLE 6

A COBOL application program that accesses a CDCS controlled data base is shown in this section. The application uses a manufacturing data base that is shown in appendix H. This appendix contains all the jobs required to set up the data base environment. The schema should be referenced for access control keys required to access the realms included in the subschemas. The data stored in the data base is shown in the Query Update job that creates the data base files.

The COBOL program uses subschema C5SS-PRODUCT-MANAGEMENT shown in figure 6-1. The listing shows a three-realm relation, DPD-REL, which joins realms DEPARTMENTS, PROJECTS, and PRODUCTS. The listing also shows that a restriction has been placed on data item STATUS-CODE in record PRODREC (realm PRODUCTS). With the restriction, only record occurrences that meet the requirement specified in the restriction (that is, the value of STATUS-CODE must equal A or N) can be returned to the application program when the relation is read.

The subschema must be avaliable when the COBOL program is compiled. The control statements that are used to attach the subschema and compile and execute the application program are as follows:

```
ATTACH,subschema-library ...
COBOL5,D=subschema-library.
LGO.
```

The D parameter of the COBOL5 control statement must specify the local file name of the file on which the subschema directory resides. The system default file LGO is specified as the file containing the relocatable binary program being executed.

The COBOL application program reads a relation and produces a report by using control break and null record information returned in the data base status block. The program, named RELREAD, is shown in figure 6-2. The Special-Names clause specifies the name of the subschema. USE FOR ACCESS CONTROL statements are specified in the DECLARATIVES section to provide the access control keys. Because the realms are only being read, access keys for retrieval provide sufficient realm access for the application program. The program must specify access control keys as follows:

VERY*PRIVATE required for any access of realm DEPARTMENTS

VERIFIED-INPUT required for retrieval from realm PROJECT

ACCESS(/)OK required for any access of realm PRODUCTS

To obtain information for the report, the program positions the relation and then sequentially reads the relation. First, the value of the primary key for the root realm (data item DEPT-NO for realm DEPARTMENTS) is set. Then the START statement is executed to position the realm. Execution of the subsequent sequential READ statement causes the relation DPD-REL to be read.

The data base status block is used to check the status of data base operations. Program logic determines the action taken when a control break or null record occurs.

The program uses control break information to test for the management levels. A department has projects that develop products. The levels appear as follows:

```
Department
    Projects
        Products
```

A control break value of 2 indicates that the program is reading information about a new department. A control break value of 3 indicates that the program is reading information about a new project. The report generated shows all products assigned to a project and all the projects assigned to a department.

The program uses null record information to test for qualifying records; a nonzero value indicates there is no qualifying record. When this occurs, the program does not print data but initiates another read.

After the program reads the relation, it closes the files.

The report generated by the execution of RELREAD is shown in figure 6-3.

```
                                * SOURCE LISTING *   (82029)  DDL  3.2+552.

              00001                 TITLE DIVISION.
              00002                     SS  C5SS-PRODUCT-MANAGEMENT WITHIN MANUFACTURING-DB.
              00003
              00004                 ALIAS DIVISION.
              00005                     AD  REALM DEVELOPMENT-PRODUCTS BECOMES PRODUCTS.
              00006                     AD  RECORD DEVREC BECOMES PRODREC.
              00007                     AD  DATA CUM-TEST-AVERAGE BECOMES CUMULATIVE-AVERAGE.
              00008
              00009                 REALM DIVISION.
              00010                     RD DEPARTMENTS, PROJECT, PRODUCTS.
              00011
              00012                 RECORD DIVISION.
              00013                 01  DEPTREC.                                        ** WITHIN DEPARTM
              00014                     03  DEPT-NO                PICTURE X(4).        ** ORDINAL    1
              00015                     03  DEPT-NAME              PICTURE X(20).       ** ORDINAL    2
              00016                     03  MGR-ID                 PICTURE X(8).        ** ORDINAL    3
              00017                     03  MGR-NAME               PICTURE X(20).       ** ORDINAL    4
              00018
              00019                 01  PROJREC.                                        ** WITHIN PROJECT
              00020                     03  PROJECT-ID             PICTURE X(10).       ** ORDINAL    1
              00021                     03  PROJ-DESCR             PICTURE X(40).       ** ORDINAL    2
              00022                     03  RESPONSIBILITY         PICTURE X(8).        ** ORDINAL    3
              00023
              00024                 01  PRODREC.                                        ** WITHIN PRODUCT
              00025                     03  PRODUCT-ID             PICTURE X(10).       ** ORDINAL    1
              00026                     03  PROJECT-ID             PICTURE X(10).       ** ORDINAL    2
              00027                     03  STATUS-CODE            PICTURE A.           ** ORDINAL    3
              00028
PRIMARY KEY   00014       DEPT-NO FOR AREA DEPARTMENTS
ALTERNATE KEY 00016       MGR-ID FOR AREA DEPARTMENTS
PRIMARY KEY   00020       PROJECT-ID FOR AREA PROJECT
ALTERNATE KEY 00022       RESPONSIBILITY FOR AREA PROJECT
PRIMARY KEY   00025       PRODUCT-ID FOR AREA PRODUCTS
ALTERNATE KEY 00026       PROJECT-ID FOR AREA PRODUCTS
              *****       RECORD MAPPING IS NEEDED FOR REALM -   DEPARTMENTS
              *****       RECORD MAPPING IS NEEDED FOR REALM -   PROJECT
              *****       RECORD MAPPING IS NEEDED FOR REALM -   PRODUCTS
              00029                 RELATION DIVISION.
              00030                     RN IS DPD-REL
              00031                         RESTRICT PRODREC WHERE STATUS-CODE EQ "A"
              00032                                            OR  STATUS-CODE EQ "N".
              00033
              00034
              *****       END OF SUB-SCHEMA SOURCE INPUT

              *****             RELATION   STATISTICS                *****
RELATION 001              DPD-REL JOINS          AREA - DEPARTMENTS
                                                 AREA - PROJECT
                                                 AREA - PRODUCTS


              -----       BEGIN SUB-SCHEMA FILE MAINTENANCE            -----

        SUBSCHEMA                              CHECKSUM
        C5SS-PRODUCT-MANAGEMENT                3422331606054476417 2

              -----       END OF FILE MAINTENANCE              -----
DDL COMPLETE.          0 DIAGNOSTICS.
```

Figure 6-1. COBOL Subschema

```
1
2               IDENTIFICATION DIVISION.
3               PROGRAM-ID.  RELREAD.
4               ENVIRONMENT DIVISION.
5               CONFIGURATION SECTION.
6               SOURCE-COMPUTER.  CYBER-170.
7               OBJECT-COMPUTER.  CYBER-170.
8               SPECIAL-NAMES.
9                   SUB-SCHEMA IS C5SS-PRODUCT-MANAGEMENT.
10              INPUT-OUTPUT SECTION.
11              FILE-CONTROL.
12                  SELECT PRINT-FILE ASSIGN TO OUTPUT.
13              DATA DIVISION.
14              FILE SECTION.
15              FD PRINT-FILE
16                  LABEL RECORDS ARE OMITTED
17                  DATA RECORD IS PRINT-LINE.
18              01  PRINT-LINE                  PIC X(136).
19              WORKING-STORAGE SECTION.
20              01  ACCESS-KEY                  PIC X(30).
21              01  DEPART-KEY                  PIC X(4) VALUE "M130".
22              01  EOF                         PIC 9    VALUE ZERO.
23                  88  END-OF-REALM            VALUE 1.
24              01  BLK-LENGTH                  PIC 99    USAGE IS COMP-1.
25              01  DATABASE-STATUS-BLOCK.
26                  02  DATABASE-STATUS         PIC 9(5)  USAGE IS COMP-1.
27                  02  AUXLIARY-STATUS.
28                      03  DB-ITEM-ORDINAL     PIC 9(5)  USAGE IS COMP-1.
29                      03  DB-FILE-POSITION    PIC 9(3)  USAGE IS COMP-1.
30                      03  DB-SEV-CODE         PIC 9(3)  USAGE IS COMP-1.
31                  02  DB-FUNCTION             PIC A(10).
32                  02  RELATION-RANK-STATUS.
33                      03  DB-REL-RANK-ERROR   PIC 9(3)  USAGE IS COMP-1.
34                      03  DB-REL-RANK-CTLBK   PIC 9(3)  USAGE IS COMP-1.
35                      03  DB-REL-RANK-NULL    PIC 9(3)  USAGE IS COMP-1.
36                  02  DB-REALM                PIC X(30).
37              01  HLINE-1.
38                  03  FILLER                  PIC X(5)  VALUE "0    ".
39                  03  FILLER                  PIC X(11) VALUE "DEPARTMENT ".
40                  03  DEPT-OUT                PIC X(20).
41                  03  FILLER                  PIC X(10) VALUE SPACES.
42                  03  FILLER                  PIC X(8) VALUE "MANAGER ".
43                  03  MGR-OUT                 PIC X(20).
44                  03  FILLER                  PIC X(60) VALUE SPACES.
45              01  HLINE-2.
46                  03  FILLER                  PIC X(5)  VALUE SPACES.
47                  03  FILLER                  PIC X(15) VALUE "PROJECT NUMBER".
48                  03  PROJID                  PIC X(10).
49                  03  FILLER                  PIC X(16) VALUE SPACES.
50                  03  FILLER                  PIC X(8)  VALUE "PROJECT ".
51                  03  PROJ-NAME               PIC X(40).
52                  03  FILLER                  PIC X(48) VALUE SPACES.
53              01  HLINE-3.
54                  03  FILLER                  PIC X(10) VALUE SPACES.
55                  03  FILLER                  PIC X(8)  VALUE "PRODUCT ".
56                  03  PRODID                  PIC X(10).
57                  03  FILLER                  PIC X(23) VALUE SPACES.
58                  03  FILLER                  PIC X(15)
59                                                   VALUE "PRODUCT STATUS ".
60                  03  STAT                    PIC A.
61                  03  FILLER                  PIC X(97)  VALUE SPACES.
62
63              PROCEDURE DIVISION.
64              DECLARATIVES.
65              ACCESS-CONTROL-1 SECTION.
66                  USE FOR ACCESS CONTROL ON INPUT;
67                      KEY IS ACCESS-KEY FOR DEPARTMENTS.
68              010-PAR-1.
69                  MOVE "VERY*PRIVATE" TO ACCESS-KEY.
70              ACCESS-CONTROL-2 SECTION.
71                  USE FOR ACCESS CONTROL ON INPUT;
72                      KEY IS ACCESS-KEY FOR PROJECT.
```

Figure 6-2. COBOL Program (Sheet 1 of 2)

```
020-PAR-2.
    MOVE "VERIFIED-INPUT" TO ACCESS-KEY.
ACCESS-CONTROL-3 SECTION.
    USE FOR ACCESS CONTROL ON INPUT;
        KEY IS ACCESS-KEY FOR PRODUCTS.
030-PAR-3.
    MOVE "ACCESS(/)OK" TO ACCESS-KEY.
END DECLARATIVES.

0100-MAIN-LOGIC SECTION.
0110-START-UP.
    PERFORM 0200-OPENING.
    PERFORM 0300-POSITION-RELATION.
    PERFORM 0410-READ-REL UNTIL END-OF-REALM.
    PERFORM 0800-CLOSING.

0200-OPENING SECTION.
0210-BEGIN-OPEN.
    MOVE 11 TO BLK-LENGTH.
    ENTER "DB$DBST" USING DATABASE-STATUS-BLOCK, BLK-LENGTH.
    OPEN OUTPUT PRINT-FILE.
    OPEN INPUT DPD-REL.
    PERFORM 0700-STATUS-CHECK.

0300-POSITION-RELATION SECTION.
0310-POSITION-REL.
    MOVE DEPART-KEY TO DEPT-NO.
    START DPD-REL KEY IS EQUAL TO DEPT-NO.

0400-READ-WRITE-RELATION SECTION.
0410-READ-REL.
    READ DPD-REL NEXT RECORD AT END MOVE 1 TO EOF.
    PERFORM 0700-STATUS-CHECK.
    IF DB-REL-RANK-NULL EQUAL TO ZERO
        PERFORM 0500-WRITE-REPORT.

0500-WRITE-REPORT SECTION.
0510-CHECK-CONTROL-BREAK.
    IF DB-REL-RANK-CTLBK EQUALS 2
        PERFORM 0610-WRITE-DEPARTMENT
        THRU 0630-WRITE-PRODUCT
    ELSE IF DB-REL-RANK-CTLBK EQUALS 3
        PERFORM 0620-WRITE-PROJECT
        THRU 0630-WRITE-PRODUCT
    ELSE PERFORM 0630-WRITE-PRODUCT.

0600-WRITE-HEADING SECTION.
0610-WRITE-DEPARTMENT.
    MOVE DEPT-NAME TO DEPT-OUT.
    MOVE MGR-NAME TO MGR-OUT.
    WRITE PRINT-LINE FROM HLINE-1.
0620-WRITE-PROJECT.
    MOVE PROJ-DESCR TO PROJ-NAME.
    MOVE PROJECT-ID IN PROJECT TO PROJID.
    WRITE PRINT-LINE FROM HLINE-2.
0630-WRITE-PRODUCT.
    MOVE PRODUCT-ID TO PRODID.
    MOVE STATUS-CODE TO STAT.
    WRITE PRINT-LINE FROM HLINE-3.

0700-STATUS-CHECK SECTION.
0710-STATUS-CHECK-1.
    IF DATABASE-STATUS NOT EQUAL TO ZERO
        DISPLAY "RELATION ERROR " DATABASE-STATUS
        "ON FILE " DB-REALM
    PERFORM 0800-CLOSING.

0800-CLOSING SECTION.
0810-CLOSE-UP.
    CLOSE DPD-REL.
    CLOSE PRINT-FILE.
    STOP RUN.
```

Figure 6-2. COBOL Program (Sheet 2 of 2)

```
DEPARTMENT PACKAGING-DESIGN          MANAGER B. L. CARPENTER
PROJECT NUMBER M130001560            PROJECT ADVANCED BIG BOX
     PRODUCT 537KLPN037                   PRODUCT STATUS N
     PRODUCT 826NAMW019                   PRODUCT STATUS N
PROJECT NUMBER M130001720            PROJECT ADVANCED THERMAL PROTECTOR
     PRODUCT 025CBLE055                   PRODUCT STATUS A
     PRODUCT 432DRTF043                   PRODUCT STATUS A

DEPARTMENT RESEARCH                  MANAGER R. H. FINDER
PROJECT NUMBER M200001570            PROJECT ADVANCED NETWORK
     PRODUCT 537KLPN077                   PRODUCT STATUS A
     PRODUCT 537KLPN078                   PRODUCT STATUS A
     PRODUCT 537KLPN079                   PRODUCT STATUS A
     PRODUCT 567CRTX882                   PRODUCT STATUS N
     PRODUCT 567LINE094                   PRODUCT STATUS N
     PRODUCT 826NAMW069                   PRODUCT STATUS N
     PRODUCT 826NAMW070                   PRODUCT STATUS N
PROJECT NUMBER M200001590            PROJECT ARTIC AGRICULTURE
     PRODUCT 387ARAG322                   PRODUCT STATUS A
     PRODUCT 387ARAG323                   PRODUCT STATUS N

DEPARTMENT DEVELOPMENT               MANAGER L. C. HOWE
PROJECT NUMBER M210001320            PROJECT GAMMA SPOOK SUPPORT
     PRODUCT 693GSPK020                   PRODUCT STATUS A
     PRODUCT 693GSPK022                   PRODUCT STATUS A
PROJECT NUMBER M210001322            PROJECT BETA SUPPORT
     PRODUCT 280BSPK910                   PRODUCT STATUS N
PROJECT NUMBER M210002540            PROJECT EPSILON SPOOK DEVICES
     PRODUCT 466EPSD311                   PRODUCT STATUS A

DEPARTMENT CHEM LAB                  MANAGER I. A. BUNSEN
PROJECT NUMBER M890001550            PROJECT ADVANCED BONDING AGENT-99
     PRODUCT 138CBND926                   PRODUCT STATUS A
     PRODUCT 138CBND930                   PRODUCT STATUS A
     PRODUCT 138CBND940                   PRODUCT STATUS A
```

Figure 6-3. Report Generated by COBOL Program

Examples of FORTRAN 5 application programs that interface with CDCS are included in this section. The examples show the application program along with the subschema it references, the control statements required to compile and execute it, and program output if applicable.

The examples use a manufacturing data base shown in appendix H. This appendix contains all the jobs required to set up the data base environment. The schema should be referenced for access control keys required to access the realms included in the subschemas. The data stored in the data base is shown in the Query Update job that creates the data base files.

FORTRAN 5 EXAMPLE

The FORTRAN 5 example is a program that reads a relation, generates a report, changes data base versions, and performs data base updates through the use of transactions.

The program uses subschema F5SS-PRODUCT-MANAGEMENT shown in figure 7-1. The subschema listing shows a three-realm relation, DPD-REL, which joins realms DEPARTMENTS, PROJECTS, and PRODUCTS. The listing also shows that a restriction has been placed on data item STATUS in realm PRODUCTS. With the restriction, only record occurrences that meet the requirements specified in the restriction (that is, the value of STATUS equals A or R) can be returned to the application program when the relation is read.

The subschema must be available when the FORTRAN 5 program is preprocessed. The control statements that are used to attach the subschema and preprocess, compile, and execute the program are as follows:

```
ATTACH,subschema library ...
DML,LV=F5,SB=subschema library.
FTN5,I=DMLOUT.
LIBRARY(DMSLIB)
LGO.
```

The ATTACH control statement is specified to identify the local file name of the file that contains the subschema. The DML control statement is specified to execute the DML preprocessor. The FTN5 control statement is specified, indicating that DMLOUT (the output file of the DML preprocessor) is the input file to the FORTRAN 5 compiler. The LIBRARY(DMSLIB) control statement is specified to load the DMS-170 library, which is required for program execution. The system default file LGO is specified as the file containing the relocatable binary program being executed.

The FORTRAN 5 program is shown in figure 7-2. A data base status block is declared. The subschema F5SS-PRODUCT-MANAGEMENT is specified. Privacy statements are included to provide the privacy keys; the keys required for all processing performed by the program on a particular realm must be specified before the realm is first accessed. The program must specify privacy keys as follows:

```
VERY*PRIVATE required for any access of realm
DEPARTMENTS

VERIFIED-INPUT required for retrieval from realm
PROJECT

OKAYED-OUTPUT required for update on realm
PROJECT

ACCESS(/)OK required for any access of realm
PRODUCT-FILE
```

The three-level relation DPD-REL is opened for input only. The root realm of the relation is positioned by a START statement. The relation is read sequentially and a report is printed. To print the report, the program tests for control breaks by using array element RELSTAT(2) and for null records by using array element RELSTAT(3).

The program uses control break information to test for the management levels. A department has projects that develop products. The levels appear as follows:

```
Department
    Projects
        Products
```

A control break value of 2 indicates that the program is reading information about a new department. A control break value of 3 indicates that the program is reading information about a new project. The report generated shows all products assigned to a project and all the projects assigned to a department.

The program uses null record information to test for qualifying records; a nonzero value indicates there is no qualifying record. When this occurs, the program suppresses printing and initiates another read.

After the program reads the relation, it closes the files. The program must close the files before it can change to another data base version.

After the program generates the report, the program updates files of version BRANCH1. A NEWVERSION statement changes the data base version to BRANCH1. (Version MASTER was assumed for previous processing.) The ASSIGNID statement obtains a restart identifier. The restart identifier is printed. Two realms, PROJECT and PRODUCT-FILE, are opened for input/output. The BEGINTRAN statement begins a transaction. Within the transaction, records from the PROJECT and PRODUCT-FILE realms are updated. The transaction is terminated by the COMMITTRAN statement.

A listing of the output produced by RELREAD is shown in figure 7-3.

```
F5SS-PRODU                    * SOURCE LISTING *   (82028)  DDLF 1.2+552.

                00001               SUBSCHEMA F5SS-PRODUCT-MANAGEMENT, SCHEMA=MANUFACTURING-DB
                00002
                00003               ALIAS (ITEM) DEPTNO = DEPT-NO
                00004               ALIAS (ITEM) DEPTNAM = DEPT-NAME
                00005               ALIAS (ITEM) MGRID = MGR-ID
                00006               ALIAS (ITEM) MGRNAM = MGR-NAME
                00007               ALIAS (ITEM) PROJID = PROJECT-ID.PROJREC
                00008               ALIAS (ITEM) PROJDES = PROJ-DESCR
                00009               ALIAS (ITEM) BUDGTOT = BUDGET-TOTAL
                00010               ALIAS (REALM) BOSS = RESPONSIBILITY
                00011               ALIAS (REALM) PRODUCT-FILE = DEVELOPMENT-PRODUCTS
                00012               ALIAS (ITEM) PRODUCT = PRODUCT-ID
                00013               ALIAS (ITEM) PRODES = PRODUCT-DESCR
                00014               ALIAS (ITEM) PROJECT = PROJECT-ID.DEVREC
                00015               ALIAS (ITEM) STATUS = STATUS-CODE
                00016               ALIAS (ITEM) YTDCOST = DEV-COST-YTD
                00017
                00018               REALM DEPARTMENTS, PROJECT, PRODUCT-FILE
                00019
                00020               RECORD DEPTREC
** WITHIN DEPARTM
                00021               CHARACTER DEPTNO *4, DEPTNAM *20
** ORDINAL    2
                00022               CHARACTER MGRID *8, MGRNAM *20
                00023
** ORDINAL    4
                00024               RECORD PROJREC
** WITHIN PROJECT
                00025               CHARACTER PROJID *10, PROJDES *40
** ORDINAL    2
                00026               REAL BUDGTOT
                00027
** ORDINAL    3
                00028               RECORD DEVREC
** WITHIN PRODUCT
                00029               CHARACTER PRODUCT *10, PRODES *20
** ORDINAL    2
                00030               CHARACTER PROJECT *10
** ORDINAL    3
                00031               CHARACTER STATUS *1
** ORDINAL    4
                00032               REAL YTDCOST
                00033
** ORDINAL    5
                00034                       RELATION DPD-REL
 PRIMARY KEY    00021          DEPTNO FOR AREA DEPARTMENTS
 ALTERNATE KEY  00022          MGRID FOR AREA DEPARTMENTS
 PRIMARY KEY    00025          PROJID FOR AREA PROJECT
 PRIMARY KEY    00029          PRODUCT FOR AREA PRODUCT-FILE
 ALTERNATE KEY  00030          PROJECT FOR AREA PRODUCT-FILE
                *****          RECORD MAPPING IS NEEDED FOR REALM -   DEPARTMENTS
                *****          RECORD MAPPING IS NEEDED FOR REALM -   PROJECT
                *****          RECORD MAPPING IS NEEDED FOR REALM -   PRODUCT-FILE
                00035                   RESTRICT DEVREC (STATUS .EQ. "A" .OR. STATUS .EQ. "R")
                00036
                00037                   END
                *****          END OF SUB-SCHEMA SOURCE INPUT


                *****                    RELATION   STATISTICS                  *****
RELATION 001                    DPD-REL JOINS           AREA - DEPARTMENTS
                                                        AREA - PROJECT
                                                        AREA - PRODUCT-FILE
```

Figure 7-1. FORTRAN 5 Subschema (Sheet 1 of 2)

```
          -----       BEGIN SUB-SCHEMA FILE MAINTENANCE              -----

        SUBSCHEMA                                  CHECKSUM
        F5SS-PRODUCT-MANAGEMENT                    1002241345672415647O

          -----       END OF FILE MAINTENANCE                        -----
DDLF COMPLETE.          O DIAGNOSTICS.
  507OOB CM USED.      0.331  CP SECS.
```

Figure 7-1. FORTRAN 5 Subschema (Sheet 2 of 2)

Input to DML Preprocessor:

```
      PROGRAM RELREAD
      CHARACTER RSTID * 10
      INTEGER REALM(3),FUNCT,SEVCODE,CRMCODE
      INTEGER STATBLK(11), RELSTAT(3)
      EQUIVALENCE (STATBLK (1),IER),(STATBLK (2),IORD),
     *(STATBLK (3),CRMCODE),(STATBLK (4),SEVCODE),(STATBLK (5),FUNCT),
     *(STATBLK (6),RELSTAT (1)),(STATBLK (9),REALM)

      SUBSCHEMA (F5SS-PRODUCT-MANAGEMENT)

      INVOKE

      CALL DMLDBST(STATBLK,11)

      PRIVACY(DEPARTMENTS,MODE=IO,PRIVACY='VERY*PRIVATE')
      PRIVACY(PROJECT,MODE=I,PRIVACY='VERIFIED-INPUT')
      PRIVACY(PROJECT,MODE=O,PRIVACY='OKAYED-OUTPUT')
      PRIVACY(PRODUCT-FILE,MODE=IO,PRIVACY='ACCESS(/)OK')

C     RELATION READ OPERATION
      OPEN(DPD-REL,MODE=I,ERR=70)

      DEPTNO='M130'
      START(DPD-REL,KEY .EQ. DEPTNO,ERR=70)

      PRINT 89

      DO 15,K=1,15
   10 READ(DPD-REL,ERR=70,END=16)
      IF(RELSTAT(3).NE. 0)THEN
         GO TO 10
      ELSE IF(RELSTAT(2) .EQ. 2)THEN
         PRINT 90,DEPTNAM,MGRNAM,PROJID,PROJDES,PRODUCT,STATUS
      ELSE IF(RELSTAT(2) .EQ. 3)THEN
         PRINT 95,PROJID,PROJDES,PRODUCT,STATUS
      ELSE
         PRINT 100,PRODUCT,STATUS
      ENDIF

   15 CONTINUE

   16 CLOSE(DPD-REL)

C     VERSION CHANGE OPERATION AND
C     UPDATE USING DATA BASE TRANSACTIONS
      NEWVERSION("BRANCH1")

      OPEN(PROJECT,MODE=IO,ERR=70)
      OPEN(PRODUCT-FILE,MODE=IO,ERR=70)

      ASSIGNID(RSTID,ERR=70)
      PRINT *,'RESTART ID IS',RSTID
```

Figure 7-2. FORTRAN 5 Program (Sheet 1 of 4)

```
      BEGINTRAN('MYTRANID1',ERR=70)
      PROJID='M130001560'
      READ(PROJECT,KEY .EQ. PROJID,ERR=70)
      BUDGTOT=BUDGTOT + 100000.00
      REWRITE(PROJECT,ERR=70)
      PRODUCT='826NAMW019'
      READ(PRODUCT-FILE,KEY .EQ. PRODUCT,ERR=70)
      YTDCOST=YTDCOST + 20000.00
      REWRITE(PRODUCT-FILE,ERR=70)
      COMMITTRAN(ERR=70)

      GO TO 75

   70 PRINT 110,IER,IORD,CRMCODE,FUNCT,REALM
      DROPTRAN

   75 CLOSE(PROJECT)
      CLOSE(PRODUCT-FILE)
      TERMINATE

   89 FORMAT(2X,'DEPARTMENT',12X,'MANAGER',15X,'PROJECT NUMBER',4X,
     *'PROJECT DESCRIPTION'/ 1X,107('-')/)

   90 FORMAT(/2X,A20,2X,A20,2X,A10,8X,A40/
     *      73X,'PRODUCT ID = ',A10,'; STATUS = ',A1)

   95 FORMAT(46X,A10,8X,A40/
     *      73X,'PRODUCT ID = ',A10,'; STATUS = ',A1)

  100 FORMAT(73X,'PRODUCT ID = ',A10,'; STATUS = ',A1)

  105 FORMAT(2X,'STATUS BLOCK'/
     *2X,04,2X,I5,2X,03,2X,A10,2X,A30,2X,A30)

  110 FORMAT(2X,'ERROR DURING PROCESSING'/
     *'STATUS BLOCK VALUES ARE'/
     *2X,04,2X,I5,2X,03,2X,A10,2X,A30)

      END
```

FORTRAN 5 Compilation Output Listing:

```
    1                PROGRAM RELREAD
    2                CHARACTER RSTID * 10
    3                INTEGER REALM(3),FUNCT,SEVCODE,CRMCODE
    4                INTEGER STATBLK(11), RELSTAT(3)
    5                EQUIVALENCE (STATBLK (1),IER),(STATBLK (2),IORD),
    6               *(STATBLK (3),CRMCODE),(STATBLK (4),SEVCODE),(STATBLK (5),FUNCT),
    7               *(STATBLK (6),RELSTAT (1)),(STATBLK (9),REALM)
    8          **    SUBSCHEMA (F5SS-PRODUCT-MANAGEMENT)
    9          C$    LIST(ALL=0)
  109          C$    LIST(ALL)
  110          **    INVOKE
  111          C$    LIST(ALL=0)
  125          C$    LIST(ALL)
  126                CALL DMLINV(0003,DBF0001,10HF5SS-PRODU,10HCT-MANAGEM,
  127               +10HENT       ,0"1002241345624156470")
  128                CALL DMLDBST(STATBLK,11)
  129          **    PRIVACY(DEPARTMENTS,MODE=IO,PRIVACY='VERY*PRIVATE')
  130                CALL DMLPRV(1,1,0,0001,
  131               +0"60","VERY*PRIVA","TE        ","          ")
  132          **    PRIVACY(PROJECT,MODE=I,PRIVACY='VERIFIED-INPUT')
  133                CALL DMLPRV(1,1,0,0002,
  134               +0"40","VERIFIED-I","NPUT      ","          ")
  135          **    PRIVACY(PROJECT,MODE=O,PRIVACY='OKAYED-OUTPUT')
  136                CALL DMLPRV(1,1,0,0002,
  137               +0"20","OKAYED-OUT","PUT       ","          ")
  138          **    PRIVACY(PRODUCT-FILE,MODE=IO,PRIVACY='ACCESS(/)OK')
```

Figure 7-2. FORTRAN 5 Program (Sheet 2 of 4)

```
      139           C  RELATION READ OPERATION
      140                CALL DMLPRV(1,1,0,0003,
      141               +0"60","ACCESS(/)0","K          ","          ")
      142           **   OPEN(DPD-REL,MODE=I,ERR=70)
      143                CALL DMLOPNR(DBN0001,DBA0001,2HI ,*70   )
      144                DEPTNO='M130'
      145           **   START(DPD-REL,KEY .EQ. DEPTNO,ERR=70)
      146                CALL DMLRST(DBN0001,0001,00001,0001,1,0004,0,0000,00,DEPTNO ,
      147               + 0001,*70     )
      148                PRINT 89
      149                DO 15,K=1,15
      150           **   READ(DPD-REL,ERR=70,END=16)
      151              10 CALL DMLRL(DBN0001,0001,1,1,*70   ,*16   )
      152                IF(RELSTAT(3).NE. 0)THEN
    1 153                   GO TO 10
    1 154                ELSE IF(RELSTAT(2) .EQ. 2)THEN
    1 155                   PRINT 90,DEPTNAM,MGRNAM,PROJID,PROJDES,PRODUCT,STATUS
    1 156                ELSE IF(RELSTAT(2) .EQ. 3)THEN
    1 157                   PRINT 95,PROJID,PROJDES,PRODUCT,STATUS
    1 158                ELSE
    1 159                   PRINT 100,PRODUCT,STATUS
    1 160                ENDIF
      161              15 CONTINUE
      162           **   CLOSE(DPD-REL)
      163           C  VERSION CHANGE OPERATION AND
      164           C  UPDATE USING DATA BASE TRANSACTIONS
      165              16 CALL DMLCLSR(DBN0001,DBA0001)
      166           **   NEWVERSION("BRANCH1")
      167                CALL DMLVERS("BRANCH1")
      168           **   OPEN(PROJECT,MODE=IO,ERR=70)
      169                CALL DMLOPN(DBF0002,0002,2HIO,*70   )
      170           **   OPEN(PRODUCT-FILE,MODE=IO,ERR=70)
      171                CALL DMLOPN(DBF0003,0003,2HIO,*70   )
      172           **   ASSIGNID(RSTID,ERR=70)
      173                CALL DMLGTID(RSTID  ,*70   )
      174                PRINT *,'RESTART ID IS',RSTID
      175           **   BEGINTRAN('MYTRANID1',ERR=70)
      176                CALL DMLBEG("MYTRANID1 ",*70   )
      177                PROJID='M130001560'
      178           **   READ(PROJECT,KEY .EQ. PROJID,ERR=70)
      179                CALL DMLRDK(DBF0002,0002,00001,0002,1,0010,0,0000,00,
      180               +PROJID ,*70   )
      181                BUDGTOT=BUDGTOT + 100000.00
      182           **   REWRITE(PROJECT,ERR=70)
      183                CALL DMLREW(DBF0002,0,0002,00001,*70   )
      184                PRODUCT='826NAMW019'
      185           **   READ(PRODUCT-FILE,KEY .EQ. PRODUCT,ERR=70)
      186                CALL DMLRDK(DBF0003,0003,00001,0003,1,0010,0,0000,00,
      187               +PRODUCT,*70   )
      188                YTDCOST=YTDCOST + 20000.00
      189           **   REWRITE(PRODUCT-FILE,ERR=70)
      190                CALL DMLREW(DBF0003,0,0003,00001,*70   )
      191           **   COMMITTRAN(ERR=70)
      192                CALL DMLCMT(*70   )
      193                GO TO 75
      194              70 PRINT 110,IER,IORD,CRMCODE,FUNCT,REALM
      195           **   DROPTRAN
      196                CALL DMLDRP
      197           **   CLOSE(PROJECT)
      198              75 CALL DMLCLS(DBF0002,0002)
      199           **   CLOSE(PRODUCT-FILE)
      200                CALL DMLCLS(DBF0003,0003)
      201           **   TERMINATE
      202                CALL DMLEND
      203              89 FORMAT(2X,'DEPARTMENT',12X,'MANAGER',15X,'PROJECT NUMBER',4X,
      204                *'PROJECT DESCRIPTION'/ 1X,107('-')/)
      205              90 FORMAT(/2X,A20,2X,A20,2X,A10,8X,A40/
      206                *      73X,'PRODUCT ID = ',A10,'; STATUS = ',A1)
      207              95 FORMAT(46X,A10,8X,A40/
      208                *      73X,'PRODUCT ID = ',A10,'; STATUS = ',A1)
```

Figure 7-2. FORTRAN 5 Program (Sheet 3 of 4)

```
209              100 FORMAT(73X,'PRODUCT ID = ',A10,'; STATUS = ',A1)
210              105 FORMAT(2X,'STATUS BLOCK'/
211                 *2X,04,2X,I5,2X,03,2X,A10,2X,A30,2X,A30)
212              110 FORMAT(2X,'ERROR DURING PROCESSING'/
213                 *'STATUS BLOCK VALUES ARE'/
214                 *2X,04,2X,I5,2X,03,2X,A10,2X,A30)
215                  END
```

Figure 7-2. FORTRAN 5 Program (Sheet 4 of 4)

```
DEPARTMENT          MANAGER              PROJECT NUMBER    PROJECT DESCRIPTION
-------------------------------------------------------------------------------------------

PACKAGING-DESIGN    B. L. CARPENTER      M130001560        ADVANCED BIG BOX
                                                                 PRODUCT ID = 7684GRD028; STATUS = R
                                         M130001720        ADVANCED THERMAL PROTECTOR
                                                                 PRODUCT ID = 025CBLE055; STATUS = A
                                                                 PRODUCT ID = 432DRTF043; STATUS = A

RESEARCH            R. H. FINDER         M200001570        ADVANCED NETWORK
                                                                 PRODUCT ID = 537KLPN077; STATUS = A
                                                                 PRODUCT ID = 537KLPN078; STATUS = A
                                                                 PRODUCT ID = 537KLPN079; STATUS = A
                                                                 PRODUCT ID = 567CRTX081; STATUS = R
                                                                 PRODUCT ID = 567CRTX881; STATUS = R
                                         M200001590        ARCTIC AGRICULTURE
                                                                 PRODUCT ID = 387ARAG322; STATUS = A
                                                                 PRODUCT ID = 387ARAG555; STATUS = R

DEVELOPMENT         L. C. HOWE           M210001320        GAMMA SPOOK SUPPORT
                                                                 PRODUCT ID = 693GSPK020; STATUS = A
                                                                 PRODUCT ID = 693GSPK022; STATUS = A
                                         M210001322        BETA SUPPORT
                                                                 PRODUCT ID = 280BSPK950; STATUS = R
                                         M210002540        EPSILON SPOOK DEVICES
                                                                 PRODUCT ID = 466EPSD311; STATUS = A

CHEM LAB            I. A. BUNSEN         M890001550        ADVANCED BONDING AGENT-99
                                                                 PRODUCT ID = 138CBND926; STATUS = A
RESTART ID ISEQIDV
```

Figure 7-3. Report Generated by FORTRAN 5 Program

QUERY UPDATE EXAMPLE 8

A Query Update application that accesses a CDCS controlled data base is included in this section. The application uses a manufacturing data base that is shown in appendix H. This appendix contains all the jobs required to set up the data base environment. The schema should be referenced for access control keys required to access the areas included in the subschemas. The data stored in the data base is shown in the Query Update job that creates the data base files.

The Query Update application uses subschema QUPRODMGT shown in figure 8-1. The listing shows a three-area relation, DPD-REL, which joins areas DEPTAREA, PROJECT, and PRODAREA. A restriction has been placed on data item STATUS-CODE in record PRODREC (area PRODAREA). With the restriction, only record occurrences that meet the requirement specified in the restriction (that is, the value of STATUS-CODE must equal A) can be returned to the user when the relation is queried.

The Query Update application is shown in figure 8-2. Notations in the figure indicate the directives used for the application.

Query Update is called by the QU control statement. The default transmission length (TL parameter) of 1030 characters is used. This length corresponds to the maximum data length specified for QUCAT-ITEM described in the subschema for the catalog file.

The INVOKE directive establishes the CDCS data base access mode and the use of subschema QUPRODMGT. The figure shows use on NOS with the specification of UN=CDCS23. For use on NOS/BE, the specification must be in the form ID=id.

The ACCESS directive provides the access keys. The access keys required to read and update the areas are as follows:

VERY*PRIVATE required for any access of area DEPTAREA

VERIFIED-INPUT required for retrieval from area PROJECT

OKAYED-OUTPUT required for update on area PROJECT

ACCESS(/)OK required for any access of area PRODAREA

The DISPLAY directive causes information to be displayed from each of the three areas in relation DPD-REL. Relation processing is automatically performed by Query Update because of the data items specified in the directive. The restriction on data item STATUS-CODE is in effect; the only record occurrences returned from PRODREC are those in which the value STATUS-CODE equals A.

The MODIFY directive causes three records to be updated. The records are identified by value of the primary key, PROJECT-ID OF PROJREC. The total budget of each record is increased by 10000.

The STORE directive causes two records to be stored in area PROJECT. Records are identified by value of the primary key PROJECT-ID of PROJREC. PROJECT-ID must be qualified because it is not a unique data name in the subschema.

The UNIVERSAL directive causes the symbol # to be used as the universal character for the next directive.

The IF-REMOVE directive combination identifies records by the first three characters of the primary key. Three qualifying records are removed.

The VERSION directive establishes CDCS catalog mode. The ACCESS directive specifies the access key for the catalog file and must be entered before any access to the file is attempted.

The EXHIBIT directive causes the indicated session to be listed. Session DUMMY is an initial session copied to the file to make the file usable in CDCS catalog mode.

```
                                * SOURCE LISTING *   (82029)  DDL  3.2+552.

            00001               TITLE DIVISION.
            00002                   SS  QUPRODMGT WITHIN MANUFACTURING-DB.
            00003
            00004               ALIAS DIVISION.
            00005                   AD  REALM DEVELOPMENT-PRODUCTS BECOMES PRODAREA.
            00006                   AD  RECORD DEVREC BECOMES PRODREC.
            00007                   AD  DATA CUM-TEST-AVERAGE BECOMES CUMULATIVE-AVERAGE.
            00008                   AD  REALM DEPARTMENTS BECOMES DEPTAREA.
            00009
            00010               REALM DIVISION.
            00011                   RD  DEPTAREA, PROJECT, PRODAREA, CDCSCAT.
            00012
            00013               RECORD DIVISION.
            00014               01  DEPTREC.                                        ** WITHIN DEPTARE
            00015                   03  DEPT-NO              PICTURE X(4).          ** ORDINAL    1
            00016                   03  DEPT-NAME            PICTURE X(20).         ** ORDINAL    2
            00017                   03  MGR-ID               PICTURE X(8).          ** ORDINAL    3
            00018                   03  MGR-NAME             PICTURE X(20).         ** ORDINAL    4
            00019
            00020               01  PROJREC.                                        ** WITHIN PROJECT
            00021                   03  PROJECT-ID           PICTURE X(10).         ** ORDINAL    1
            00022                   03  PROJ-DESCR           PICTURE X(40).         ** ORDINAL    2
            00023                   03  BUDGET-TOTAL         PICTURE Z(9).99.       ** ORDINAL    3
            00024                   03  RESPONSIBILITY       PICTURE X(8).          ** ORDINAL    4
            00025
            00026               01  PRODREC.                                        ** WITHIN PRODARE
            00027                   03  PRODUCT-ID           PICTURE X(10).         ** ORDINAL    1
            00028                   03  CLASS                PICTURE Z9.            ** ORDINAL    2
            00029                   03  PRICE                PICTURE Z(5).99        ** ORDINAL    3
            00030                                                 USAGE IS COMP-1.
            00031                   03  PROJECT-ID           PICTURE X(10).         ** ORDINAL    4
            00032                   03  STATUS-CODE          PICTURE A.             ** ORDINAL    5
            00033                   03  DEV-COST-YTD         PICTURE Z(8)9.99       ** ORDINAL    6
            00034                                                 USAGE IS COMP-1.
            00035
            00036               01  QUCATREC.                                       ** WITHIN CDCSCAT
            00037                   03 QUCAT-KEY              PICTURE X(10).         ** ORDINAL    1
            00038                   03 QUCAT-ITEM             PICTURE X(1030).       ** ORDINAL    2
            00039
PRIMARY KEY 00015       DEPT-NO FOR AREA DEPTAREA
ALTERNATE KEY 00017     MGR-ID FOR AREA DEPTAREA
PRIMARY KEY 00021       PROJECT-ID FOR AREA PROJECT
ALTERNATE KEY 00024     RESPONSIBILITY FOR AREA PROJECT
PRIMARY KEY 00027       PRODUCT-ID FOR AREA PRODAREA
ALTERNATE KEY 00031     PROJECT-ID FOR AREA PRODAREA
PRIMARY KEY 00037       QUCAT-KEY FOR AREA CDCSCAT
            *****       RECORD MAPPING IS NEEDED FOR REALM -   DEPTAREA
            *****       RECORD MAPPING IS NEEDED FOR REALM -   PROJECT
            *****       RECORD MAPPING IS NEEDED FOR REALM -   PRODAREA
            *****       RECORD MAPPING IS NOT NEEDED FOR REALM -   CDCSCAT
            00040               RELATION DIVISION.
            00041                   RN IS DPD-REL
            00042                       RESTRICT PRODREC WHERE STATUS-CODE EQ "A".
            00043
            *****       END OF SUB-SCHEMA SOURCE INPUT

            *****               RELATION   STATISTICS                  *****
RELATION 001            DPD-REL JOINS           AREA - DEPTAREA
                                                AREA - PROJECT
                                                AREA - PRODAREA


            -----       BEGIN SUB-SCHEMA FILE MAINTENANCE              -----

        SUBSCHEMA                               CHECKSUM
        QUPRODMGT                               3470360761510606200l

            -----       END OF FILE MAINTENANCE                -----
DDL COMPLETE.          0 DIAGNOSTICS.
  50200B CM USED.     0.337  CP SECS.
```

Figure 8-1. Query Update Subschema

Statements/Directives	Query Update Application

```
QU Control Statement        /QU

                            QUERY UPDATE 3.4 I3A-82064    82/03/16  09.59.57
                             --
INVOKE (CDCS Data Base      ? INVOKE QUPRODMGT FROM LIBRARY QUSSLIB (UN=CDCS23)
        Access Mode)         --

ACCESS (Areas)              ? ACCESS $VERY*PRIVATE$ FOR AREA DEPTAREA
                             --
                            ? ACCESS $ACCESS(/)OK$ FOR AREA PRODAREA
                             --
                            ? ACCESS
                             >>>
                            ? $VERIFIED-INPUT$ ON INPUT FOR AREA PROJECT
                             >>>
                            ? $OKAYED-OUTPUT$ ON OUTPUT FOR AREA PROJECT
                             >>>
                            ? *END
                                    *END
                             --
DISPLAY                     ? DISPLAY DEPT-NAME,MGR-NAME,PROJ-DESCR,$ PRODUCT $,PRODUCT-ID, +
                            ? $ BUDGET = $,BUDGET-TOTAL

                             PURCHASING           C. B. BYERS
                                        PRODUCT              BUDGET =           .00
                             PACKAGING-DESIGN     B. L. CARPENTER      ADVANCED BIG BOX
                                        PRODUCT              BUDGET =     143000.00
                             PACKAGING-DESIGN     B. L. CARPENTER      ADVANCED THERMAL PROTECTOR
                                        PRODUCT  025CBLE055 BUDGET =     105000.00
                             PACKAGING-DESIGN     B. L. CARPENTER      ADVANCED THERMAL PROTECTOR
                                        PRODUCT  432DRTF043 BUDGET =     105000.00
                             RESEARCH             R. H. FINDER         ADVANCED NETWORK
                                        PRODUCT  537KLPN077 BUDGET =     275000.00
                             RESEARCH             R. H. FINDER         ADVANCED NETWORK
                                        PRODUCT  537KLPN078 BUDGET =     275000.00
                             RESEARCH             R. H. FINDER         ADVANCED NETWORK
                                        PRODUCT  537KLPN079 BUDGET =     275000.00
                             (MORE... ANSWER Y OR N)
                            ? Y
                             RESEARCH             R. H. FINDER         ARCTIC AGRICULTURE
                                        PRODUCT  387ARAG322 BUDGET =     192500.00
                             DEVELOPMENT          L. C. HOWE           GAMMA SPOOK SUPPORT
                                        PRODUCT  693GSPK020 BUDGET =      55000.00
                             DEVELOPMENT          L. C. HOWE           GAMMA SPOOK SUPPORT
                                        PRODUCT  693GSPK022 BUDGET =      55000.00
                             DEVELOPMENT          L. C. HOWE           BETA SUPPORT
                                        PRODUCT              BUDGET =      85000.00
                             DEVELOPMENT          L. C. HOWE           EPSILON SPOOK DEVICES
                                        PRODUCT  466EPSD311 BUDGET =      95000.00
                             TESTING-EVALUATION   X. P. MENTOR
                                        PRODUCT              BUDGET =          .00
                             QUALITY CONTROL      S. A. GOODE
                                        PRODUCT              BUDGET =          .00
                             (MORE... ANSWER Y OR N)
                            ? Y
                             CHEM LAB             I. A. BUNSEN         ADVANCED BONDING AGENT-99
                                        PRODUCT  138CBND926 BUDGET =     130000.00
                             CHEM LAB             I. A. BUNSEN         ADVANCED BONDING AGENT-99
                                        PRODUCT  138CBND930 BUDGET =     130000.00
                             CHEM LAB             I. A. BUNSEN         ADVANCED BONDING AGENT-99
                                        PRODUCT  138CBND940 BUDGET =     130000.00
                             CHEM LAB             I. A. BUNSEN         GENETIC ENGINEERING: NITROGE
                             N FIXING   PRODUCT              BUDGET =     258000.00
                                   18 ACCESSES, 18 HITS, 18 IO-S
                             --
```

Figure 8-2. Query Update Application (Sheet 1 of 2)

Statements/Directives	Query Update Application
MODIFY	? MODIFY USING PROJECT-ID OF PROJREC MOVE + ? BUDGET-TOTAL + 10000 TO BUDGET-TOTAL >> ? $M2100013205 >> ? $M2100013225 >> ? $M2100025405 >> *END 3 ACCESSES, 3 HITS, 6 IO-S --
STORE	? STORE SETTING PROJECT-ID OF PROJREC, PROJ-DESCR,RESPONSIBILITY >> ? $M6800082605 $OMEGA VARIENCE$ $5730GOOD$ >> ? $M6800082905 $THERMAL TRANSFER$ $5730GOOD$ >> ? *END 2 ACCESSES, 2 HITS, 2 IO-S --
UNIVERSAL	? UNIVERSAL IS # --
IF ... REMOVE	? IF PRODUCT-ID EQ $138#$ REMOVE PRODREC 26 ACCESSES, 3 HITS, 29 IO-S --
VERSION (CDCS Catalog Mode)	? VERSION IS CDCSCAT OF QUPRODMGT LIBRARY QUSSLIB (UN=CDCS23) --
ACCESS (CDCS Catalog File)	? ACCESS $PERMISSION*GRANTED$ FOR CATALOG CDCSCAT --
EXHIBIT (Session)	? EXHIBIT SESSION DUMMY 1 DISPLAY DUMMY -- ? END

Figure 8-2 Query Update Application (Sheet 2 of 2)

STANDARD CHARACTER SETS A

Control Data operating systems offer the following variations of a basic character set:

CDC 64-character set

CDC 63-character set

ASCII 64-character set

ASCII 63-character set

The set in use at a particular installation was specified when the operating system was installed.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use). Under NOS/BE, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect through the end of the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Under NOS, the alternate mode can be specified also by a 26 or 29 punched in columns 79 and 80 of any 6/7/9 card, as described above for a 7/8/9 card. In addition, 026 mode can be specified by a card with 5/7/9 multipunched in column 1, and 029 mode can be specified by a card with 5/7/9 multipunched in column 1 and a 9 punched in column 2.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of the standard character set table (table A-1) are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

Standard collating sequences for the two printer character sets are shown in tables A-2 and A-3.

TABLE A-1. STANDARD CHARACTER SETS

Display Code (octal)	CDC			ASCII		
	Graphic	Hollerith Punch (026)	External BCD Code	Graphic Subset	Punch (029)	Code (octal)
00†	: (colon)††	8-2	00	: (colon)††	8-2	072
01	A	12-1	61	A	12-1	101
02	B	12-2	62	B	12-2	102
03	C	12-3	63	C	12-3	103
04	D	12-4	64	D	12-4	104
05	E	12-5	65	E	12-5	105
06	F	12-6	66	F	12-6	106
07	G	12-7	67	G	12-7	107
10	H	12-8	70	H	12-8	110
11	I	12-9	71	I	12-9	111
12	J	11-1	41	J	11-1	112
13	K	11-2	42	K	11-2	113
14	L	11-3	43	L	11-3	114
15	M	11-4	44	M	11-4	115
16	N	11-5	45	N	11-5	116
17	O	11-6	46	O	11-6	117
20	P	11-7	47	P	11-7	120
21	Q	11-8	50	Q	11-8	121
22	R	11-9	51	R	11-9	122
23	S	0-2	22	S	0-2	123
24	T	0-3	23	T	0-3	124
25	U	0-4	24	U	0-4	125
26	V	0-5	25	V	0-5	126
27	W	0-6	26	W	0-6	127
30	X	0-7	27	X	0-7	130
31	Y	0-8	30	Y	0-8	131
32	Z	0-9	31	Z	0-9	132
33	0	0	12	0	0	060
34	1	1	01	1	1	061
35	2	2	02	2	2	062
36	3	3	03	3	3	063
37	4	4	04	4	4	064
40	5	5	05	5	5	065
41	6	6	06	6	6	066
42	7	7	07	7	7	067
43	8	8	10	8	8	070
44	9	9	11	9	9	071
45	+	12	60	+	12-8-6	053
46	-	11	40	-	11	055
47	*	11-8-4	54	*	11-8-4	052
50	/	0-1	21	/	0-1	057
51	(	0-8-4	34	(	12-8-5	050
52	)	12-8-4	74	)	11-8-5	051
53	$	11-8-3	53	$	11-8-3	044
54	=	8-3	13	=	8-6	075
55	blank	no punch	20	blank	no punch	040
56	, (comma)	0-8-3	33	, (comma)	0-8-3	054
57	. (period)	12-8-3	73	. (period)	12-8-3	056
60	≡	0-8-6	36	#	8-3	043
61	[	8-7	17	[	12-8-2	133
62	]	0-8-2	32	]	11-8-2	135
63	%††	8-6	16	%††	0-8-4	045
64	≠	8-4	14	" (quote)	8-7	042
65	←	0-8-5	35	_ (underline)	0-8-5	137
66	∨	11-0	52	!	12-8-7	041
67	∧	0-8-7	37	&	12	046
70	↑	11-8-5	55	' (apostrophe)	8-5	047
71	↓	11-8-6	56	?	0-8-7	077
72	<	12-0	72	<	12-8-4	074
73	>	11-8-7	57	>	0-8-6	076
74	≤	8-5	15	@	8-4	100
75	≥	12-8-5	75	\	0-8-2	134
76	¬	12-8-6	76	^ (circumflex)	11-8-7	136
77	; (semicolon)	12-8-7	77	; (semicolon)	11-8-6	073

†Twelve zero bits at the end of a 60-bit word in a zero byte record are an end of record mark rather than two colons.

††In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations yield a blank (55_8).

TABLE A-2. CDC CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal	Collating Sequence Octal	CDC Graphic	Display Code	External BCD	Collating Sequence Decimal	Collating Sequence Octal	CDC Graphic	Display Code	External BCD
00	00	blank	55	20	32	40	H	10	70
01	01	≤	74	15	33	41	I	11	71
02	02	%	63 †	16 †	34	42	∨	66	52
03	03	[	61	17	35	43	J	12	41
04	04	→	65	35	36	44	K	13	42
05	05	≡	60	36	37	45	L	14	43
06	06	∧	67	37	38	46	M	15	44
07	07	↑	70	55	39	47	N	16	45
08	10	↓	71	56	40	50	O	17	46
09	11	>	73	57	41	51	P	20	47
10	12	≥	75	75	42	52	Q	21	50
11	13	¬	76	76	43	53	R	22	51
12	14	.	57	73	44	54	]	62	32
13	15	)	52	74	45	55	S	23	22
14	16	;	77	77	46	56	T	24	23
15	17	+	45	60	47	57	U	25	24
16	20	$	53	53	48	60	V	26	25
17	21	*	47	54	49	61	W	27	26
18	22	-	46	40	50	62	X	30	27
19	23	/	50	21	51	63	Y	31	30
20	24	,	56	33	52	64	Z	32	31
21	25	(	51	34	53	65	:	00 †	none†
22	26	=	54	13	54	66	0	33	12
23	27	≠	64	14	55	67	1	34	01
24	30	<	72	72	56	70	2	35	02
25	31	A	01	61	57	71	3	36	03
26	32	B	02	62	58	72	4	37	04
27	33	C	03	63	59	73	5	40	05
28	34	D	04	64	60	74	6	41	06
29	35	E	05	65	61	75	7	42	07
30	36	F	06	66	62	76	8	43	10
31	37	G	07	67	63	77	9	44	11

†In installations using the 63-graphic set, the % graphic does not exist. The : graphic is display code 63, External BCD code 16.

TABLE A-3. ASCII CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal	Collating Sequence Octal	ASCII Graphic Subset	Display Code	ASCII Code	Collating Sequence Decimal	Collating Sequence Octal	ASCII Graphic Subset	Display Code	ASCII Code
00	00	blank	55	20	32	40	@	74	40
01	01	!	66	21	33	41	A	01	41
02	02	"	64	22	34	42	B	02	42
03	03	#	60	23	35	43	C	03	43
04	04	$	53	24	36	44	D	04	44
05	05	%	63†	25	37	45	E	05	45
06	06	&	67	26	38	46	F	06	46
07	07	'	70	27	39	47	G	07	47
08	10	(	51	28	40	50	H	10	48
09	11	)	52	29	41	51	I	11	49
10	12	*	47	2A	42	52	J	12	4A
11	13	+	45	2B	43	53	K	13	4B
12	14	,	56	2C	44	54	L	14	4C
13	15	-	46	2D	45	55	M	15	4D
14	16	.	57	2E	46	56	N	16	4E
15	17	/	50	2F	47	57	O	17	4F
16	20	0	33	30	48	60	P	20	50
17	21	1	34	31	49	61	Q	21	51
18	22	2	35	32	50	62	R	22	52
19	23	3	36	33	51	63	S	23	53
20	24	4	37	34	52	64	T	24	54
21	25	5	40	35	53	65	U	25	55
22	26	6	41	36	54	66	V	26	56
23	27	7	42	37	55	67	W	27	57
24	30	8	43	38	56	70	X	30	58
25	31	9	44	39	57	71	Y	31	59
26	32	:	00†	3A	58	72	Z	32	5A
27	33	;	77	3B	59	73	[	61	5B
28	34	<	72	3C	60	74	\	75	5C
29	35	=	54	3D	61	75	]	62	5D
30	36	>	73	3E	62	76	^	76	5E
31	37	?	71	3F	63	77	_	65	5F

†In installations using a 63-graphic set, the % graphic does not exist. The : graphic is display code 63.

DIAGNOSTIC MESSAGES B

Both the execution time diagnostic messages issued by CYBER Database Control System (CDCS), and the diagnostic messages issued by the FORTRAN Data Manipulation Language (DML) preprocessor are listed in this appendix.

Messages in this appendix are listed in the following order:

Messages beginning with an alphabetic character are listed first.

Messages beginning with a number are listed next.

Messages beginning with a variable are listed last.

A variable appearing in a message signifies that the field is replaced by applicable text when the diagnostic is issued.

CDCS DIAGNOSTIC MESSAGES

All diagnostic messages that can be issued by CDCS are listed in table B-1. The general significance of the message and the action to be taken by the user accompany each message listed in the table. A destination column designates the dayfile or output file to which the message is written.

The severity level of the diagnostic messages listed in table B-1 can be of several types. The types of messages and their meanings are as follows:

C Critical error. The system control point is aborted and all user control point jobs are aborted.

F Fatal error. The run-unit is aborted, with the execption of a FORTRAN or COBOL program executing with the immediate return feature enabled, an application program executing through the Transaction Facility (TAF), or a Query Update application executing in interactive mode. (In these cases the program is terminated from CDCS processing. Control is returned to the program if the immediate return feature is enabled. Control is returned to TAF if the program is processing through TAF or to Query Update if the application is executing in interactive mode.)

N Nonfatal error. Processing continues.

I Informative message. Processing continues.

The CDCS diagnostic messages listed in table B-1 are categorized several ways. The following paragraphs describe the messages in categories that describe the destination of the message and the type of message sent to that destination.

The first category of CDCS diagnostics listed in table B-1 includes the unnumbered messages that are sent to the CDCS dayfile. These messages are of the following types:

C Critical error

N Nonfatal error

I Informative message

The second category of diagnostics includes the unnumbered messages issued by CDCS when it is operating as the CDCS Batch Test Facility (CDCSBTF). These messages are sent to the CDCSBTF control point dayfile, which is the user control point dayfile since CDCSBTF executes at the user control point. These messages are of the following types:

F Fatal error

I Informative message

The third category of CDCS messages includes the unnumbered messages that are sent to the user control point dayfile and the CDCS output file. These messages can be of the following types:

F Fatal error

I Informative message

The fourth category of CDCS messages include the unnumered messages that are sent to both the CDCS control point dayfile and the CDCS output file. These messages are usually issued during system recovery processing. They can also be issued at other times (for example, during invoke processing or when a transaction is reversed). These messages can be of the following types:

N Nonfatal error

I Informative message

The fifth category of CDCS messages includes the unnumbered messages that are sent to the user control point dayfile only. These messages can be of the following types:

F Fatal error

N Nonfatal error

I Informative message

The sixth category of messages includes the unnumbered messages that are sent to the CDCS output file only. These error messages are usually accompanied by a fatal error message sent to the CDCS system control point dayfile. These messages can be of the following type:

I Informative message

The seventh category of CDCS message includes the messages that occur when the system operator communicates with CDCS through the K (NOS) or the L (NOS/BE) display. These informative messages that indicate activities triggered by operator commands are sent to the CDCS system control point dayfile. In some cases, the message returned is an echo of the command entered by the operator. These messages are of the following type:

I Informative message

The last category of CDCS diagnostic messages includes the CDCS numbered execution-time messages for user errors. Each octally numbered message is accompanied by a conversion from the octal value of the CDCS error code to the equivalent decimal value. The decimal conversion of the error code (found in the column headed Dec. E. C. of table B-1) can be of assistance to a COBOL programmer using the data base status block to obtain error and status information because codes returned in the status block of a COBOL program must be referenced by their decimal values. The error and status codes returned in the status block of a FORTRAN program can be referenced by either their decimal or octal values. The octal value of codes are returned by Query Update in a diagnostic message. The destination of each message depends on the type. These messages can be of the following types:

F Fatal error. The message is sent to the CDCS output file, the data base status block if one exists, and the user control point dayfile. A message is sent to the ZZZZZEG file if the message is associated with a realm (area).

N Nonfatal error. The message is sent to the CDCS output file and the data base status block if one exists. The message is sent to the ZZZZZEG file if the message is associated with a realm (area).

T Trivial error. The message is sent to the data base status block if one exists. The message is also sent to the ZZZZZEG file if the message is associated with a realm (area).

I Informative message. The message is sent to the CDCS output file. The message is sent to the ZZZZZEG file if the message is associated with a realm (area).

Messages written by CDCS to the user's ZZZZZEG error file are known as data manager messages. When the ZZZZZEG file is processed, each data manager error message is preceded by the following information:

DM ERROR ec ON LFN fn EXIT ADDRESS ad

The fields ec, fn, and ad are replaced by applicable text to designate error code, schema file name, and address, respectively.

TABLE B-1. CDCS DIAGNOSTICS

Type	Dec. E.C.	Message	Significance	Action	Destination
N		ALL AREAS WERE DOWN, SCHEMA DOWN	None of the areas in the schema were in up status at the completion of system recovery. The schema is set to down status.	Correct the errors causing the areas to be down. Manually recover the areas in the schema.	CDCS output file CDCS system control point dayfile
F		AREA id VERSION vn NOT UP. SCHEMA sn	The area id of version vn (version MASTER if vn does not appear) was not in an up, idling, or downing status during processing for system recovery or CDCS transaction reversal. Schema sn is printed out for nonsystem recovery only.	If this error occurred during system recovery, manually recover the area. If transaction processing was in effect, manually reverse the uncommitted updates. Place the area in up status.	CDCS output file CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
I		AREA RECOVERY IS COMPLETE. THE AREA IS LEFT IN DOWN STATUS. TO REGAIN USE OF THE AREA USE THE -UP- COMMAND.	The database area that suffered from an internal malfunction has been recovered. It has been left in a "DOWN" status to insure that the occurrence of the malfunction is noticed.	The console operator may change the area to "UP" status.	CDCS system control point dayfile
F		ATTACH ERROR nnn ON AREA id VERSION vn - AREA DOWN. SCHEMA sn	The attach error nnn occurred during system recovery processing or invoke processing on area id of version vn (version MASTER if vn does not appear). Schema sn is printed out for nonsystem recovery only.	If this error occurred during system recovery, manually recover the area. Otherwise, check that file is not attached elsewhere. For a non-PUBLIC file unavailable on NOS, ensure that a PERMIT control statement specifying the user name for CDCS or CDCSBTF execution is in effect or that the procedure to initialize CDCS includes a USER control statement specifying the user name for CDCS execution. Refer to the applicable permanent file error diagnostic in volume 4 of the NOS reference set, or in the description of the FDB macro in the NOS/BE reference manual.	CDCS output file CDCS system control point dayfile.
F/N		ATTACH ERROR nnn ON CDCS file	The operating system attach error nnn occurred during system recovery or invoke processing on one of the following CDCS files: the JOURNAL LOG FILE, the PROCEDURE LIB FILE, the QUICK RECOVERY FILE, the RESTART IDENTIFIER FILE, or the TRANSACTION RECOVERY FILE. The schema is set to error status unless the file is busy.	If this error occurred during system recovery, manually recover the areas in the schema before placing the schema in up status. If this error occurred during invoke processing, resubmit the application program. Otherwise, check that file is not attached elsewhere. For a non-PUBLIC file unavailable on NOS, ensure that a PERMIT control statement specifying the user name for CDCS or CDCSBTF execution is in effect or that the procedure to initialize CDCS includes a USER control statement specifying the user name for CDCS execution. Refer to the applicable permanent file error diagnostic in volume 4 of the NOS reference set, or in the description of the FDB macro in the NOS/BE reference manual.	CDCS output file CDCS system control point dayfile
I		BL SET TO bbbbbb	In the absence of a BL parameter a value of bbbbbb has been assigned.	None	CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
N		BOTH VSN AND SET NAME MUST BE GIVEN	In either the CDCS control statement or the directive file, if VSN is specified set name must be specified. If set name is specified VSN must be specified.	Specify both VSN and set name in the CDCS control statement or the directive file.	CDCS system control point dayfile
C		CANNOT GET num WORDS CM FOR jn	The 6-digit number indicates the number of words of memory that cannot be obtained to process the run-unit request. Under NOS/BE, jn is the 7-character job name; under NOS, jn is the 4-character sequence number of the system-assigned job name.	If several messages of this type appear, the data administrator should adjust the CDCS field length. If only one job is affected, rerun the job.	CDCS system control point dayfile
C		CDCS ABORT	Errors occurred while processing the CDCS control statement or the CDCS directive file.	Correct the control statement or the directive file.	CDCS system control point dayfile
C		CDCS ABORT, INVALID MASTER DIRECTORY	A master directory created and maintained by a previous version of CDCS is not compatible with CDCS 2.3.	Recreate the master directory by reintroducing all previous specifications and by using recompiled schemas and subschemas. Schemas and subschemas must be compiled with DDL 3.1 or DDL 3.2 for use with CDCS 2.3.	CDCS system control point dayfile
C		CDCS ABORT, NO SCHEMA ENTRIES IN MASTER DIRECTORY	Information for at least one schema must be included in master directory.	Check that information for the schema has not been deleted from the master directory. If deleted, add information for the schema in a master directory run.	CDCS system control point dayfile
C		CDCS ABORTED-- RECOVERY FOR ALL SCHEMAS IMPOSSIBLE	All schemas are in error status after the CDCS system recovery phase.	Correct the schema errors and reinitiate CDCS execution.	CDCS system control point dayfile
I		CDCS CHARGED xxxxxx.xxx CP yyyyyy.yyy IO	CDCS accounting charged xxxxxx.xxx seconds of central processor time and yyyyyy.yyy seconds of I/O time to user jobs.	None.	CDCS system control point dayfile
I		CDCS DOWN COMPLETE	CDCS has been shut down by the system operator. No new invokes to CDCS are allowed.	None.	CDCS system control point dayfile
I		CDCS ERRORS	The CDCS control statement or the directive file contains errors. This message is followed by a list of the specific errors.	Correct the control statement or the directive file errors. Reinitialize CDCS.	CDCS system control point dayfile
F		CDCS FUNCTION CODE UNKNOWN DB$$SIM 1	This is an internal error.	Follow site-defined procedures for reporting software errors or operational problems.	CDCSBTF control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
I		CDCS IDLE COMPLETE	CDCS has been shut down by the system operator. No new invokes to CDCS are allowed.	None.	CDCS system control point dayfile
F		CDCS ILLEGAL REQUEST, DOUBLE INVOKE	More than one invoke was issued by the COBOL program. This is an internal error.	Follow site-defined procedures for reporting software errors or operational problems.	User control point dayfile CDCS output file
F		CDCS ILLEGAL REQUEST, ILLEGAL FUNCTION CODE	CDCS cannot process the request issued because it is an unrecognizable function code. This is an internal error.	Follow site-defined procedures for reporting software errors or operational problems.	User control point dayfile CDCS output file
F		CDCS ILLEGAL REQUEST, NOT INVOKED	CDCS cannot process the user program request because CDCS invocation has not yet occurred. This is an internal error.	Follow site-defined procedures for reporting software errors or operational problems.	User control point dayfile CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
F		CDCS ILLEGAL REQUEST, TWO OUTSTANDING REQUESTS	CDCS received another request from a user program before the previous request was satisfied. This is an internal error.	Follow site-defined procedures for reporting software errors or operational problems.	User control point dayfile CDCS output file
I		CDCS INITIALIZATION COMPLETE	CDCS is available at a system control point and is ready to start accepting requests from user programs.	None.	CDCS system control point dayfile
I		CDCS INTERFACE TERMINATED	CDCS termination processing has occurred.	None.	User control point dayfile CDCS output file
C		CDCS INTERNAL ERROR -- mdn	A CDCS internal error occurred in mdn; the first seven characters are the module name and the last three characters designate the location in the module where the error occurred.	Follow site-defined procedures for reporting software errors or operational problems.	CDCS system control point dayfile
I		CDCS INVOKED BY user-id	CDCS invocation has occurred by the job identified by user-id. The user-id was passed when CDCS was invoked.	None.	User control point dayfile CDCS output file
I		CDCS JOB ABORT OR ENDED BY SYSTEM	CDCS or the operating system aborted the job, or the job terminated without CDCS termination processing.	Determine cause of abort from other messages issued to the dayfiles.	User control point dayfile CDCS output file
N		CDCS JOURNAL LOG FILE NOT AVAILABLE	No journal log file in new or in use status can be found during system recovery or invoke processing. The schema is set to error status.	Analyze the reason for the journal log file being unavailable. If this error occurred during system recovery, manually recover the areas in the schema.	CDCS output file CDCS system control point dayfile
F		CDCS NOT ACTIVE AT SYSTEM CONTROL POINT	A job issued a request to CDCS when CDCS was not active at a system control point.	Check that CDCS is active at a system control point before resubmitting the application program.	User control point dayfile
I		CDCS RECOVERY COMPLETED	The CDCS system recovery phase is complete.	None.	CDCS system control point dayfile
I		CDCS RECOVERY STARTED	The CDCS system recovery phase is started.	None.	CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
I		CDCS REPRIEVE PROCESSING DONE	CDCS processing for the abnormal termination of a job has been completed. All files attached by CDCS have been returned.	None.	CDCS system control point dayfile CDCS output file
F		CDCS REQUEST NOT IN FL	The job is aborted; the request cannot be processed by CDCS because of an invalid central memory address in the request.	Follow site-defined procedures for reporting software errors or operational problems.	User control point dayfile CDCS output file
I		CDCS SCP STATUS TERMINATED	CDCS has been terminated and is no longer at a system control point.	None.	CDCS system control point dayfile
I		CDCS USED aaaaaa.aaa CP SECONDS bbbbbb.bbb IO SECONDS	CDCS actually used aaaaaa.aaa central processor seconds and bbbbbb.bbb I/O seconds.	None.	CDCS system control point dayfile
I		CDCS xx.yy PERCENT CPU USAGE	CDCS CPU utilization is defined as the ratio of the total CP time used to the real time elapsed while CDCS was active (that is, from the time CDCS was initialized until the time CDCS was terminated).	None.	CDCS system control point dayfile
F		CDCSBTF ABORTED, 2 TRANSFER ADDRESSES ON USER LOAD FILE fn	The user's load file fn has multiple transfer addresses because two main programs are in the same file.	Change the load file so that it has only one main program.	CDCSBTF control point dayfile
F		CDCSBTF DEADLOCK, JOB ABORTED	CDCSBTF has determined that the user job and CDCS are both waiting for a particular event to occur, but that event is not occurring. CDCSBTF is aborted.	Correct the deadlock and resubmit the job.	CDCSBTF control point dayfile
I		CDCSBTF TERMINATED BEFORE ALL USERS COMPLETED	A CDCSBTF user program has placed "END" in RA+1 without referencing the external procedure SYS=.	Review each user program to be sure that it uses the normal termination statement provided by the language. For COMPASS, use the ENDRUN macro.	CDCSBTF control point dayfile
F		CDCSBTF TERMINATED, LOADER ERROR nnnnn ON USER LOAD FILE fn	A fatal loader error nnnnn occurred during the loading of user file fn.	Refer to the error message in the CYBER Loader reference manual for the appropriate action to correct the error.	CDCSBTF control point dayfile
F		CDCSBTF TERMINATED, NO PROGRAMS	No file name parameter was specified in the CDCSBTF control statement.	Specify the file name of the relocatable binary program as the parameter on the CDCSBTF control statement and resubmit the job.	CDCSBTF control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
N		CIO ERROR nnn DURING ROLLOUT	CIO encountered error nnn while rolling out part of the CDCS field length.	Follow site-defined procedures for reporting software errors or operational problems.	CDCS system control point dayfile
I		CIO ERROR nnn ON CDCS JOURNAL LOG FILE-SCHEMA sn	CIO encountered error number nnn while performing an operation on the CDCS journal log file for the schema named sn. Some or all of the log records might have been lost. The CIO error causes the following events to occur: A memory dump is taken of the FET and of the I-O buffer at the time of the error. Requests in progress at the time of the error are aborted with error message 474. CDCS switches to an alternate log file if one is available.	To determine the disposition of the journal log file, refer to the CDCS initiated job that uses DBREC to dump the journal log file. It might be necessary to purge the journal log file and rerun DBREC to initialize a new copy of this file. If this occurred during CDCS system recovery, manual recovery of the schema might be needed.	CDCS output file
I		CIO ERROR nnn ON QUICK RECOVERY FILE-SCHEMA sn	CIO encountered error number nnn while performing an operation on the quick recovery file for the schema named sn. The CIO error causes the following events to occur: A memory dump of the FET and a memory dump of the I-O buffer at the time of the error. Requests in progress at the time of the error are aborted with error message 474. CDCS sets the schema to error status and does not allow further access to it.	Purge the existing quick recovery file and rerun DBREC to initialize a new copy of this file before reinitiating CDCS. If this occurred during CDCS system recovery, manually recover the schema.	CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
I		CIO ERROR nnn ON TRANSACTION RECOVERY FILE-SCHEMA sn	CIO encountered error number nnn while performing an operation on the transaction recovery file for the schema named sn. The CIO error causes the following events to occur: A memory dump of the FET and a memory dump of the I-O buffer at the time of the error. Requests in progress at the time of the error are aborted with error message 474. CDCS sets the schema to error status and does not allow further access to it.	Purge the existing transaction recovery file and rerun DBREC to initialize a new copy of this file before reinitiating CDCS. If a restart identifier file exists for schema sn, it should also be initialized. Manually recover the areas affected by any uncommitted transactions. If this occurred during CDCS system recovery, manually recover the schema.	CDCS output file
F		CLTC ERROR	An internal error has occurred in clearing the long term connect.	Follow site-defined procedures for reporting software errors or operational problems.	CDCSBTF control point dayfile
I		CONTROL CARD PARAMETER xxxxx	The CDCS control statement parameter xxxxx is in error.	Correct the control statement error. Reinitialize CDCS.	CDCS system control point dayfile
I		CP SECONDS CHARGED BY CDCS xxxxxx.xxx	Central processor time of xxxxxx.xxx was used by CDCS at the system control point for processing requests for the particular user job.	The user job should be charged for xxxxxx.xxx seconds of central processor time.	User control point dayfile
I		CRM ERROR nnn ON CDCS RESTART IDENTIFIER FILE-SCHEMA sn	The indicated CRM error nnn occurred while performing an operation on the restart identifier file. If error 446 occurred, the transaction recovery file might have been reinitialized without also reinitializing the restart identifier file.	Correct the CRM error. If necessary, reinitialize the restart identifier file.	CDCS output file
C		CRM ERROR nnn WHILE ACCESSING MASTER DIRECTORY	The CRM error nnn occurred while reading the master directory.	Refer to the applicable CRM error diagnostic and correct as noted in the CYBER Record Manager Basic Access Methods reference manual.	CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
I		DBREC JOB REQUEST	A journal log file is full. CDCS attempts to create a DBREC job to dump the log file to tape.	None.	CDCS system control point dayfile
I		DBREC JOB SUBMITTED JOBNAME = jn	A journal log file is full. CDCS has submitted a DBREC job with the job name jn to dump the full journal log file to tape.	None.	CDCS system control point dayfile
I		DIR FILE PARAMETER xxxxx	The CDCS directive file parameter xxxxx is in error.	Correct the directive file error and reinitialize CDCS.	CDCS system control point dayfile
I		DOWN,CDCS	The system operator has entered a DOWN,CDCS command.	None.	CDCS system control point dayfile
I		DOWN COMPLETE	The area or schema specified in a DOWN command by the system operator has been given a down status.	None.	CDCS system control point dayfile
I		DOWN IN PROGRESS	The area or schema specified in a DOWN command by the system operator is being shut down.	None.	CDCS system control point dayfile
I		DUMP CDCS JOURNAL LOG NAME = pfn	CDCS is in the process of creating a DBREC job to dump the journal log file pfn to tape.	None.	CDCS system control point dayfile
N		DUPLICATE PARAMETER	A parameter was specified more than once on the CDCS control statement or directive file.	Correct the control statement or the directive file so that the parameter is specified only once. Reinitialize CDCS.	CDCS system control point dayfile
I		END CDCSBTF RUN	CDCSBTF processing has terminated.	None.	CDCSBTF control point dayfile
N		EQUAL SIGN MUST BE PRESENT	In either the CDCS control statement or the directive file, a parameter requiring an equal sign was specified without an equal sign.	Correct the CDCS control statement or the directive file and reinitialize CDCS.	CDCS system control point dayfile
N		EQUAL SIGN NOT ALLOWED	In either the CDCS control statement or the directive file, a parameter that must not have an equal sign was specified with an equal sign.	Correct the CDCS control statement or the directive file. Reinitialize CDCS.	CDCS system control point dayfile
C		ERROR DURING ROLL IN--CDCS ABORTED	An error occurred while rolling in the CDCS field length that was previously rolled out for system recovery.	Follow site-defined procedures for reporting software errors or operational problems.	CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
C		ERROR DURING ROLL OUT--CDCS ABORTED	An error occurred while rolling out that part of the CDCS field length that is unnecessary for system recovery.	Follow site-defined procedures for reporting software errors or operational problems.	CDCS system control point dayfile
F		ERROR FLAG n	The CDCS error flag has been set for user n.	Correct the error condition and return.	CDCSBTF control point dayfile
N		ERROR OR AREA id VERSION sn	During quick recovery file application processing for system recovery, an error occurred on area id.	Correct the error and manually recover the area.	CDCS system control point dayfile
I		ERROR nn WHILE REQUESTING A QUEUE DEVICE	During an attempt to route the CDCS OUTPUT file to the printer, the NOS/BE REQUEST call returned error code nn. Consult the NOS/BE Reference Manual for a description of the error code. The OUTPUT file is not routed to a printer, but no information is lost from the file. Continued writing to the OUTPUT file is not affected.	Immediate action is not required. If the problem persists notify the data administrator.	CDCS system control point dayfile
I		ERROR nn WHILE ROUTING A PRINT FILE	During an attempt to route the CDCS OUTPUT file to the printer, the ROUTE call returned error code nn. Consult the appropriate operating system reference manual for a description of the error code. The OUTPUT file is not routed to a printer, but no information is lost from the file. Continued writing to the OUTPUT file is not affected.	Immediate action is not required. If the problem persists notify the data administrator.	CDCS system control point dayfile
F		FATAL CDCS ERROR -- RUN-UNIT ABORTED	A fatal error occurred during CDCS processing.	Correct the error and resubmit the application program.	User control point dayfile
F		FC COMPLETE	An SCP function was requested with the complete bit set. This is an internal error.	Follow site-defined procedures for reporting software errors or operational problems.	CDCSBTF control point dayfile
N		FDL LOAD ERROR nnn ON AREA id VERSION vn - AREA DOWN. SCHEMA sn	The FDL load error nnn occurred on the area id during processing for system recovery or CDCS transaction reversal.	Correct the error and manually recover the area.	CDCS output file CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
			The area is set to error status. The schema sn is printed out for a non-system recovery error only.		
I		FIELD LENGTH DUMP IS ON THE OUTPUT FILE	A dump of the CDCS field length has been written to the output file.	None	CDCS system control point dayfile
N		FIRST TWO CHARACTERS MUST BE LETTERS	If the DT parameter is specified on the CDCS control statement or the directive file, the first two characters must be letters of the alphabet.	Correct the CDCS control statement or directive file and reinitialize CDCS.	CDCS system control point dayfile
I		IDLE,CDCS	The system operator has entered an IDLE,CDCS command.	None.	CDCS system control point dayfile
I		IDLE COMPLETE	The number of users of the area or schema specified in an IDLE command by the system operator has reached zero. The area or schema is given a down status.	None.	CDCS system control point dayfile
I		IDLE IN PROGRESS	The area or schema specified in an IDLE command by the system operator is being shut down. All active jobs using the area or schema are allowed to complete processing; all new jobs issuing invokes using the area or schema are aborted.	None.	CDCS system control point dayfile
I		INITIAL LOAD FL - iiiiii	Following CDCS initialiazation the system control point field length is iiiiii.	None	CDCS system control point dayfile
F		INSUFFICIENT MEMORY FOR CDCS type	CMM overflow occurred during the following type of processing: QUICK RECOVERY FILE APPLICATION, ROLL FORWARD, or ROLL BACK. This processing (and all further recovery of the schema if in system recovery mode) is terminated. The schema is set to error status.	Manually recover all areas in the schema. Increase the CDCS field length limit.	CDCS output file CDCS system control point dayfile
I		INTERNAL DOWN COMPLETE	As a result of internal problems, CDCS has shut down the area or schema specified in the preceding THE AREA ID/NAME IS or THE SCHEMA ID/NAME IS message. An internal down of a schema occurs	None.	CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
			when the schema log files are down. An internal down of an area occurs as a result of a CRM error that is classified by CDCS as a class 1 error.		
N		INTERNAL ERROR - DB$MABT	An internal error has occurred while processing an operator command.	Follow site-defined procedures for reporting software errors or operational problems.	CDCS output file CDCS system control point dayfile
N		INVALID AREA IDENTIFIER id ON CDCS QUICK RECOVERY FILE	The area identifier id found on the CDCS quick recovery file during system recovery is not found in the master directory. The schema is set to error status.	Check that information for the area identified by id is present in the master directory. Manually recover the areas in the schema.	CDCS output file CDCS system control point dayfile
N		INVALID CHARACTER	An invalid character appears in one of the parameters in the CDCS control statement or directive file.	Correct the CDCS control statement or the directive and reinitialize CDCS.	CDCS system control point dayfile
N		INVALID DATA ON CDCS TRANSACTION FILE	Invalid data was found on the CDCS transaction recovery file during processing for system recovery or transaction reversal. The temporary updates made within the CDCS transaction remain in effect. The schema is set to error status.	Analyze the invalid data on the CDCS transaction recovery file. Manually reverse the updates made within any uncommitted transactions. If this error occurred during system recovery, manually recover the areas in the schema.	CDCS output file CDCS system control point dayfile
N		INVALID VERSION NAME vn	The version name vn was not found in the master directory during processing for system recovery or CDCS transaction reversal. Processing is terminated for the schema; the schema is set to error status.	Check that the version name vn is a valid version name in the master directory. Manually reverse the updates made within any uncommitted CDCS transactions. If this error occurred during system recovery, manually recover the areas in the schema.	CDCS output file CDCS system control point dayfile
I		IO SECONDS CHARGED BY CDCS yyyyyy.yyy	Channel time of yyyyyy.yyy was used by CDCS at the system control point for processing requests for the particular user job.	The user job should be charged for yyyyyy.yyy seconds of channel time.	User control point dayfile
F		I/O (CIO) ERROR nnn ON CDCS file	The indicated CIO error nnn occurred during processing for system recovery or CDCS transaction reversal on one of the following files: the JOURNAL LOG FILE, the QUICK RECOVERY FILE, or the TRANSACTION RECOVERY FILE. The schema is set to error status.	Correct the CIO error. If this error occurred during system recovery, manually reverse the updates made within the uncommitted CDCS transaction. If this error occurred during transaction reversal processing, manually finish reversing any uncommitted transactions.	CDCS output file CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
F		I/O (CRM) ERROR nnn ON AREA id VERSION vn - AREA DOWN. SCHEMA sn	The CYBER Record Manager I/O error nnn occurred on the area during processing for system recovery or CDCS transaction reversal. The area is set to error status. The schema name is printed out for a nonsystem recovery error only.	Correct the error, and manually recover the area.	CDCS output file CDCS system control point dayfile
N		I/O (CRM) ERROR nnn ON CDCS RESTART IDENTIFIER FILE	The CYBER Record Manager error nnn occurred on the restart identifier file during system recovery. The schema is set to error status.	Correct the CRM error. Manually recover the areas in the schema.	CDCS output file CDCS system control point dayfile
F		JOB UNKNOWN TO SYSTEM	The CDCS request being processed is for a job that is no longer in the system.	Follow site-defined procedures for reporting software errors or operational problems.	CDCS output file
N		JOURNAL LOG FILE pfname FULL	The journal log file whose permanent file name is pfname has been released to be dumped to tape by DBREC. When both journal log files are in this state, jobs that use the schema associated with these files are delayed until a journal log file is available.	If both journal log files for the schema are in this state, determine if the CDCS-initiated DBREC jobs to dump the log files are executing. If they are executing, give them priority so they can complete the dump. If they are not executing, manually dump or replace the journal log files.	CDCS system control point dayfile
N		LAST RECOVERY POINT NOT FOUND	During system recovery, a matching recovery point record cannot be found in the journal log file to match the recovery point in the journal log file header. The schema is set to error status.	Analyze the journal log file. If this error occurred during system recovery, manually recover the areas in the schema.	CDCS output file CDCS system control point dayfile
N		LFN IS REQUIRED	No local file name was specified on the CDCSBTF control statement.	Add the local file name to the CDCSBTF control statement and resubmit the job.	CDCSBTF control point dayfile
N		MASTER DIRECTORY ALREADY ATTACHED	The master directory file specified on the CDCS control statement is already attached at the system control point.	Either attach the master prior to the CDCS call or specify the master directory attach information on the CDCS control statement or directive file.	CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
C		MASTER DIRECTORY MUST BE A PERMANENT FILE	The master directory is not a permanent file, or there is not a permanent file named MSTRDIR at the CDCS control point.	The data administrator must make the master directory permanent after running the DBMSTRD utility, or must attach the master directory with the local file name MSTRDIR. The master directory can be attached in the control stream, or the master directory attach parameters can be supplied via the CDCS control statement. (The latter option is preferred.)	CDCS system control point dayfile
N		MASTER DIRECTORY NOT AVAILABLE	During CDCS initialization, an error occurred while attempting to attach the master directory specified in the CDCS control statement or directive file.	Check that the master specified is a valid permanent file, that the correct permissions have been specified, and that the master directory is not attached elsewhere. Reinitialize CDCS.	CDCS system control point dayfile
I		MESSAGE FOR USER nn =	Any normal user control point dayfile message produced by CDCSBTF is preceded by this message.	None.	CDCSBTF control point dayfile
N		MFL MUST BE GREATER THAN BL	In the CDCS control statement or directive file, the value specified for the MFL parameter must be greater than the value specified for the BL parameter.	Correct the error and reinitialize CDCS.	CDCS system control point dayfile
N		MUST BEGIN WITH LETTER	An alphanumeric value was specified that did not begin with a letter in the CDCS control statement or directive file.	Correct the error and reinitialize CDCS.	CDCS system control point dayfile
N		NO TERMINATOR ON CONTROL CARD	A terminator was not specified in the control CDCS statement.	Specify either a period or a right parenthesis as the last character in the CDCS control statement.	CDCS system control point
N		NOT ALLOWED ON FILE	A parameter was specified in the CDCS directive file that can only be specified in the CDCS control statement.	Move the incorrectly specified parameter from the directive file to the CDCS control statement.	CDCS system control point dayfile
I		OUTPUT FILE ROUTED TO A PRINTER	A portion of the CDCS listing has been released for printing.	None	CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
I		PF WAIT ON AREA an (vn)	Another job has exclusive file access for the area an of version vn (version MASTER if vn does not appear). The job trying to access the area must wait until control has been relinquished and CDCS can attach the file.	None.	User control point dayfile CDCS output file
N		PFM ERROR nnn ATTACHING pfn	The permanent file error nnn occurred while trying to attach file pfn.	Check that file is not attached elsewhere. For a non-PUBLIC file unavailable on NOS, ensure that a PERMIT control statement specifying the user name for CDCS or CDCSBTF execution is in effect or that the procedure to initialize CDCS includes a USER control statement specifying the user name for CDCS execution. Refer to the applicable permanent file error diagnostic in volume 4 of the NOS reference set, or in the description of the FDB macro in the NOS/BE reference manual.	CDCS system control point dayfile
I		RECOVERY COMPLETED FOR SCHEMA sn	The system recovery phase for schema sn has completed successfully.	None.	CDCS system control point dayfile
N		RECOVERY IMPOSSIBLE FOR SCHEMA sn	The schema sn is in error status after the system recovery phase.	Correct the schema errors. Manually recover the areas in the schema before placing the schema in up status.	CDCS system control point dayfile
I		RECOVERY STARTED FOR SCHEMA sn	The system recovery phase for schema sn has started.	None.	CDCS system control point dayfile
F		RUN UNIT ABORT, CDCS CMM OVERFLOW	The run-unit is aborted because not enough memory is available to load the CDCS modules required.	CDCS field length requirements and usage should be examined by the data administrator and adjusted if necessary.	User control point dayfile CDCS output file
I		RUN-UNIT TERMINATED BEFORE CDCS COMMIT OR DROP REQUEST - DROP ASSUMED	The run-unit aborted within a CDCS begin/commit sequence. The drop function is automatically performed.	None.	User control point dayfile CDCS output file
N		SCHEMA ID MISMATCH ON CDCS file	During system recovery, the schema id specified in the header of either the QUICK RECOVERY FILE or the TRANSACTION RECOVERY FILE does not match the corresponding schema id specified in the master directory. The schema is set to error status.	Analyze the master directory and system file to locate the cause of the mismatch. Manually recover the areas in the schema.	CDCS output file CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
N		SCHEMA NOT UP - CDCS TRANSACTION ROLL BACK NOT COMPLETE SCHEMA sn	The schema is not in an up, idling or downing status during processing for system recovery or CDCS transaction reversal. The temporary updates made within the transaction remain in effect. Schema sn is printed out for non-system recovery only.	Manually reverse the updates made within the uncommitted transaction.	CDCS output file CDCS system control point dayfile
I		SCHEMA sn DBREC JOB REQUEST	A journal log file for schema sn is full. CDCS attempts to create a DBREC job to dump the log file to tape.	None.	CDCS system control point dayfile
F		SLTC ERROR	An internal error has occurred in setting the long term connect.	Follow site-defined procedures for reporting software errors or operational problems.	CDCSBTF control point dayfile
I		SMALL BLOCK BOUNDARY ADVANCED TO OR BEYOND ssssss nnnn TIMES	During CDCS execution the small block boundary has been advanced into the range of the small block increment beginning at ssssss on nnnn different occassions.	None	CDCS output file
I		TERM	The system operator has entered a TERM command.	None.	CDCS system control point dayfile
I		TERMINATE ENCOUNTERED BEFORE CDCS COMMIT OR DROP REQUEST - DROP ASSUMED	The application program issued a terminate within a CDCS begin/commit sequence. CDCS automatically performs the drop function to reverse uncommitted updates.	None.	User control point dayfile CDCS output file
I		THE AREA ID/NAME IS id/name	CDCS has changed the status of the specified area as a result of an operator command or an internal down. This message precedes all messages relating to the status of the area. The affected area is specified by name or identification where id is a 1- to 4-digit identification in the form ZZZ9 and name is a 1- to 30-character name in the form X(30).	None.	CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
N		THE CDCS file IS EMPTY - MUST BE ALLOCATED	One of the following files was found to be empty during system recovery: the JOURNAL LOG FILE, the QUICK RECOVERY FILE, or the TRANSACTION RECOVERY FILE. The schema is set to error status.	Rerun DBREC to allocate the indicated file. Manually recover the areas in the schema.	CDCS output file CDCS system control point dayfile
I		THE SCHEMA/ID NAME IS id/name	CDCS has changed the status of the specified schema as a result of an operator command or an internal down. This message precedes all messages relating to the status of the schema. The affected schema is specified by name or identification where id is a 1- to 4-digit identification in the form ZZZ9, and name is a 1- to 30-character name in the form X(30).	None.	CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
N		THIRD CHARACTER MUST BE NUMERIC	When the DT parameter is specified on the CDCS control statement or CDCS directive file, the third character must be a number between 0 and 9.	Correct the CDCS control statement or the CDCS directive file. Reinitialize CDCS.	CDCS system control point dayfile
N		TOO MANY CHARACTERS	The number of characters in a parameter value was greater than the maximum number allowed in the CDCS control statement or the CDCS directive file.	Correct the CDCS control statement or the CDCS directive file. Reinitialize CDCS.	CDCS system control point dayfile
N		TOO MANY PASSWORDS	More than five passwords for the master directory were specified in the CDCS control statement or the CDCS directive file.	Correct the CDCS control statement or the CDCS directive file. Reinitialize CDCS.	CDCS system control point dayfile
N		UN OR ID MUST BE GIVEN WITH MDPFN	The master directory name was specified without also specifying the UN or ID parameters in the CDCS control statement or CDCS directive file.	Correct the CDCS control statement or the CDCS directive file to specify the UN or ID parameter. Reinitialize CDCS.	CDCS system control point dayfile
N		UNABLE TO CREATE DBREC JOB - NO JOB/ATTACH DATA	Either the job control information was not specified in the master directory, or the attach data for the master directory was not specified in the CDCS control statement or the CDCS directive file.	Modify the master directory to include the job control information. Add the master directory attach data to the CDCS control statement or to the CDCS directive file. Manually dump and reallocate the full journal log file.	CDCS system control point dayfile
N		UNABLE TO SUBMIT DBREC JOB - I/O ERROR nnn	The I/O error nnn occurred on either the REQUEST statement (NOS/BE only) or the ROUTE statement of the DBREC job that CDCS was trying to submit.	Correct the I/O error. Manually dump and reallocate the full journal log file.	CDCS system control point dayfile
N		UNKNOWN PARAMETER	An invalid parameter was specified in the CDCS control statement or the CDCS directive file.	Correct the CDCS control statement or the CDCS directive file. Reinitialize CDCS.	CDCS system control point dayfile
I		UP,CDCS	The system operator has entered an UP,CDCS command. Subsequent invoke calls to CDCS will be allowed.	None.	CDCS system control point dayfile

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
I		UP COMPLETE	The system operator has removed the down status of an area or schema or terminated the effects of an IDLE command for an area or schema. Subsequent invokes using the area or schema will be allowed.	None.	CDCS system control point dayfile
I		UP IN PROGRESS	The area or schema is being restarted by the system operator following a previous DOWN or IDLE command.	None.	CDCS system control point dayfile
N		VALUE TOO LARGE	The value specified for the BL parameter was 1 larger than the maximum allowed (377777 octal) in the CDCS control statement or the CDCS directive file.	Specify a smaller value for the BL parameter. Reinitialize CDCS.	CDCS system control point dayfile
I		VERSION AFFECTED IS vn	The status of version vn has changed as a result of an operator command. This message precedes all messages relating to the status of the version.	None.	CDCS system control point dayfile
I		WAITING FOR CDCS	A user is trying to access CDCS when it is not active.	Contact the data administrator to initiate CDCS at a system control point.	User control point dayfile
F	384	600 - CHECKSUM MISMATCH SUBSCHEMA ssn	The checksum of the subschema ssn used to compile the applications program does not match the checksum of the subschema used at master directory run time.	Recompile and rerun program using correct subschema.	Data base status block User control point dayfile CDCS output file
N	385	601 - VIOLATION OF CONSTRAINT cn ON op OF RECORD rn	An attempt was made to perform the operation op (WRITE, DELETE, or REWRITE) on the record rn; the operation would have violated constraint cn.	Notify the data administrator.	Data base status block ZZZZZEG error file CDCS output file
N	386	602 - CRM ERROR nnn ON AREA an (vn) IN CONSTRAINT PROCESSING	A CRM error nnn occurred while performing an update operation on the area an of version vn (version MASTER if vn does not appear), which is involved in a constraint.	Correct the CRM error and resubmit the job.	Data base status block ZZZZZEG error file CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
N	387	603 - LOCKED RECORD/AREA-- REQUEST NOT PROCESSED	A TAF or interactive Query Update job has made a request to lock an area or to read a record in an area opened for input/output processing when the particular area or record in the area is locked by another user job. CDCS unlocks all the locks held by the TAF or interactive Query Update job and returns control to TAF or Query Update. If the TAF user is in CDCS transaction mode, the CDCS transaction is dropped.	The TAF user should include appropriate code in the application program to handle this diagnostic. Query Update issues the user a diagnostic and gives the user a choice about continuing processing.	Data base status block CDCS output file
N	388	604 - PF WAIT ON AREA an (vn)	Another job has exclusive file access to area an of version vn (version MASTER if vn does not appear) when a TAF or interactive Query Update job requested access to the area. CDCS returns control to TAF or Query Update. The TAF or Query Update job must wait for access until the other job returns the area file so that CDCS can attach the file.	TAF aborts the user job. Query Update issues the user a diagnostic and gives the user a choice about continuing processing.	Data base status block CDCS output file
N	389	605 - No message	CDCS is not active at a system control point.	Notify the data administrator.	Data base status block
F	390	606 - VERSION vn NOT IN SCHEMA sn	An invoke or version change operation specified a version name vn that cannot be found in the master directory for the schema sn.	Recompile and rerun the application program, using a correct version name for the schema sn.	Data base status block User control point dayfile CDCS output file
N	391	607 - AREA an (vn) IS NOT OPEN FOR I-O, op IS NOT ALLOWED	The operation op (REWRITE, or DELETE) is not allowed when the area an of version vn (version MASTER when vn does not appear) has not been opened for input/output processing.	Ensure that the updating program includes a statement that opens the area for both update and retrieval operations (that is, input/output processing) before the statement specifying the operation op.	Data base status block ZZZZZEG error file CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
N	392	610 - FOR AREA an (vn), KEY OF PRIOR READ MUST MATCH KEY ON op	The key of the record being written does not match the key of the record previously read from the area an of version vn (version MASTER if vn does not appear). The value of a data item that is a key cannot be modified; instead, the record containing the old key value must be deleted and a record containing the new key value must be written.	Resubmit the job specifying the statements to delete the old record and to write the new record.	Data base status block ZZZZZEG error file CDCS output file
F	393	611 - SCHEMA sn NOT IN MASTER DIRECTORY	The schema name sn used at execution time does not match the name of a schema for which information exists in the master directory.	Check that information for the schema has not been deleted from the master directory. If deleted, add information for the schema in a master directory run.	Data base status block User control point dayfile CDCS output file
F	394	612 - RESTART ID xxxxxxxxxx DOES NOT MATCH ID PREVIOUSLY ASSIGNED BY CDCS	The restart identifier specified in a DB$ASK or FINDTRAN request does not match the restart identifier that CDCS assigned to the run-unit earlier in a DB$GTID/ASSIGNID or DB$ASK/FINDTRAN request.	Verify the value of the restart identifier and resubmit the request.	Data base status block User control point dayfile CDCS output file
F	395	613 - AREA an NOT CLOSED BEFORE VERSION CHANGE	A version change operation was requested, but the area an was still open and had not been closed by the application program.	Recompile and rerun the application program, including the appropriate changes to the status of the open area before issuing the version change request.	Data base status block User control point dayfile CDCS output file
F	396	614 - SUBPROG SCHEMA/SUBSCHEMA NOT IDENTICAL TO MAIN PROGRAM	The subschema specified in the SUB-SCHEMA clause in the COBOL subprogram or in the SUBSCHEMA statement in the FORTRAN subprogram is not identical to the one specified in the main program.	Correct the SUB-SCHEMA clause in the COBOL subprogram and recompile, or correct the SUBSCHEMA statement in the FORTRAN subprogram and resubmit the program to the FORTRAN DML preprocessor.	Data base status block User control point dayfile CDCS output file
N	397	615 - LOCK MUST BE SET BEFORE READ ON AREA an (vn)	The request for a lock on area an of version vn (version MASTER if vn does not appear) was executed after a read was executed in the application program. When an area is open for input/output processing, an area lock (if specified) must be executed before a read is performed on the area.	Resubmit the job specifying an area lock before specifying a read request if the area is open for input/output processing.	Data base status block ZZZZZEG error file CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
F	398	616 - OUTSTANDING FATAL ERROR ON AREA an (vn)	A fatal error occurred on area an of version vn (version MASTER if vn does not appear) prior to the current request for input/output processing; CDCS denies the current request because of the previous error.	Check the data base status block, the user control point dayfile, or other status variables for information about the fatal error. Correct the error and resubmit the job.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file
F	399	617 - NO VERSION CURRENTLY ATTACHED	A previous version change request resulted in a nonfatal error. The current request is not allowed since no files are attached.	Correct the version change request error. Recompile and rerun the application program.	Data base status block User control point dayfile CDCS output file
F	400	620 - CDCS TRANSACTION UPDATE NOT IN EFFECT sn	For a FORTRAN program an attempt was made to issue a BEGINTRAN, a COMMITTRAN, a DROPTRAN, or a FINDTRAN request for schema sn, when the master directory does not specify a transaction recovery file for this schema. For a COBOL program an attempt was made to issue a DB$BEG, a DB$CMT, a DB$DROP, or a DB$ASK request for schema sn, when the master directory for schema sn does not specify a transaction recovery file for this schema.	Modify the master directory to include the transaction recovery file clause.	Data base status block User control point dayfile CDCS output file
F	401	621 - CDCS TRANSACTION IDENTIFIER NON-BLANK	A blank CDCS transaction identifier was used in a BEGINTRAN request for a FORTRAN program or in a DB$BEG request in a COBOL program.	Recompile and rerun the application program, using a non-blank transaction identifier in the BEGINTRAN or DB$BEG request.	Data base status block User control point dayfile CDCS output file
N	402	622 - MAXIMUM NUMBER CDCS BEGIN/ COMMIT SEQUENCES EXCEEDED	The maximum number of outstanding CDCS transactions for all users of the schema has been exceeded.	Reissue the BEGINTRAN or DB$BEG request periodically. The master directory transaction unit limit might need increasing.	Data base status block CDCS output file
F	403	623 - NO OUTSTANDING CDCS BEGIN TRANSACTION REQUEST	Either a commit request or a drop request was attempted without previously issuing a begin request.	Correct and recompile the application program.	Data base status block User control point dayfile CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
F	405	625 - REQUEST rq NOT ALLOWED IN CDCS BEGIN/COMMIT SEQUENCE	The request rq was attempted within a CDCS begin/commit sequence.	Correct the program to issue the request outside of the begin/commit sequence.	Data base status block User control point dayfile CDCS output file
F	406	626 - ILLEGAL AREA NAME	The area name used by the application program does not match the name of an area for which information exists in the master directory.	Change the program to supply the correct area name.	Data base status block User control point dayfile CDCS output file
I	407	627 - No message	A null record occurrence was encountered on a file during relation processing.	Check the condition during processing if the application program is interested in recording null record occurances.	Data base status block C.DMRST object routine in COBOL or variable DBSTAT or DBSnnnn in FORTRAN
F	408	630 - ILLEGAL LOCK MODE	The lock mode specified in the lock request is an invalid lock mode; valid lock modes are: exclusive and protected.	Change the program to supply either the value EXCLUSIVE or PROTECTED.	Data base status block User control point dayfile CDCS output file
F	409	631 - NO CDCS RESTART IDENTIFIER FILE DEFINED	An attempt was made to either request a restart identifier or find the last committed transaction for a schema which does not have a restart identifier specified within the master directory.	Either remove the request from the application program, or define the restart identifier file within the master directory.	Data base status block User control point dayfile CDCS output file
I	410	632 - No message	A control break was encountered on a file during relation processing.	Check the condition during processing if the application program is interested in recording control break information.	Data base status block C.DMRST object routine in COBOL or variable DBSTAT or DBSnnnn in FORTRAN
N	411	633 - RESTART IDENTIFIER ALREADY ASSIGNED BY CDCS	An attempt is made to request a restart identifier after a restart identifier has already been assigned.	Correct and recompile the application program.	Data base status block CDCS output file
F	412	634 - MAXIMUM CDCS BEGIN/COMMIT UPDATE COUNT EXCEEDED	The maximum number of updates allowed within a CDCS begin/commit sequence has been exceeded.	Change the application program to issue fewer updates in a begin/commit sequence or increase the update count limit in the master directory.	Data base status block User control point dayfile CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
F	413	635 - SYSTEM FILE NOT AVAILABLE FOR SCHEMA sn	One of the following CDCS system files is not available: the journal log file, the transaction recovery file, the restart identifier file, the quick recovery file, or the procedure library file.	Verify that the file in question has been initialized by DBREC.	Data base status block User control point dayfile CDCS output file
F	414	636 - AREA an (vn) UNUSABLE DURING ROLL-BACK	An area error occurred applying a record to the area an version vn (version MASTER if vn does not appear) during processing to reverse a CDCS transaction. The temporary updates made within the transaction remain in effect.	Correct the problem causing the error and manually finish reversing the updates made within the CDCS transaction.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file
F	415	637 - RUN ABORT, SCHEMA sn DOWN	The schema sn needed by the application program cannot be accessed because the schema sn has been marked down by the system operator. This message occurs during active processing using this schema.	Resubmit the job when the schema is brought up.	Data base status block User control point dayfile CDCS output file
F	416	640 - RUN ABORT, CDCS DOWN	CDCS is terminated; issued to active application programs.	Resubmit the job when CDCS is brought up again at a system control point.	Data base status block User control point dayfile CDCS output file
F	417	641 - SUBSCHEMA ssn NOT IN MASTER DIRECTORY	The subschema ssn used at execution time does not match the name of a subschema for which information exists in the master directory.	Check that information for the subschema is present in the master directory. If it is not present, add information for the subschema in a master directory run.	Data base status block User control point dayfile CDCS output file
F	418	642 - CDCS JOURNAL LOG FILES UNAVAILABLE SCHEMA sn	The CDCS journal log files specified for the schema sn have not been attached at CDCS initialization time, are down, or cannot be accessed for some other reason.	Check that journal log files are not in error status or are being dumped to tape by a DBREC job.	Data base status block User control point dayfile CDCS output file
F	419	643 - SCHEMA sn NOT AVAILABLE	The schema sn needed by the application program cannot be accessed. The message occurs at invocation time.	Resubmit the job when the schema is brought up again.	Data base status block User control point dayfile CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
F	420	644 - CDCS UNAVAILABLE--AN INVOKE IS NOT ALLOWED	The system operator is terminating, idling, or downing CDCS. CDCS no longer accepts requests from new run-units.	Resubmit job when CDCS is brought up again at a system control point.	Data base status block User control point dayfile CDCS output file
F	421	645 - ILLEGAL AREA ORDINAL	An illegal value was received from the application program.	Follow site-defined procedures for reporting software errors or operational problems.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file
F	422	646 - RESTART IDENTIFIER xxxxxxxxxx	The restart identifier specified in a DB$ASK or FINDTRAN request is in use by another run-unit. This restart identifier can no longer be used in the request.	Verify and correct the old restart identifier. If this does not correct the problem, consult the data administrator.	Data base status block User control point dayfile CDCS output file
F	423	647 - PFM ERROR ec ON AREA an (vn)	The permanent file error ec occurred while attaching the area an of version vn (version MASTER if vn does not appear).	For a non-PUBLIC file unavailable on NOS, ensure that a PERMIT control statement specifying the user name for CDCS or CDCSBTF execution is in effect or that the procedure to initialize CDCS includes a USER control statement specifying the user name for CDCS execution. Refer to the applicable permanent file error diagnostic in volume 4 of the NOS reference set, or the description of the FDB macro in the NOS/BE reference manual.	Data base status block User control point dayfile CDCS output file
F	424	650 - PFM ERROR ec ON AREA an (vn) INDEX FILE	The permanent file error ec occurred while attaching the index file for the area an of version vn (version MASTER if vn does not appear).	For a non-PUBLIC file unavailable on NOS, ensure that a PERMIT control statement specifying the user name for CDCS or CDCSBTF execution is in effect or that the procedure to initialize CDCS includes a USER control statement specifying the user name for CDCS execution. Refer to the applicable permanent file error diagnostic in volume 4 of the NOS reference set, or the description of the FDB macro in the NOS/BE reference manual.	Data base status block User control point dayfile CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
F	425	651 - AREA an (vn) DOWN	The area an of version vn (version MASTER if vn does not appear) cannot be used because of physical storage failure, a fatal CRM error during area processing, or because the operator gives the area a down status. The area can be brought up again by issuing the operator command UP for the area or by reinitializing CDCS.	Notify the data administrator.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
N	426	652 - AREA an (vn) ALREADY OPEN	An open was attempted on area an of version vn (version MASTER if vn does not appear). This area has already been opened.	Correct and recompile the application program.	Data base status block ZZZZZEG error file CDCS output file
T/N	427	653 - CRM ERROR nnn DURING op ON AREA an (vn)	The indicated CRM error nnn occurred while performing the operation op on the area an of version vn (version MASTER if vn does not appear). If the file is structurally bad, CDCS marks the area down and does not allow further access to it.	Refer to the ZZZZZEG file and applicable CRM error diagnostic and correct as noted in the CRM Advanced Access Methods reference manual. If the file is structurally bad, reconstruct the file and give it an up status in order to allow access to the area.	Data base status block ZZZZZEG error file CDCS output file
N	428	654 - AREA an (vn) NOT OPEN	The indicated area an of version vn (version MASTER if vn does not appear) has not been opened.	Correct and recompile the application program.	Data base status block ZZZZZEG error file CDCS output file
F	429	655 - NO PROCEDURE LIBRARY SCHEMA sn	No data base procedure library has been defined in the master directory for the indicated schema sn. Data base procedures are specified in the schema definition.	Modify the master directory to include the procedure library specification.	Data base status block User control point dayfile CDCS output file
F	430	656 - INSUFFICIENT MEMORY FOR ROLLBACK SCHEMA sn	A CYBER Memory Manager (CMM) overflow condition occurred during processing to reverse a CDCS transaction. The temporary updates made within the transaction remain in effect. The schema sn is set to error status.	Manually reverse the updates made within any uncommitted transaction. Increase the memory limit for CDCS.	Data base status block User control point dayfile CDCS output file
N	431	657 - INCORRECT RECORD TYPE DURING op, AREA an (vn)	The DDL record type specified on the operation op does not match the record type determined by the RECORD CODE criteria for the area an of version vn (version MASTER if vn does not appear).	Check that record code is correct for record name given in the I/O statement.	Data base status block ZZZZZEG error file CDCS output file
N	432	660 - KEY MAPPING ERROR DURING op, AREA an (vn)	An illegal key value occurs during the operation op on the area an of version vn (version MASTER if vn does not appear). This could be a conversion error.	Correct the key value.	Data base status block ZZZZZEG error file CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
F	433	661 - FDL ERROR ec ON DBPROC pn	The indicated FDL error ec occurred while loading the indicated data base procedure pn.	Check that the data base procedure is in capsule form and has been properly put in a procedure library. Refer to the applicable FDL error code and correct as noted in the CYBER Loader reference manual.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file
F	434	662 - ILLEGAL RECORD ORDINAL	An illegal value was received from the application program.	Recompile program. Follow site-defined procedures for reporting software errors or operational problems.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file
N	435	663 - DEADLOCK ON AREA an (vn)	CDCS has unlocked all locks held by the program.	Provide appropriate code to handle recovery from a deadlock.	Data base status block ZZZZZEG error file CDCS output file
N	436	664 - ILLEGAL REQUEST ON AREA an (vn), READ OR FILE LOCK REQUIRED BEFORE op	The CDCS locking mechanism requires that the indicated operation op (REWRITE or DELETE) be preceded by a READ statement or a file lock to lock the record for area an version vn (version MASTER if vn does not appear).	Issue a READ statement in the program or lock the entire area using either the C.LOK or the DB$LKAR routine in COBOL or the DML LOCK statement in FORTRAN.	Data base status block ZZZZZEG error file CDCS output file
F	437	665 - PRIVACY BREACH ATTEMPT	The correct access control key to gain access to a given area was not supplied.	A USE FOR ACCESS CONTROL procedure must be coded in the COBOL program, a DML PRIVACY statement must be coded in the FORTRAN program, or an ACCESS directive must be entered to Query Update to supply the correct privacy key.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file
F	438	666 - ERROR IN RELATION DATA NAME DEFINITION	The RESTRICT clause in the subschema references an improperly defined data name in the application program.	Correct data name and recompile the application program. If problem persists, follow site-defined procedures for reporting software errors or operational problems.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file
N	439	667 - BAD RECTYPE CODE VALUE	The record type supplied by the VALUE option does not match one of the values expected by CDCS.	Correct the RECORD CODE information in the schema or the value stored in the record in the application program.	Data base status block ZZZZZEG error file CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
N	440	670 - CANNOT CHANGE REC TYPE ON op, AREA an (vn)	The record type of the data base record, as determined by the RECORD CODE criteria, cannot be modified during the indicated operation op for the area an of version vn (version MASTER if vn does not appear).	Check that the record code on the operation is the same as the original.	Data base status block ZZZZZEG error file CDCS output file
F	441	671 - ILLEGAL RELATION ORDINAL	An illegal value was received from the application program.	Recompile program. If problem persists, follow site-defined procedures for reporting software errors or operational problems.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file
N	442	672 - BAD RECTYPE FROM DBPROC pn	The record type supplied by the indicated data base procedure pn does not match one of the values expected by CDCS.	Correct the data base procedure or the value stored in the record.	Data base status block ZZZZZEG error file CDCS output file
N	443	673 - op ABORT BY DBPROC pn	Data base procedure pn specified the indicated operation op should be aborted. Error might be intentional.	Check that data base procedure is being used correctly.	Data base status block ZZZZZEG error file CDCS output file
F	444	674 - RUN UNIT ABORT (DURING op) BY DBPROC pn	Data base procedure pn specified that the application program should be aborted during the indicated operation op. DURING op does not appear if the data base procedure was called on an error.	Check that data base procedure is being used correctly.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file
N	445	675 - RECORD MAPPING ERROR DURING op ON RECORD rn	A record mapping error occurred on the indicated record rn during the specified operation op. The value of an item could not be converted or a data validation error occurred on that item (data stored into the item does not satisfy requirements in the CHECK clause). No data is transferred to or from the program working storage area. The subschema ordinal of the item in error is returned in the auxiliary status word of the data base status block.	Correct the value of the item to be converted.	Data base status block ZZZZZEG error file CDCS output file

TABLE B-1. CDCS DIAGNOSTICS (Contd)

Type	Dec. E.C.	Message	Significance	Action	Destination
F	446	676 - BAD RETURN CODE FROM DBPROC pn	The indicated data base procedure pn supplied a return code other than 0, 1, or 3.	Change the procedure to return a valid return code.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file
F	447	677 - DBPROC pn NOT DEFINED FOR SCHEMA sn	The indicated data base procedure pn cannot be found in the data base procedure library for the schema sn.	Check that data base procedure is in the correct library and that the library has been specified in the master directory entry for the particular schema.	Data base status block ZZZZZEG error file User control point dayfile CDCS output file
F	472	730 - CDCS TRANSACTION RECOVERY FILE I/O ERROR DURING ROLL BACK, - SCHEMA DOWN	An error occurred reading the CDCS transaction recovery file while processing to reverse a CDCS transaction. The temporary updates made within the transaction remain in effect. The schema is set to error status.	Correct the problem causing the I/O error. Manually reverse the updates made within the uncommitted transaction.	Data base status block User control point dayfile CDCS output file
F	474	732 - CDCS SYSTEM FILE I-O ERROR	The requested function was not performed because of a CIO or CRM error on one of the CDCS logging files.	Notify the data administrator who has specific information about the error.	Data base status block User control point dayfile CDCS output file
N	475	733 - INQUIRE TRANSACTION RESTART IDENTIFIER ridname UNKNOWN TO CDCS	An attempt was made to find the CDCS transaction identifier associated with the restart identifier of a job that has terminated normally. Because the job terminated normally, the restart identifier cannot be found.	Assume the user job that was using the restart identifier has terminated normally.	Data base status block User control point dayfile CDCS output file
I		num CMM OVERFLOW CALLS MADE	The 6-digit number num indicates the number of CMM overflow calls that occurred during CDCS processing. A large number indicates excessive overhead.	Usually none. If an excessive number occurred, the data administrator should consider increasing the maximum field length for CDCS.	CDCS system control point dayfile
I		num MAXIMUM SCM WORDS USED	The 6-digit number num indicates the maximum number of central memory words used by CDCS.	None.	CDCS system control point dayfile

FORTRAN-DML DIAGNOSTIC MESSAGES

All diagnostic messages that can be issued by the DML preprocessor are listed in table B-2. Messages are listed in numeric order according to the error code associated with the message. The general significance of the diagnostic message and the action to be taken by the user accompany each message listed in the table.

Messages are written to the local file named by the E parameter of the DML control statement. These messages are of the following types:

C Catastrophic error. Compilation cannot continue. DML advances to the end of the current program unit and attempts to process the next program unit.

F Fatal error. DML cannot process the statement or routine in which the error occurs. Unresolvable semantic errors also fall into this category. DML continues processing with the next statement.

W Warning error. The syntax of the DML statement or routine is incorrect, but DML has been able to recover by making an assumption about what was intended. Processing continues.

T Trivial error. The syntax or usage of the DML statement or routine is correct, but it is questionable. Processing continues.

TABLE B-2. FORTRAN DML DIAGNOSTICS

Error Code	Message	Severity	Significance	Action
100	EMPTY INPUT FILE	C	The input file is empty and processing is terminated.	Create a new file and resubmit the program.
101	LEFT PARENTHESIS MISSING	F	A left parenthesis was expected but not found.	Correct the error and resubmit the program.
102	INVALID SUBSCHEMA NAME	F	The specified subschema name does not conform to naming conventions.	Correct the error and resubmit the program.
103	RIGHT PARENTHESIS MISSING	F	A right parenthesis was expected but not found.	Correct the error and resubmit the program.
104	EXTRANEOUS DATA FOLLOWING RIGHT PARENTHESIS	F	Data was found beyond the end of a statement.	Correct the error and resubmit the program.
105	NO INVOKE STATEMENT WAS FOUND IN PRECEDING PROGRAM UNIT	F	An INVOKE statement must be included in each program unit that contains DML statements.	Correct the error and resubmit the program.
107	MODE=O INVALID ON OPEN RELATION	F	The mode option must be I or IO for an OPEN relation.	Correct the error and resubmit the program.
108	UNKNOWN SOURCE WORD	F	The DML preprocessor is unable to interpret the statement.	Correct the error and resubmit the program.
109	INVALID MODE	F	The mode must be I, IO, or O.	Correct the error and resubmit the program.
110	EQUAL SIGN MISSING	F	An equal sign was expected but not found.	Correct the error and resubmit the program.
111	SUBSCHEMA NOT AVAILABLE	F	The subschema could not be found in the specified library.	Specify the correct library and resubmit.
112	EXTRANEOUS DATA IN PARAMETER LIST	F	DML does not recognize the item following the equal sign in a keyword specification such as END=, ERR=, KEY=, or MODE=.	Correct the error and resubmit the program.
113	KEY PARAMETER MISSING	F	The keyword KEY was expected but not found.	Correct the error and resubmit the program.

TABLE B-2. FORTRAN DML DIAGNOSTICS (Contd)

Error Code	Message	Severity	Significance	Action
114	INVALID RELATIONAL OPERATOR	F	The relational operator is missing or is invalid.	Correct the error and resubmit the program.
115	INVALID ITEM NAME	F	The item name does not conform to FORTRAN naming conventions.	Correct the error and resubmit the program.
116	ITEM IS NOT DEFINED AS A KEY FOR THIS REALM	F	The item is not defined as a key in the schema.	Specify the correct key or update the schema.
117	PERIOD MISSING	F	A period was expected but not found.	Correct the error and resubmit the program.
118	INTERNAL DML ERROR	C	This is an internal error.	Follow site-defined procedures for reporting software errors and operational problems.
119	COMMA OR RIGHT PARENTHESIS NOT FOUND	F	A comma or right parenthesis was expected but not found.	Correct the error and resubmit the program.
121	PRIVACY TYPE MUST BE LITERAL OR ARRAY NAME	F	The privacy key must be specified for FORTRAN 5 as a character constant, variable, or 3-word array.	Correct the error and resubmit the program.
122	INVALID PARAMETER	F	The specified parameter is not valid in this statement.	Correct the error and resubmit the program.
123	DUPLICATE PARAMETER	F	The same parameter cannot be specified more than once.	Correct the error and resubmit the program.
124	PRIVACY PARAMETER LITERAL IS GREATER THAN 30 CHARACTERS	F	The privacy literal must be 30 or fewer characters.	Correct the error and resubmit the program.
125	PRIVACY PARAMETER NOT SPECIFIED	F	The keyword privacy= followed by the privacy key is required.	Correct the error and resubmit the program.
126	INSUFFICIENT FIELD LENGTH - DML ABORTED	C	Not enough field length was specified to complete preprocessing. The job is aborted.	Use the RFL control statement to increase field length.
127	DML LANGUAGE VERSION (LV) DIFFERS FROM SUB-SCHEMA	F	A subschema and a FORTRAN DML applications program must specify the same version of FORTRAN. For example, when a subschema is compiled with F5 specified in the DDLF control statement, application programs that use that subschema must specify LV=F5 in the DML control statement.	Correct the error and resubmit the program.
128	INVALID FORTRAN LABEL	F	The END=s and ERR=s parameters must specify a valid statement label (any 1- through 5-digit positive nonzero integer).	Correct the error and resubmit the program.

TABLE B-2. FORTRAN DML DIAGNOSTICS (Contd)

Error Code	Message	Severity	Significance	Action
129	END= IS NOT VALID IN FORTRAN 4	F	The parameter END= is allowed only in FORTRAN 5.	Correct the error and resubmit the program.
130	ERR= IS NOT VALID IN FORTRAN 4	F	The parameter ERR= is allowed only in FORTRAN 5.	Correct the error and resubmit the program.
131	TRANSACTION PARAMETER NOT VALID FORTRAN NAME	F	The variable used in the DML statement for transaction processing does not conform to FORTRAN naming conventions.	Correct the error and recompile the program.
132	TRANSACTION IDENTIFIER EXCEEDS 10 CHARACTERS	F	The literal used in the DML statement for transaction processing must be 10 characters or less.	Correct the error and recompile the program.
133	VERSION PARAMETER NOT VALID FORTRAN NAME	F	The variable that specifies version name does not conform to FORTRAN naming conventions.	Correct the error and recompile the program.
134	VERSION-NAME EXCEEDS 7 CHARACTERS	F	The version name must be 7 characters or less.	Correct the error and recompile the program.
135	LOCK TYPE PARAMETER NOT VALID FORTRAN NAME	F	The variable that specifies the lock type does not conform to FORTRAN naming conventions.	Correct the error and recompile the program.
136	LOCK TYPE LITERAL IS INVALID - SHOULD BE EITHER EXCLUSIVE OR PROTECTED	F	The lock type literal is invalid. Valid values are either exclusive or protected.	Correct the error and recompile the program.
137	RESTART IDENTIFIER NOT VALID FORTRAN NAME	F	The variable that specifies the restart identifier does not conform to FORTRAN naming conditions.	Correct the error and recompile the program.
138	ITEM IN MAJOR KEY IS INVALID	F	A data item in the major key is not valid. The item is not contiguous or is not the first item of the concatenated key.	Correct the error and recompile the program.

GLOSSARY C

The glossary contains terms unique to the description of the components of DMS-170 and terms common within the data processing industry that have special connotations within the context of DMS-170.

AAM -
See Advanced Access Methods.

Access Control -
Protection of data from unauthorized access or modification. Also called privacy.

Access Control Key -
The value an application program must supply to CDCS in order to gain access to a particular data base area. Also called a privacy key.

Access Control Lock -
The value associated with a data base area which must be known by the application program if the program is to gain access to the area.

Actual Item -
An item for which the value is materialized when the record is stored or updated. The value is determined by a data base procedure and is physically stored in the record.

Actual Key -
A file organization in which records are identified by system-assigned keys.

Advanced Access Methods (AAM) -
A file manager that processes indexed sequential, direct access, and actual key file organizations and supports the Multiple-Index Processor. See CYBER Record Manager.

After-Image -
A copy of a data base record after it has been updated. Contrast with Before-Image.

Alias -
A data name used in a COBOL or Query Update subschema in place of a schema data name; a data item used in a FORTRAN subschema in place of a schema data name.

Alphanumeric -
Any character in the computer character set defined in appendix A.

Alternate Key -
A data item for which the value can be used to randomly access a record in a CRM file.

Application Program -
A COBOL program, a FORTRAN program, or a Query Update application that interfaces with CDCS. The FORTRAN source program must have DML statements that provide for the CDCS interface.

Area -
A uniquely named schema data base subdivision that contains data records; identified in the subschema as a realm; associated in the master directory with at least one permanent file.

Array -
A data item consisting of a set of elements of the same type that is defined by a single name; FORTRAN subschema data structure. See Elementary Item.

Attach -
The process of making a permanent file accessible to a job by specifying the proper permanent file identification and passwords.

Automatic Recovery -
CDCS initiated recovery operations that make a data base usable and consistent after some type of software or hardware failure.

Backup Dump -
A copy of all or selected portions of a data base, which is produced on a regularly scheduled basis for the explicit purpose of data base recovery.

BAM -
See Basic Access Methods.

Basic Access Methods (BAM) -
A file manager that processes sequential and word addressable file organizations. See CYBER Record Manager.

Batch Test Facility -
An absolute program residing on the CDCS system library that allows CDCS to run at the same control point as a user program.

Before-Image -
A copy of a data base record before it has been updated. Contrast with After-Image.

Beginning-of-Information (BOI) -
As defined by CRM, the start of the first user record in a file. System-supplied information, such as an index block or control word, does not affect beginning-of-information. Any label on a tape exists prior to beginning-of-information.

Block -
On tape, information between interrecord gaps on a tape. CRM defines several blocks depending on file organization, as shown in table C-1.

TABLE C-1. BLOCK TYPES

Organization	Blocks
Indexed sequential	Data block; index block
Direct access	Home block; overflow block
Actual key	Data block
Sequential	Block type I, C, K, E

Cascade Effect -
A phenomenon causing the indirect propagation of erroneous data in a data base. Erroneous data is processed by a properly functioning program and made part of another previously correct record.

CDCS -
See CYBER Database Control System.

Checksum -
A one-word attribute generated by DDL for each area and relation in a schema and for each subschema. Checksums are stored in the schema and subschema directories, and in the master directory. CDCS references them to check the validity of using a previously compiled subschema with the current schema or of using a previously compiled application program with a current subschema.

Child Record Occurrence -
A record occurrence that has another record occurrence (the parent record occurrence) at the next numerically lower rank in a hierarchical tree structure of the relation. Contrast with Parent Record Occurrence.

Compression -
The process of condensing a record to reduce the amount of storage space required. Compression can be performed by either a system-supplied or a user-supplied routine. Contrast with Decompression.

Concurrency -
Simultaneous access to the same data in a data base by two or more application programs during a given span of time.

Condensed Schema/Subschema Table (CST) -
A table comprised of information from the schema and subschema directories. A CST is part of the master directory and is generated for every schema and subschema combination.

Constant -
A fixed value, explicitly written in a source statement. In a FORTRAN subschema, the term corresponds to a literal in the schema.

Constraint -
A control imposed on records in interdependent files or on items in a single file for the purpose of protecting the integrity of data in a data base during update operations. A constraint is defined in the schema and is based on common data items in the records or within a record.

Control Break -
A condition during a relation read which signifies that a new record occurrence was read for the parent file.

Control Word -
A system-supplied word that precedes each W type record in storage.

Conversion -
The process of changing data characteristics between the schema and the subschema.

CRM -
See CYBER Record Manager.

CYBER Database Control System (CDCS) -
The DMS-170 controlling module that provides the interface between an application program and a data base.

CYBER Record Manager (CRM) -
A generic term relating to the common products BAM and AAM, which run under the NOS and NOS/BE operating systems and allow a variety of record types, blocking types, and file organizations to be created and accessed. The execution time input/output of COBOL, FORTRAN, Sort/Merge 4, Sort/Merge 5, ALGOL, and the DMS-170 products is implemented through CRM. Neither the input/output of the NOS or NOS/BE operating systems themselves nor any of the system utilities such as COPY or SKIPF is implemented through CRM. All CRM file processing requests ultimately pass through the operating system input/output routines.

Data Administrator -
A person or a group who defines the format and organization of a data base and is responsible for maintaining and monitoring a data base.

Data Base -
A systematically organized, central pool of information; organization is described by a schema.

Data Base Procedure -
A special-purpose routine that performs a predefined operation; its use is specified in a schema and is initiated by CDCS.

Data Base Status Block -
An area of memory defined within an application program to which CDCS returns information concerning the status of operations on data base files and relations. The status block is updated after each CDCS operation.

Data Base Transaction -
See Transaction.

Data Base Version -
A set of data files that is described by a schema. Data base versions are defined in the master directory. When data base versions are used, a schema (the description of the data base) can be used with more than one set of files (each set of files being a data base version).

Data Description Language (DDL) -
The language used to structure a schema and a subschema.

Data Integrity -
Validity of data. Checking the validity of data items on a data base update can be performed by means of the CHECK clause or data base procedures.

Data Item -
Smallest unit of data within a record; can be an elementary or group data item (for FORTRAN, an elementary item only).

Data Manipulation Language (DML) -
A language, patterned after application language statements, that provides access to the data base. In the DMS-170 environment, DML is a part of COBOL statements; FORTRAN DML statements must be incorporated in a FORTRAN source program and translated by the DML preprocessor into statements acceptable to the FORTRAN compiler.

Data Name -
Used in a schema, in a COBOL or Query Update subschema, and in a COBOL or Query Update application program; a name identifying a group or elementary data item in the data base; can contain up to 30 letters, digits, or embedded hyphens, but must contain at least one letter. Similar to item name.

Deadlock -
A situation that arises in concurrent data base access when an application programs is contending for a resource that is locked by another program, and neither program can proceed without that resource.

Decompression -
The process of expanding a compressed record to restore it to its original size. The user can supply a decompression routine or use a system-supplied routine. Contrast with Compression.

Dependent Record Occurrence -
A record occurrence that is the dependent member of a condition defined by a constraint. Contrast with Dominant Record Occurrence.

Direct Access -
In the context of CRM, one of the five file organizations. It is characterized by the system hashing of the unique key within each file record to distribute records randomly in blocks called home blocks of the file.

In the context of NOS permanent files, a direct access file is a file that is accessed and modified directly, as contrasted with an indirect access permanent file.

Directed Relationship -
The logical relational structure that defines the specific order in which the files in a relation are traversed and the order in which the record occurrences are retrieved. The relational structure is formed by the join terms declared in the schema.

Directory -
A file that contains area and record attributes of the data base; created when the schema or subschema is compiled; an object schema or subschema.

Dominant Record Occurrence -
A record occurrence that is the dominant member of a condition defined by a constraint. Contrast with Dependent Record Occurrence.

Duration Loading -
A procedure that allows certain CDCS overlay capsules and CRM capsules to be loaded during CDCS initialization and kept in memory during the entire execution time of CDCS. Performance is improved in situations of heavy usage by avoiding memory fragmentation and the extra time that occurs when the capsules are each loaded as they are used.

Elementary Item -
A data item that is not subdivided into other data items. An elementary item that is part of a group item has the highest level number in the group item. A nonrepeating elementary item in the schema corresponds to a variable in a FORTRAN subschema; a repeating elementary item corresponds to an array.

End-of-Information (EOI) -
Defined by CRM in terms of the file organization and file residence as shown in table C-2.

TABLE C-2. END-OF-INFORMATION BOUNDARIES

File Organization	File Residence	Physical Position
Sequential	Mass storage	After the last user record.
	Labeled tape in SI, I, S, or L format	After the last user record and before any file trailer labels.
	Unlabeled tape in SI or I format	After the last user record and before any file trailer labels.
	Unlabeled tape in S or L format	Undefined.
Word Addressable	Mass storage	After the last word allocated to the file, which might be beyond the last user record.
Indexed Sequential, Actual Key	Mass storage	After the record with the highest key value.
Direct Access	Mass storage	After the last record in the most recently created overflow block or home block with the highest relative address.

Figurative Constant -
A fixed value with a predefined name.

File -
A collection of records treated as a unit; an area in the schema; a realm in the subschema.

Fixed Occurrence Data Item -
A data item that is repeated the same number of times in all records.

Floating Point Literal -
A string of digits with a decimal point and an optional exponent.

Flushing -
A process of force writing log file and data buffers.

FORM -
A general-purpose file management utility for manipulating records and creating and converting files.

FORTRAN DML -
See Data Manipulation Language.

Group Item -
A data item that is subdivided into other data items; a collection of data items. Group items cannot be referenced in a FORTRAN subschema.

Hierarchical Tree Structure -
A representation that commonly illustrates record occurrences for files joined in a directed relation. The root of the tree is a record occurrence in the root file and each successive level represents the record occurrences in each joined file.

Home Block -
Mass storage allocated for a file with direct access organization at the time the file is created.

Identifier -
A data name that is referenced uniquely through a combination of subscripts and qualifiers; used in a schema and COBOL and Query Update subschemas.

Indexed Sequential -
A file organization in which records are stored in ascending order by key.

Interrelated Files -
Those data base files that are connected through a relation defined in the schema using join terms.

Invocation -
Preparation by CDCS to handle input/output requests from application programs. The invocation call is generated by the COBOL compiler, by the FORTRAN DML INVOKE statement, or by Query Update when executing particular directives. The invocation call must occur prior to any other CDCS processing requests.

Item Name -
Used in a FORTRAN subschema or application program; a name identifying an elementary item in the data base; can contain up to seven letters or digits; must begin with a letter. Similar to data name.

Join Terms -
The identifiers that are used to join two files in a relation.

Joining Files -
The logical linkage of one file to another in a relation through the use of data items called join terms or identifiers.

Journal Log File -
An independent sequential permanent file (not a data base area) assigned for the purpose of collecting designated information to be used to reconstruct or restore a data base.

Keyword -
A reserved word that is required in a source program clause of a schema, of a COBOL or Query Update subschema, or as input to a data base utility; a word that is required in a source program statement of a FORTRAN subschema.

Level -
For system-logical-records, an octal number 0 through 17 in the system-supplied 48-bit marker that terminates a short or zero-length PRU.

Level Number -
Used in a schema and in a COBOL or Query Update subschema; a number defining the structure of data within a record; if not specified in data description entry in a schema, level number 01 is assumed by default.

Literal -
A constant completely defined by its own identity; called a constant in a FORTRAN subschema.

Local File Name-
The 1 to 7 display code alphabetic or numeric characters by which the operating system recognizes a file. Every local file name in a job must be unique and begin with a letter.

Logging -
The facility of CDCS through which historical records are kept of operations performed by users on data base areas. Logging information is used in data base recovery and restoration operations.

Logical Record -
Under NOS, a data grouping that consists of one or more PRUs terminated by a short PRU or zero-length PRU. Equivalent to a system-logical-record under NOS/BE.

Mapping -
The process by which CDCS produces a record or item image conforming to the schema or subschema description.

Master Directory -
A file containing information used by CDCS in processing. This information consists of schema and subschema tables, media parameters, and data base procedure library and logging specifications.

Nested Group Item -
A group item that is subordinate to another group item. Up to three levels of nested groups can be specified in a schema.

Noise Record -
The number of characters the tape drivers discard as being extraneous noise rather than a valid record. Value depends on installation settings.

Nonrepeating Group Item -
A COBOL subschema data item that contains subordinate data items. This group item occurs only once in each record occurrence; used to identify a series of related data items.

Null Record Occurrence -
A record occurrence composed of the display code right bracket symbol in each character position. It is used in a relation occurrence to denote that no record occurrence qualifies or that a record occurrence does not exist at a given level in the relation.

Operation -
A particular function performed on units of data; for instance, opening or closing an area, or storing or deleting a record.

Overflow Block -
Mass storage the system adds to a file with direct access organization when records cannot be accommodated in the home block.

Overlay Capsules -
A special type of capsule called OVCAP, designed for use with overlays, and called into memory by an overlay (see Duration Loading).

Parent Record Occurrence -
A record occurrence that has another record occurrence at the next numerically higher rank in a hierarchical tree structure of the relation. Contrast with Child Record Occurrence.

Partition -
As defined by CRM, a division within a file with sequential organization. Generally, a partition contains several records or sections. Implementation of a partition boundary is affected by file structure and residence, as shown in table C-3.

Notice that in a file with W type records, a short PRU of level 0 terminates both a section and a partition.

Permanent File -
A file on a mass storage permanent file device that is protected against accidental destruction by the system and can be protected against unauthorized access or destruction.

Physical Record Unit (PRU) -
Under NOS and NOS/BE, the amount of information transmitted by a single physical operation of a specified device (see table C-4).

A PRU that is not full of user data is called a short PRU; a PRU that has a level terminator but no user data is called a zero-length PRU.

Primary Key -
A key that must be defined for an indexed sequential, direct access, or actual key file when the file is created. Random access of a record depends on the use of the primary key.

Privacy -
See Access Control.

TABLE C-3. PARTITION BOUNDARIES

Device	Record Type (RT)	Block Type (BT)	Physical Boundary
PRU device	W	I	A short PRU of level 0 containing a one-word deleted record pointing back to the last I block boundary, followed by a control word with a flag indicating a partition boundary.
	W	C	A short PRU of level 0 containing a control word with a flag indicating a partition boundary.
	D F,R T,U Z	C	A short PRU of level 0 followed by a zero-length PRU of level 17 octal.
	S	-	A zero-length PRU of level number 17 octal.
S or L format tape	W	I	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a deleted one-word record pointing back to the last I block boundary, followed by a control word with a flag indicating a partition boundary.
	W	C	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a control word with a flag indicating a partition boundary.
	D,F,T, R U Z	C,K,E	A tapemark.
	S	-	A tapemark.
Any other tape format	-	-	Undefined.

TABLE C-4. PRU SIZES

Device	Size in Number of 60-Bit Words
Mass storage (NOS and NOS/BE only).	64
Tape in SI format with coded data (NOS/BE only).	128
Tape in SI format with binary data.	512
Tape in I format (NOS only).	512
Tape in any other format.	Undefined.

Privacy Key -
The value an application program must supply to CDCS in order to gain access to a particular data base area. Also called an access control key.

Procedure Library -
A permanent file containing the data base procedure referenced in a schema.

PRU Device -
Under NOS and NOS/BE, a mass storage device or a tape in SI or I format, so called because records on these devices are written in PRUs.

Qualification -
The method whereby a nonunique name can be made unique. If the name exists within a hierarchy of names, it can be made unique by mentioning one or more of the higher levels (normally the record name) of the hierarchy.

Qualifier Identifier -
A data item that restricts which record occurrences are to be retrieved when reading a relation. It is specified in the subschema. This data item can be the same data item specified as a join field in the schema.

Quick Recovery File -
A random permanent file used internally by CDCS to restore the structural integrity of the data base. It contains a copy of all data blocks for a data base which have been updated since the last recovery point.

Random File -
In the context of CRM, a file with word addressable, indexed sequential, direct access, or actual key organization in which individual records can be accessed by the values of their keys; in the context of the NOS and NOS/BE operating systems, a file with the random bit set in the file environment table in which individual records are accessed by their relative PRU numbers.

Rank -
The rank of a file in a DMS-170 relation corresponds to the position of the file in the schema definition of the relation. The ranks of the files joined in a relation are numbered consecutively, with the root file having a rank of 1.

Realm -
A uniquely named DMS-170 subschema data base subdivision that contains data records; identified in the schema as an area; a file.

Realm Ordinal -
A unique identifier assigned to each realm in a DMS-170 subschema when the subschema is compiled. Realm ordinals for a FORTRAN subschema are used in conjunction with the FORTRAN status variables.

Reconstruction -
Re-creation of all or specified portions of a DMS-170 data base utilizing a backup dump of the data base and the after-image record entries from the journal log file.

Record -
As defined by CRM, a group of related characters. A record or a portion thereof is the smallest collection of information passed between CRM and a user program. Eight different record types exist, as defined by the RT field of the file information table.

For CDCS processing, a record is equivalent to a record occurrence.

Other parts of the operating systems and their products can have additional or different definitions of records.

Record Code -
A unique value that identifies a specific record type in an area with multiple record types. The value is either contained in a data item in the record or derived by execution of a data base procedure.

Record Mapping -
See Mapping.

Record Occurrence -
The actual data base record, which conforms in the data base file to a record type described in the schema and conforms for use in the application program to the record description in the subschema.

Record Qualification -
The method used to restrict which records are to be returned to the user by specifying criteria that must be satisfied by a record occurrence. Record qualification is allowed only for relation reads.

Record Type -
A term that can have one of several meanings, depending on the context. CRM defines eight record types established by an RT field in the file information table. Tables output by the loader are classified as record types such as text, relocatable, or absolute, depending on the first few words of the tables.

In DDL, record type is defined in the record description entry of the schema. It is a description of the attributes of a record and the items included in the record which serves as a template by which record occurrences are interpreted; an arbitrary number of record occurrences can exist for each record type.

Recovery -
A process that makes a data base useful after some type of software or hardware failure has occurred.

Recovery Point -
A user-generated or system-generated point to which CDCS guarantees recovery with no loss of data. User-generated recovery points are initiated by COBOL calls to DB$RPT, by FORTRAN calls to DMLRPT, or by transmission of the Query Update directive RECOVERY. System-generated recovery points are initiated by CDCS when certain conditions occur, such as a full quick recovery file, a termination of the CDCS interface by a user program, or the committing of a data base transaction.

Relation -
The logical structure formed by the joining of files for the purpose of allowing an application program to retrieve data from more than one file at the same time. The structure is declared in the schema and is based on common identifiers in the files.

Relation Occurrence -
The logical concatenation of a record occurrence from each record type specified in the relation. Each read of the relation yields a relation occurrence. The record descriptions of the records that compose the relation occurrence are contained in the subschema.

Relational Data Base -
A data base of files joined in relations through data item identifiers.

Repeating Group -
A collection of data items which occurs a number of times within a record occurrence; can consist of elementary items, group items, and vectors; cannot be referenced in a FORTRAN subschema.

Restart Identifier
A unique identifier for a run-unit that is maintained by CDCS for program restart operations in transaction processing.

Restart Identifier File
A random permanent file used internally by CDCS to support program restart operations for programs that request a restart identifier.

Restoration -
Resetting of a data base to a previous state by applying the before-image record entries from the journal log file to the data base in its current state.

Result Item -
An item whose value is determined by a data base procedure. The time at which the value is determined depends on whether the item is an actual or virtual result item. The value of the item is generated by a data base procedure operating on the value of other items in the same record.

Root File -
The file that has the rank 1 in a relation; its record occurrences are pictured at the root of a tree in a hierarchical tree structure.

Run-Unit -
In the CDCS Environment, the execution of a user job at a control point. The user job becomes a run-unit at invocation and ceases to be a run-unit at termination. The user job can be an application program (a main program with associated subprograms), a Query Update application, or a sequence of TAF tasks.

Schema -
A detailed description of the internal structure of a data base.

Schema Identification Entry -
A schema source program statement that assigns a name to the schema.

Section -
As defined by CRM, a division within a file with sequential organization. Generally, a section contains more than one record and is a division within a partition of a file. A section terminates with a physical representation of a section boundary (see table C-5).

The NOS and NOS/BE operating systems equate a section with a system-logical-record of level 0 through 16 octal.

Sequential -
A file organization in which records are stored in the order in which they are generated.

Short PRU -
A PRU that does not contain as much user data as the PRU can hold and is terminated by a system terminator with a level number.

Under NOS, a short PRU defines EOR; under NOS/BE, a short PRU defines the end of a system-logical-record. In the CRM context, a short PRU can have several interpretations depending on the record and blocking types.

Source Data Item -
The data item received by CDCS. For a CRM GET function, the data item is in the schema format; for a CRM STORE function or REWRITE function, the data item is in the subschema format. Contrast with Target Data Item.

Source Identifier -
An identifier that links a parent record type in one file to a child record type in another file in a relation.

Subschema -
A detailed description of the portion of a data base to be made available to one or more application programs.

TABLE C-5. SECTION BOUNDARIES

Device	Record Type (RT)	Block Type (BT)	Physical Representation
PRU device	W	I	A deleted one-word record pointing back to the last I block boundary followed by a control word with flags indicating a section boundary. At least the control word is in a short PRU of level 0.
	W	C	A control word with flags indicating a section boundary. The control word is in a short PRU of level 0.
	D,F R, T,U,Z	C	A short PRU with a level less than 17 octal.
	S	-	Undefined.
S or L format tape	W	I	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a deleted one-word record pointing back to the last I block boundary, followed by a control word with flags indicating a section boundary.
	W	C	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a control word with flags indicating a section boundary.
	D F R T U Z	C K,E	Undefined.
	S	-	Undefined.
Any other tape format	-	-	Undefined.

Subschema Item Ordinal -
A unique identifier within a record assigned to each item in a subschema when the subschema is compiled. Subschema item ordinals are used in conjunction with the data base status block.

Subschema Library -
A permanent file containing one or more subschemas.

System-Logical-Record -
Under NOS/BE, a data grouping that consists of one or more PRUs terminated by a short PRU or zero-length PRU. These records can be transferred between devices without loss of structure.

Equivalent to a logical record under NOS.

Equivalent to a CRM S type record.

Target Data Item -
The data item transferred by CDCS. For a CRM GET function, the data item is in the subschema format; for the CRM STORE or MODIFY function, the data item is in the schema format. Contrast with Source Data Item.

Target Identifier -
An identifier in a child record type which must be identical to the source identifier in the parent record type for two files in a relation to be joined.

Transaction -
A series of update operations identified by a user-assigned transaction identifier. A transaction is bracketed by a begin transaction operation and either a commit or drop operation.

Transaction Facility (TAF) -
A network product that controls transaction processing, called TAF-transaction processing in this manual. See Transaction Processing.

Transaction Processing-
In the context of CDCS, a process that provides control for interrelated updates to data base files by means of a transaction; also provides a program restart capability which can be used for restarting an application program after a system failure.

In the context of TAF (called TAF-transaction processing in this manual), a manipulation of a data base by high speed handling or repetitive executions of a relatively small number of jobs called tasks.

Transaction Recovery File
A random permanent file used internally by CDCS to perform automatic recovery. It contains before-image records of records updated within an incompleted transaction.

Traversing Files -
The process by which CDCS goes from one to another of the files joined in a relation. Record occurrences are retrieved based on identifiers. A record occurrence from each file is returned to the user.

Type
The storage format of a data item which determines permitted values, length, and arithmetic meaning.

User Function -
A request from an application program to open or close an area or to perform a specific operation on a record or data item.

User Work Area -
The collection of record areas allocated by the COBOL compiler, the FORTRAN DML preprocessor, or Query Update within a run-unit, for record types defined in the subschema used by a run-unit.

Variable -
A single named data item in the FORTRAN subschema data structure.

Variable Occurrence Data Item -
A data item that is repeated a specific number of times in each record occurrence. The number of occurrences is controlled by a preceding elementary data item; the data name option of the OCCURS clause in a schema; or format 2 of the OCCURS clause in a COBOL or Query Update subschema.

Vector -
An elementary data item that is repeated a number of times in each record occurrence.

Version -
A data base version (see Data Base Version); a Query Update directive (VERSION) that specifies a catalog file.

Virtual Item -
An item that is part of a record when it is retrieved, but was not part of the record (had no value) when it was stored. The value of the item is determined by a data base procedure.

W Type Record -
One of the eight record types supported by CRM. Such records appear in storage preceded by a system-supplied control word. The existence of the control word allows files with sequential organization to have both partition and section boundaries.

Zero-Byte Terminator -
12 bits of zero in the low order position of a word that marks the end of the line to be displayed at a terminal or printed on a line printer. The image of cards input through the card reader or terminal also has such a terminator.

Zero-Length PRU -
A PRU that contains system information, but no user data. Under CRM, a zero-length PRU of level 17 is a partition boundary. Under NOS, a zero-length PRU defines EOF.

SYNTAX SUMMARIES

D

The format specifications of the COBOL, FORTRAN, and Query Update interfaces with CDCS are shown in this appendix. The format specifications are grouped into the following types:

COBOL statements and routines

DML statements and routines for use with FORTRAN 4

DML statements and routines for use with FORTRAN 5

Query Update directives

Within each type, the directives are shown in alphabetic order. Two forms of notations appear in this appendix. One form follows the COBOL convention, and the other form follows the FORTRAN convention. Refer to the Notation section for information about the COBOL and FORTRAN notation conventions. Detailed information for each format is referenced by page numbers.

COBOL STATEMENTS AND ROUTINES

ENTER "DB$LKAR" USING realm-name, lock-type, err-code 2-11

ENTER "DB$RPT" USING rpt-num, comment 2-11

ENTER "DB$SIR" USING item-name 2-12

ENTER "DB$VERS" USING version-name, err-code 2-12

ENTER "DB$WAIT" 2-12

OPEN { INPUT realm-name-1 [, realm-name-2]... / OUTPUT realm-name-3 [, realm-name-4]... / I-O realm-name-5 [, realm-name-6]... } ... 2-13

OPEN { INPUT relation-name-1 [, relation-name-2]... / I-O relation-name-3 [, relation-name-4]... } ... 2-13

READ realm-name [NEXT] RECORD [INTO identifier] 2-14

[; AT END imperative-statement]

READ realm-name RECORD [INTO identifier] 2-14

[; KEY IS data-name]

[; INVALID KEY imperative-statement]

READ relation-name [NEXT] RECORD 2-14

[; AT END imperative-statement]

READ relation-name RECORD 2-14

[; KEY IS data-name]

[; INVALID KEY imperative-statement]

REWRITE record-name [FROM identifer] 2-15

[; INVALID KEY imperative-statement]

START realm-name [KEY { IS EQUAL TO / EQUALS / IS = / EXCEEDS / IS GREATER THAN / IS > / IS NOT LESS THAN / IS NOT < } data-name] 2-15

[; INVALID KEY imperative-statement]

```
START relation-name [ KEY { IS EQUAL TO
                          | EQUALS
                          | IS =
                          | EXCEEDS
                          | IS GREATER THAN
                          | IS >
                          | IS NOT LESS THAN
                          | IS NOT < } data-name ]
    [; INVALID KEY imperative-statement]
```

2-15

```
SUB-SCHEMA IS subschema-name
```

2-5

```
USE FOR ACCESS CONTROL
    [ ON { INPUT
         | I-O
         | INPUT I-O
         | I-O INPUT } ]
    ; KEY IS access-key
    [ FOR { realm-name-1 [, realm-name-2]... } ] .
          { REALMS }
```

2-16

```
USE FOR DEADLOCK
    ON { realm-name-1 [, realm-name-2]... } .
       { REALMS }
```

2-17

```
WRITE record-name [FROM identifier]
    [; INVALID KEY imperative-statement]
```

2-18

DML STATEMENTS AND ROUTINES FOR USE WITH FORTRAN 5

QUERY UPDATE DIRECTIVES

```
ACCESS KEY [ IS {literal    } [ ON ||INPUT|| ] [ FOR {AREA {area-name}...      } ] ... ]...
               {data-name   }   ||I-O   ||       {CATALOG [catalog-name]}
                                ||OUTPUT||       {AREAS                  }
```
4-6

```
CREATE area-name OF subschema-name
    [FROM LIBRARY permanent-file-name]
    [(permanent-file-parameter [PW])]
    [FOR DATABASE version-name]
```
4-8

```
DISPLAY [UPON file-name-1] [ KEY {literal    } ]  {[expression [AS data-name-2]]...}
                           [     {data-name-1}  ]  {SAME                          }
                           [     {IN file-name-2} ]
                           [ FROM file-name-3     ]
```
4-9

```
EXHIBIT [ data-name
        | REPORTS [report-name [layout-directive]]
        | SESSIONS [session-id [transmission-id-1 [TO transmission-id-2]]]
        | TEMPORARY
        | RELATION ]
```
4-9

```
EXTRACT UPON file-name {expression [AS data-name]}...
```
4-10

```
INVOKE subschema-name
    [OF schema-name]
    [FROM LIBRARY permanent-file-name]
    [(permanent-file-parameters [PW])]
    [FOR DATABASE version-name]
```
4-11

```
MODIFY [record-name]
    ||[SETTING {data-name-1}...] [USING key-name] [SETTING {data-name-2}...] [FROM file-name]||
    ||MOVE expression-1 TO {data-name-3}... [AND expression-2 TO {data-name-4}...]...||
    [ VETO ]
    [ PASS ]
```
4-11

```
RECOVERY POINT USING data-name-1 {literal    }
                                 {data-name-2}
```
4-12

```
REMOVE [record-name]
    [SETTING {data-name-1}...]
    USING keyname
    [SETTING {data-name-2}...]
    [FROM file-name]
    [ VETO ]
    [ PASS ]
```
4-12

```
STORE [record-name]                                                    4-13
    ||  SETTING {data-name-1}... [FROM file-name]                      ||
    ||  MOVE expression-1 TO {data-name-2}... [AND expression-2 TO {data-name-3}...]...  ||
    [ VETO ]
    [ PASS ]

UPDATE area-name                                                       4-13

VERSION is catalog-file OF subschema-name                              4-13
    [FROM LIBRARY permanent-file-name]
    [(permanent-file-parameters [PW])]
    [FOR DATABASE version-name]

VIA relation-name                                                      4-14
```

FUTURE SYSTEM MIGRATION GUIDELINES E

This appendix contains programming practices recommended by CDC for users of the software described in this manual. When possible, application programs based on this software should be designed and coded in conformance with these recommendations.

Two forms of guidelines are given. The general guidelines minimize application program dependence on the specific characteristics of a hardware system. The feature use guidelines ensure the easiest migration of an application program to future hardware or software systems.

GENERAL GUIDELINES

Good programming techniques always include the following practices to avoid hardware dependency:

Avoid programming with hardcoded constants. Manipulation of data should never depend on the occurrence of a type of data in a fixed multiple such as 6, 10, or 60.

Do not manipulate data based on the binary representation of that data. Characters should be manipulated as characters, rather than as octal display-coded values or as 6-bit binary digits. Numbers should be manipulated as numeric data of a known type, rather than as binary patterns within a central memory word.

Do not identify or classify information based on the location of a specific value within a specific set of central memory word bits.

Avoid using COMPASS in application programs. COMPASS and other machine-dependent languages can complicate migration to future hardware or software systems. Migration is restricted by continued use of COMPASS for stand-alone programs, by COMPASS subroutines embedded in programs using higher-level languages, and by COMPASS owncode routines used with CDC standard products. COMPASS should only be used to create part or all of an application program when the function cannot be performed in a higher-level language or when execution efficiency is more important than any other consideration.

FEATURE USE GUIDELINES

The recommendations in the remainder of this appendix ensure the easiest migration of an application program for use on future hardware or software systems. These recommendations are based on known or anticipated changes in the hardware or software system, or comply with proposed new industry standards or proposed changes to existing industry standards.

ADVANCED ACCESS METHODS

The Advanced Access Methods (AAM) offer several features within which choices must be made. The following paragraphs indicate preferred usage.

Access Methods

The recommended access methods are indexed sequential (IS), direct access (DA), and multiple index processor (MIP).

Record Types

The recommended record types are either F for fixed length records, or W for variable length records. Record length for W records is indicated in the control word; the length must be supplied by the user in the RL FIT field on a put operation and is returned to the user in RL on a get operation.

FORTRAN Usage

The following machine-independent coding practices are encouraged for a FORTRAN programmer using AAM:

Initialize the FIT by FILExx calls or by the FILE control statement.

Modify the FIT with STOREF calls.

Use FORTRAN 5 CHARACTER data types when working with character fields rather than octal values of display code characters; specify lengths of fields, records, and so forth, in characters rather than words.

COBOL 5

COBOL 5 offers choices among features that perform the same function. The following paragraphs indicate preferred usage.

Comment Entries

Do not use the comment entries AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, or SECURITY. This information can be documented with a normal COBOL comment, using an * in column 7. This usage complies with a proposed revision to the ANSI standard.

SUPERVISOR, MEMORY, and ASSIGN OBJECT-PROGRAM Clauses

Do not use the SUPERVISOR, MEMORY, and ASSIGN OBJECT-PROGRAM clauses. These clauses are currently used only for documentation and the information in them can be specified with a comment. This usage complies with a proposed revision to the ANSI standard.

Collating Sequence

Whenever possible, specify the ASCII collating sequence.

File Organizations

For Advanced Access Methods, use indexed or direct organization with or without alternate keys; do not use actual-key organization. For word-addressable files, use a method that can be easily modified to byte addresses; the REPLACE statement can be used to modify the code at a later date. No restrictions are imposed on sequential file usage.

Record Types

Do not specify a particular record type with the USE clause. This decision can be made by the COBOL compiler based on the characteristics of the records. If, however, the program would produce a default record type of D or T (for example, the record contains an OCCURS DEPENDING ON clause), the record type should be forced to W by specifying RT=W.

RECORD Clause

Use the VARYING phrase within the RECORD clause if records are to be variable length. This usage complies with a proposed revision to the ANSI standard.

RECORDING MODE Clause

Do not use the RECORDING MODE clause; usage of this clause should not be necessary in most application programs.

Level 77 Items

Do not use Level 77 items. Use elementary 01 items instead. This usage complies with a proposed revision to the ANSI standard.

REDEFINES and RENAMES Clauses

Avoid REDEFINES, RENAMES, or group operations involving synchronized or COMP-n items, particularly where usage is based on a knowledge of the internal representation of data, such as floating-point layout or the number of characters per word.

Group Items

Avoid operations such as reference modification, string, and unstring on group items containing noncharacter data or synchronized items. Such usage is dependent on the characteristics of the hardware system.

ALTER Statement

Do not use the ALTER statement. In general, use of the ALTER statement does not conform to good coding practices. If its use is necesary, the GO TO DEPENDING ON statement can be used instead. This usage complies with a proposed revision to the ANSI standard.

Segment Numbers

Do not use segment numbers 50 through 99; in the future, segment numbers will be limited to 0 through 49. This range should be adequate for almost all applications. Programs should no longer need the initial-state capability of numbers 50 through 99 because this capability is only useful with the ALTER statement. This usage complies with a proposed revision to the ANSI standard.

OPEN REVERSED Statement

Do not use the REVERSED phrase of the OPEN statement. Usage has always led to inefficient processing, and the capability generally should not be required. This recommendation complies with a proposed revision to the ANSI standard.

SORT Statement

Specify the WITH DUPLICATES IN ORDER phrase for those sorts that expect it. Do not rely on this ordering as the default. This usage complies with a proposed revision to the ANSI standard.

STRING and UNSTRING Statements

One of the operands in a STRING or UNSTRING statement must not be used as a subscript in the same statement. Operand reuse in subscripts is poor coding practice and generally can be avoided. This recommendation complies with a proposed revision to the ANSI standard.

DMS-170

DMS-170 offers several features among which choices must be made. The following paragraphs indicate preferred usage of CDCS, DDL, and of Query Update in support of CDCS.

Multiple Record Descriptions

Do not include multiple record descriptions on a single file.

Repeating Groups

Avoid the use of the OCCURS clause, repeating groups, or arrays within records; as an alternative, the repeating data can be normalized into separate records on a different file. If repeating data must be used, limit usage to fixed length groups (no OCCURS DEPENDING ON clause) and to simple (unnested) OCCURS clauses.

Alternate Keys on Repeating Groups

Avoid the specification of alternate keys on repeating groups. The data can be normalized as indicated under Repeating Groups.

Collating Sequence

Use the default collating sequence or the ASCII collating sequence.

REDEFINES Clause

Use the REDEFINES clause only for alphanumeric-to-alphanumeric redefinitions, where the term alphanumeric has the meaning assigned by COBOL to data. In general, avoid the use of REDEFINES where use is based on a knowledge of the internal representation of data (floating-point layout, number of characters per word, and so forth).

Query Update Syntax

Use the new directives INVOKE, STORE, MODIFY, and REMOVE instead of the directives USE, INSERT, UPDATE, and DELETE.

FORTRAN 5

FORTRAN 5 offers several capabilities that are processor-dependent. The use of such capabilities restrict FORTRAN 5 program migration. The following paragraphs indicate preferred usages.

Processor-Dependent Values

Coding should not depend on the internal representation of data (floating-point layout, number of characters per word, and so forth). Where coding must depend on these representations, use PARAMETER variables for processor-dependent characteristics such as the number of characters per word.

BOOLEAN Data Types

Do not use boolean data types and operations (SHIFT, MASK, and so forth) because they can be processor-dependent. Use type CHARACTER instead, if working with character data.

LOCF Function

Do not use the intrinsic function LOCF. For most applications, this function should not be necessary.

ENCODE and DECODE Statements

Do not use ENCODE and DECODE; use the ANSI standard internal file feature instead. ENCODE and DECODE operations are generally dependent on the number of characters per word.

DATE, TIME, and CLOCK Functions

Do not dismantle values returned by the DATE, TIME, and CLOCK functions; use these functions only for printing out values as a whole.

BUFFER IN and BUFFER OUT Statements

Do not use BUFFER IN and BUFFER OUT, especially when use depends on the number of characters per word.

CYBER Record Manager Interface Routines

Do not use the CYBER Record Manager interface routines for sequential files. Instead, use FORTRAN input/output statements such as READ or WRITE.

Overlays

If possible, use segmented loading instead. If overlays must be used, do not depend on such properties as reinitialization of variables when an overlay is reloaded.

LABEL Subroutine

Avoid use of the LABEL subroutine. Changes to the ANSI standard for tape labels might require changes to the interface used by this subroutine.

KEYWORD USED IN DML STATEMENTS AND VARIABLES AND COMMON BLOCKS GENERATED BY THE DML PREPROCESSOR F

This appendix lists the keywords used in FORTRAN DML statements. The names of variables and common blocks generated by the DML preprocessor are also listed. The names of these variables and common blocks should not be used as variable names in a FORTRAN program that utilizes DML and CDCS to access a data base.

FORTRAN DML KEYWORDS

The FORTRAN DML keywords are listed below in alphabetical order.

```
ASSIGNID

BEGINTRAN

CLOSE
COMMITRAN

DELETE
DROPTRAN

END
.EQ.
ERR

FINDTRAN

.GE.
.GT.

I
INVOKE
IO

KEY

.LE.
LOCK
.LT.

MODE

NEWVERSION

.NOT.

O
OPEN

PRIVACY

READ
REWRITE

START
SUBSCHEMA

TERMINATE

UNLOCK

WRITE
```

VARIABLES AND COMMON BLOCKS GENERATED BY THE DML PREPROCESSOR

The names of variables and common blocks generated by the DML preprocessor are listed below in alphabetical order.

DBFnnnn	where nnnn is 0001 through 9999
DBInnnn	where nnnn is 0001 through 9999
DBNnnnn	where nnnn is 0001 through 9999
DBREALM	
DBRELST	
DBRUID	
DBRnnnn	where nnnn is 0001 through 9999
DBSCNAM	
DBSTAT	
DBSnnnn	where nnnn is 0001 through 9999
DBTEMP	
DBTnnnn	where nnnn is 0001 through 9999
DBnnnn	where nnnn is 0000 through 9999
Dnnnnxx	where nnnn is 0001 through 9999 and xx is AA through ZZ

The CDCS Batch Test Facility provides the capability of running CDCS along with one or more user jobs as a normal batch job at a control point. This facility is intended primarily for use when a program is being developed and tested, since data and file definitions are changing frequently during this stage. By running the Batch Test Facility, the user can attach new versions of the master directory file each time the job is run. Normally, when CDCS is running at a system control point, the control point must be dropped and reinitiated to attach a new master directory file.

The CDCS Batch Test Facility is an absolute program, called CDCSBTF, which resides on the system library. CDCSBTF consists of the normal CDCS system control point routines and tables plus a set of special routines that communicate with the user programs at the CDCS control point. These special routines load the user programs and simulate the interface between the user control point and a system control point.

Multiple copies of CDCSBTF can be run concurrently with each other and with a system control point version of CDCS. In addition, as many as 16 user programs can be run with each copy of CDCSBTF. Because user programs are loaded by a program-initiated load from CDCSBTF, the programs must be in relocatable binary format. Programs in absolute binary format, as well as segmented programs and overlays, cannot be run with CDCSBTF.

CDCSBTF EXECUTION

The CDCSBTF program is called into execution by a control statement. Before the program is executed, however, several file requirements and restrictions should be considered. In addition, load maps from the user call load operations can be obtained by setting switches prior to the execution of CDCSBTF. The allocation of field length and other required resources is handled as for a normal batch job.

CDCSBTF CONTROL STATEMENT

The CDCS Batch Test Facility is executed by specifying the CDCSBTF control statement. The format and parameters of the CDCSBTF control statement are shown in figure G-1. The CDCSBTF control statement cannot exceed 80 characters in length. A slash (/) indicates the end of the list of user program file names and the beginning of the parameter list. Optionally, the comma immediately following CDCSBTF can be replaced by a left parenthesis; the terminating period can be replaced by a right parenthesis.

```
CDCSBTF,lfn-1[,lfn-2,]... [/p][,p] ... .

lfn_i  The local file name of up to 16 user programs to be executed

/      The end of the user program list and the beginning of the parameter list

p      A parameter; the parameters are as follows:

         DIR=lfn
             Directive file for CDCSBTF control statement parameters

         BL=nn
             Maximum pooled buffer space

         CP=t1
             Central processor time

         IO=t2
             Input/output time

         MFL=fl
             Maximum field length for CDCSBTF

         CAP=ovcap-name1[/ovcap-name2]...
             Static load of CDCS overlay capsules where possible OVCAP names are as follows:

                 CON       QRF
                 DBP       REL
                 INV       TRAN
                 JLOG

         CRM=fs1[/fs2]...
             Static load CRM capsules where fs is a file structure as follows:

                 AK
                 DA
                 IS
                 MIP or MP

         MDPFN=pfn
         UN=user-name
         ID=user-name
         PW=pwrd1[/pwrd2]...   } Permanent file
         FAM=family-name       } information for
         PN=pack-name          } master directory
         DT=device-set         } file
         SN=set-name           }

Note: The OVCAP names and CRM capsule loading parameters can themselves be parameters to the control statement.
```

Figure G-1. CDCSBTF Control Statement Format

Directive File

An optional directive file can be used to contain the parameters in addition to or instead of the parameters in the CDCSBTF control statement. Use of a directive file allows specification of a parameter list that is longer than that allowed in the control statement. With the exception of the MFL parameter, which cannot be specified in a directive file, a parameter can be specified in either the control statement or the directive file. The same parameter cannot, however, be specified in both the control statement and the directive file. Parameters can be specified in any order in the directive file. Any of the parameters, except MFL or DIR, that are valid in the CDCSBTF control statement are also valid in the directive file. Parameters can be specified in the directive file in columns 1 through 80. The first parameter specified in a line must begin in column 1. Parameters can either be placed in separate lines or combined in a line with commas acting as separators. Parameters cannot be split across lines. Blanks, which are ignored, can be used to improve readability.

Parameters

All the parameters (figure G-1) of the CDCSBTF control statement are optional and can be specified in any order. The CDCSBTF parameters provide information for the following functions:

- Allocation of maximum pooled buffer space
- Adjustment of accounting charges
- Allocation of the maximum field length that CDCSBTF is allowed to use
- Control of loading certain CDCS overlay capsules for the duration of CDCS execution
- Control of retention of certain CYBER Record Manager (CRM) capsules
- Specification of information required to attach the master directory if online dumping of journal log files is desired
- Specification of a directive file that can contain any of the CDCSBTF control statement parameters except MFL and DIR

If the CP and IO parameters are specified either in the CDCSBTF control statement or in the directive file, the accounting values returned to the user's dayfile are different from the accounting values returned when the same application executes with CDCS at the system control point. When CDCSBTF executes, the accounting values returned include the application's execution time as well as the central processor and input/output time charged by CDCS.

The parameters available for the CDCSBTF control statement are the same as the parameters available for the CDCS control statement. These parameters are explained in greater detail in the CDCS Data Administrator's reference manual. Refer to this manual for more information.

FILE REQUIREMENTS AND RESTRICTIONS

Certain files must be attached or requested before the CDCSBTF program can be executed. The master directory file must be attached. It can be attached in two ways: either by information provided in the CDCSBTF control statement or directive file or by an ATTACH control statement that precedes execution of CDCSBTF. If the ATTACH control statement is used, the master directory file must be specified with the local file name MSTRDIR; however, using this method of attaching the master directory prevents online dumping of journal log files from being performed by CDCSBTF.

The user programs that are to be run with CDCSBTF must be present in relocatable binary format either as local files or as permanent files.

When CDCS is executing as the Batch Test Facility, the name of the output file produced by CDCS is CDCSOUT. The job name recorded on a log file during execution of CDCSBTF is Cnnnnnn, where nnnnnn is the number associated with the particular user job.

Care must be taken in assigning file names to non-CDCS files when the Batch Test Facility is being used. Because several user programs can be executed during one run of CDCSBTF, every non-CDCS file referenced by the user programs must have a unique name. This restriction is especially critical for the file names INPUT and OUTPUT; they can be used by only one of the programs. Since CDCS cannot enforce this restriction, the user must take particular care in using these names.

Programs running with CDCSBTF cannot use as file names the name MSTRDIR, CDCSOUT, or a name beginning with five Zs; moreover, a name consisting of F, J, P, Q, R, T, or X followed by six digits cannot be used.

CDCSBTF cannot be used to access files that have values specified for the FAMILY clause in the permanent file information subentry of the master directory. To access files that have FAMILY clauses specified in the master directory, the job must be of system origin, but CDCSBTF is not. Refer to the CDCS Data Administration reference manual for a description of the FAMILY clause under the permanent file information subentry, including a way to work around the restriction.

PROCESSING CONSIDERATIONS

The following paragraphs describe processing limitations that apply to CDCS executing as the Batch Test Facility.

In a FORTRAN program executing with the CDCS Batch Test Facility, the Data Manipulation Language (DML) TERMINATE statement must execute before a FORTRAN STOP or END statement. If execution of the FORTRAN program ends without a TERMINATE statement being executed, processing is discontinued for all programs specified in the CDCSBTF control statement that have not completed execution.

The CDCS Batch Test Facility has a limitation on the number of jobs for which abnormal end-of-job processing can be performed. For all the programs specified in the CDCSBTF control statement, the Batch Test Facility allows a total of six concurrent calls to the RECOVR routine. If each of seven or more programs, executing with the CDCS Batch Test Facility, requires a call to the RECOVR routine, the CDCS Batch Test Facility aborts processing. The RECOVR routine issues a diagnostic stating that too many recovery requests were issued.

Both COBOL and FORTRAN 5 provide a mechanism that can eliminate automatic calls to the RECOVR routine and can help prevent the problem of too many recovery requests. Either the TDF parameter in the COBOL5 control statement or the DB parameter in the FTN5 control statement affect end-of-job processing. If either parameter is specified, the RECOVR routine is not automatically called for abnormal end-of-job processing.

LOAD MAPS

Maps of the program loading operations of a program-initiated load can be obtained by setting the sense switches 1 through 4 prior to execution of the CDCSBTF control statement. Each sense switch setting corresponds to different information on the load map. The settings and the associated types of information are shown in table G-1.

TABLE G-1. SENSE SWITCH SETTINGS AND CORRESPONDING LOAD MAP INFORMATION

Setting	Load Map Information
SWITCH,1.	Statistics (S)
SWITCH,2.	Block maps (B)
SWITCH,3.	Entry point maps (E)
SWITCH,4.	Entry point cross reference maps (X)

COMPILATION AND EXECUTION

When the CDCS Batch Test Facility is used to execute a program, the master directory file must be attached at the user control point by one of the methods described previously in the File Requirements and Restrictions subsection. If any independent files are needed by an application program, the necessary control statements must be supplied to attach the files at the user control point.

To compile and execute a COBOL program that utilizes the CDCS Batch Test Facility, the user must perform the following steps:

To compile

Attach the subschema.

Specify the COBOL5 control statement and include the D parameter to indicate the local file name of the subschema.

To execute

Attach the master directory file or specify the information needed to attach the master directory file on the CDCSBTF control statement.

Execute the program by specifying the CDCSBTF control statement.

To compile and execute a FORTRAN program containing DML statements, the user must perform the following steps:

To compile

Attach the subschema.

Specify the DML control statement to execute the DML preprocessor.

Execute the FORTRAN compiler, specifying that the output file produced by the DML preprocessor is the input file to the FORTRAN compiler.

To execute

Attach the master directory file or specify the information needed to attach the master directory file on the CDCSBTF control statement.

Make the DMS-170 library available by specifying the LIBRARY(DMSLIB) control statement.

Execute the program by specifying the CDCSBTF control statement.

The control statements in figure G-2 illustrate sample NOS and NOS/BE jobs in which the CDCS Batch Test Facility is executed for a FORTRAN program. Similar statements, shown in figure G-3, can be used to execute the CDCS Batch Test Facility for a COBOL program. Before the CDCSBTF program is called, the file containing the subschema directory (needed for program compilation) and the file containing the master directory are attached. Additional files are also attached and the desired portions of the load map are selected.

DEADLOCK TESTING

A COBOL program running with the CDCS Batch Test Facility can call the DB$WAIT subroutine to force a deadlock situation and thereby test the program code that deals with recovery from deadlock. The subroutine DB$WAIT should be called only if more than one user program is being run with CDCSBTF. When a program issues a call to DB$WAIT, execution of that program is suspended until all other user programs have also issued calls to DB$WAIT. Once all calls have been issued, the programs are released from suspension.

The DB$WAIT subroutine can be called only from a COBOL program; it cannot be called directly from a FORTRAN program. (Refer to section 2 for more information about the DB$WAIT routine.)

NOS Operating System	NOS/BE Operating System	
Jobname,CMfl.	Jobname,CMfl.	Names the job and specifies maximum field length.
USER control statement		Identifies the user.
CHARGE control statement	ACCOUNT control statement	Specifies the account to which the job's use of system resources is logged.
ATTACH,SUBSC/UN=xxx.	ATTACH,SUBSC,ID=xxx.	Attaches the subschema.
DML,SB=SUBSC.	DML,SB=SUBSC.	Preprocesses the DML statements in the FORTRAN program and writes to DMLOUT.
FTN5,I=DMLOUT.	FTN5,I=DMLOUT.	Compiles the FORTRAN program on DMLOUT and places it on the LGO file.
SWITCH,2.	SWITCH,2.	Requests block map on program-initiated load.
SWITCH,3.	SWITCH,3.	Requests entry point map on program-initiated load.
SWITCH,4.	SWITCH,4.	Requests entry point cross reference map on program-initiated load.
LIBRARY,DMSLIB.	LIBRARY,DMSLIB.	Specifies that library DMSLIB is to be used to satisfy externals.
CDCSBTF,LGO/MDPFN=MSTRDIR,UN=xxx.	CDCSBTF,LGO/MDPFN=MSTRDIR,ID=xxx.	Executes CDCSBTF and passes the master directory permanent file information as parameters..
REWIND,CDCSOUT.	REWIND,CDCSOUT.	Rewinds the CDCSBTF output file.
COPY,CDCSOUT,OUTPUT.	COPY,CDCSOUT,OUTPUT.	Prints the CDCSBTF output file.
CRMEP.	CRMEP.	Prints the CRM error file.
EXIT.	EXIT.	Establishes processing if error occurs.
DMD.	DMP.	Dumps the exchange package.
DMD,377000.	DMP,377000.	Dumps the contents of the field length.
CRMEP.	CRMEP.	Prints the CRM error file.
REWIND,CDCSOUT.	REWIND,CDCSOUT.	Rewinds the CDCSBTF output file.
COPY,CDCSOUT,OUTPUT.	COPY,CDCSOUT,OUTPUT.	Prints the CDCSBTF output file.

Figure G-2. Sample FORTRAN 5 Execution of CDCS Batch Test Facility

NOS Operating System	NOS/BE Operating System	
Jobname,CMfl.	Jobname,CMfl.	Names the job and specifies maximum field length.
USER control statement		Identifies the user.
CHARGE control statement	ACCOUNT control statement	Specifies the account to which the job's use of system resources is logged.
ATTACH,SUBSC/UN=xxx.	ATTACH,SUBSC,ID=xxx.	Attaches the subschema.
COBOL5,D=SUBSC.	COBOL5,D=SUBSC..	Compiles the COBOL program and places it on the LGO file.
SWITCH,2.	SWITCH,2.	Requests block map on program-initiated load.
SWITCH,3.	SWITCH,3.	Requests entry point map on program-initiated load.
SWITCH,4.	SWITCH,4.	Requests entry point cross reference map on program-initiated load.
CDCSBTF,LGO/MDPFN=MSTRDIR,UN=xxx.	CDCSBTF,LGO/MDPFN=MSTRDIR,ID=xxx.	Executes CDCSBTF and passes the master directory permanent file information as parameters.
REWIND,CDCSOUT.	REWIND,CDCSOUT.	Rewinds the CDCSBTF output file.
COPY,CDCSOUT,OUTPUT.	COPY,CDCSOUT,OUTPUT.	Prints the CDCSBTF output file.
CRMEP.	CRMEP.	Prints the CRM error file.
EXIT.	EXIT.	Establishes processing if error occurs.
DMD.	DMP.	Dumps the exchange package.
DMD,177000.	DMP,177000.	Dumps the contents of the field length.
CRMEP.	CRMEP.	Prints the CRM error file.
REWIND,CDCSOUT.	REWIND,CDCSOUT.	Rewinds the CDCSBTF output file.
COPY,CDCSOUT,OUTPUT.	COPY,CDCSOUT,OUTPUT.	Prints the CDCSBTF output file.

Figure G-3. Sample COBOL 5 Execution of CDCS Batch Test Facility

DATA BASE ENVIRONMENT FOR EXAMPLES H

This appendix contains the jobs, which include source programs and control statements, used to generate the data base environment for the manufacturing data base used for examples in sections 6, 7, and 8. Although all jobs reflect operation under the NOS operating system, conversion to the NOS/BE operating system can be accomplished by making the following changes:

Substitute a NOS/BE ACCOUNT control statement for the NOS USER and CHARGE control statements.

Substitute the NOS/BE REQUEST and CATALOG control statements for the NOS DEFINE control statement.

Substitute the NOS/BE file identification parameter ID for the NOS file identification parameter UN.

Omit the PERMIT control statements.

Setting up a DMS-170 data base environment is a responsibility of the data administrator; the process is shown here, however, to allow the reader to duplicate the data base environment and use it.

The source input for the jobs shown in this appendix illustrates the manufacturing data base environment being created by a series of batch jobs. The source input for each job is shown exactly as required for processing on NOS with the exception of the notation for end-of-record (--EOR--). The appropriate end-of-record notation must be used in the actual job submitted for execution.

The steps the data administrator takes to establish the data base environment are listed in the appropriate order as follows:

1. Design, write, and compile a schema and store the resulting schema directory as a permanent file. A schema named MANUFACTURING-DB is stored as a permanent file named MANUFAC. Refer to figure H-1.

2. Design, write, and compile subschemas and store resulting subschema directories in permanent files as subschema libraries. Subschema libraries contain subschemas as follows:

   C5SSLIB contains a COBOL subschema named C5SS-PRODUCT-MANAGEMENT. Refer to figure H-2.

   F5SSLIB contains a FORTRAN 5 subschema named F5S-PRODUCT-MANAGEMENT. Refer to figure H-3.

   QUSSLIB contains two Query Update subschemas: subschema QUCREA6 and subschema QUPRODMGT. Refer to figure H-4.

3. Write, compile, and store the master directory as a permanent file. The DBMSTRD utility is used to generate the master directory. Refer to figure H-5.

4. Initialize log or recovery files specified for the schema in the master directory. The master directory specifies a transaction recovery file and a restart identifier file for schema MANUFACTURING-DB. These files are required to support application program requests that are associated with data base transactions and program restart operations. The DBREC utility is used to initialize these files. Refer to figure H-6.

   The PERMIT control statements shown in the job apply only to NOS. In the example, they make the recovery files specified available to CDCS when it executes under user number DBCNTLX.

5. Create the data base. A Query Update job using subschema QUCREA6 creates the data base. Refer to figure H-7. The job establishes all the permanent files associated with data base areas and provides data for six areas, including the area associated with the Query Update catalog file.

   CDCS must be active to run this job. For NOS, the permanent file for each area and index file is established by the DEFINE and RETURN control statements; access to the file by CDCS is established by the PERMIT control statement as described in step 4. For NOS/BE, the following series of control statements must be used to establish each area or index file: REQUEST, REWIND, CATALOG, and RETURN.

6. Establish CDCS as an active system. This must be done by the data administrator; the process is not shown in this manual.

```
Job statement
USER control statement
CHARGE control statement
DEFINE,MANUFAC.
FILE(EMPLOYE,FO=IS,XN=IXEMP)
FILE(JOBDETA,FO=IS)
FILE(DEPARTM,FO=IS,XN=IXDEPT,RT=T)
FILE(PROJECT,FO=DA,XN=IXPROJ,HMB=7)
FILE(DEVELOP,FO=DA,XN=IXDEV,HMB=11)
FILE(TESTS,FO=IS,XN=IXTEST)
FILE(CDCSCAT,FO=IS)
DDL3,DS,SC=MANUFAC.
--EOR--
SCHEMA NAME IS MANUFACTURING-DB.

AREA IS EMPLOYEE
   ACCESS-CONTROL LOCK FOR RETRIEVAL IS "EMP-READ"
   ACCESS-CONTROL LOCK FOR UPDATE IS "EMP-WRITE".

RECORD IS EMPREC WITHIN EMPLOYEE.
   01  EMP-ID                  PICTURE "X(8)".
   01  SALARY                  TYPE FIXED DECIMAL 8,2.
   01  EMP-LAST-NAME           PICTURE "A(20)".
   01  EMP-INITIALS            PICTURE "A(4)".
   01  DEPT                    PICTURE "X(4)".
   01  ADDRESS-NUMBERS         PICTURE "X(6)".
   01  ADDRESS-STREET          PICTURE "X(20)".
   01  ADDRESS-CITY            PICTURE "A(15)".
   01  ADDRESS-STATE-PROV      PICTURE "A(15)".
   01  POSTAL-CODE             PICTURE "X(10)".
   01  ADDRESS-COUNTRY         PICTURE "A(15)".
   01  PHONE-NO                PICTURE "9(10)".
   01  HIRE-DATE               PICTURE "9(6)".
   01  INSURANCE-NO            PICTURE "9(10)".
   01  NUM-DEPENDENTS          TYPE DECIMAL FIXED 2.
   01  JOB-CLASS               PICTURE "A".
   01  GRADE-LEVEL             PICTURE "9"
                                   CHECK VALUE 0 THRU 8.
/*    THE FOLLOWING ARE YEAR-TO-DATE TOTALS
*/
   01  GROSS                   TYPE DECIMAL FIXED 7,2.
   01  FED-TAX                 TYPE DECIMAL FIXED 7,2.
   01  STATE-TAX               TYPE DECIMAL FIXED 6,2.
   01  DISABILITY              TYPE DECIMAL FIXED 5,2.
   01  SS-INSURANCE            TYPE DECIMAL FIXED 4,4.


AREA IS JOBDETAIL
   ACCESS-CONTROL LOCK IS "ABCDEFZ".
RECORD IS JOBREC WITHIN JOBDETAIL.
   01  EMP-ID                  PICTURE "X(8)".
   01  SEQ-NO                  PICTURE "X(4)".
   01  PRODUCT-ID              PICTURE "X(10)".
   01  SECURITY-CODE           PICTURE "X(2)".
   01  PROJECT-ID              PICTURE "X(10)".
   01  MONTHLY-COMPENSATION    OCCURS 12 TIMES.
       02  REG-HOURS           TYPE DECIMAL FIXED 5,2.
       02  REG-COMPENSATION    TYPE DECIMAL FIXED 6,2.
       02  OT-HOURS            TYPE DECIMAL FIXED 5,2.
       02  OT-COMPENSATION     TYPE DECIMAL FIXED 6,2.
   01  HOURS-YTD               TYPE DECIMAL FIXED 6,2.
   01  COMPENSATION-YTD        TYPE DECIMAL FIXED 10,2.
   01  START-DATE              PICTURE "9(6)".
   01  LOC-CODE                PICTURE "X(4)".
   01  ACTUAL-DATE             PICTURE "X(6)".
```

Figure H-1. Schema Generation (Sheet 1 of 3)

```
AREA IS DEPARTMENTS
   ACCESS-CONTROL LOCK IS "VERY*PRIVATE".
RECORD IS DEPTREC WITHIN DEPARTMENTS.
   01  DEPT-NO                  PICTURE "X(4)".
   01  DEPT-NAME                PICTURE "X(20)".
   01  MGR-ID                   PICTURE "X(8)".
   01  MGR-NAME                 PICTURE "X(20)".
   01  NUM-ITEM                 PICTURE "9(3)"
                                   CHECK VALUE 5 THRU 25.
   01  ITEM                     OCCURS NUM-ITEM TIMES.
       02  LOC-CODE             PICTURE "X(4)".
       02  HEAD-COUNT           PICTURE "9(4)".
       02  EXPENSES-YTD         PICTURE "9(8)V99T".
       02  BUDGET               PICTURE "9(8)T".


AREA IS PROJECT
   ACCESS-CONTROL LOCK FOR RETRIEVAL IS "VERIFIED-INPUT"
   ACCESS-CONTROL LOCK FOR UPDATE IS "OKAYED-OUTPUT".
RECORD IS PROJREC WITHIN PROJECT.
   01  PROJECT-ID               PICTURE "X(10)".
   01  PROJ-DESCR               PICTURE "X(40)".
   01  BUDGET-TOTAL             PICTURE "9(9)V99".
   01  MONTHLY-BUDGET           PICTURE "9(7)V99" OCCURS 12 TIMES.
   01  SCHED-COMPLETE           PICTURE "X(10)".
   01  RESPONSIBILITY           PICTURE "X(8)".


AREA IS DEVELOPMENT-PRODUCTS
   ACCESS-CONTROL LOCK IS "ACCESS(/)OK".
RECORD IS DEVREC WITHIN DEVELOPMENT-PRODUCTS.
   01  PRODUCT-ID               PICTURE "X(10)".
   01  PRODUCT-DESCR            PICTURE "X(20)".
   01  CLASS                    PICTURE "9(2)"
                                   CHECK VALUE 0 THRU 99.
   01  PRICE                    TYPE DECIMAL FIXED 5,2.
   01  EVAL-ID                  TYPE CHARACTER 20 OCCURS 10 TIMES.
   01  PROJECT-ID               PICTURE "X(10)".
   01  NUM-TESTED               TYPE DECIMAL FIXED 4.
   01  CUM-TEST-AVERAGE         TYPE FLOAT.
   01  STATUS-CODE              PICTURE "A".
   01  SECURITY-CODE            PICTURE "X(2)".
   01  EST-COST                 PICTURE "9(4)PPP".
   01  DEV-COST-YTD             TYPE DECIMAL FIXED 9,2.


AREA IS TESTS
   ACCESS-CONTROL LOCK FOR UPDATE IS "UP"
   ACCESS-CONTROL LOCK FOR RETRIEVAL IS "DOWN".
RECORD IS TESTREC WITHIN TESTS.
       TESTNO                    TYPE DECIMAL FIXED.
       TNAME                     TYPE CHARACTER 20.
       PRDCTNO                   TYPE CHARACTER 10.
       TESTER                    TYPE DECIMAL FIXED 8.
       TOTALCT                   TYPE DECIMAL FIXED 4.
       N                         TYPE DECIMAL FIXED 3
                                   CHECK VALUE 0 THRU 100.
       PASPROB                   TYPE FLOAT OCCURS 100 TIMES
                                   CHECK VALUE 0.0 THRU 1.0.


AREA NAME IS CDCSCAT
ACCESS-CONTROL LOCK IS "PERMISSION*GRANTED".
RECORD IS QUCATREC WITHIN CDCSCAT.
       QUCAT-KEY                 PICTURE "X(10)".
       QUCAT-ITEM                PICTURE "X(1030)".
```

Figure H-1. Schema Generation (Sheet 2 of 3)

```
DATA CONTROL.

AREA NAME IS EMPLOYEE
       KEY IS EMP-ID OF EMPREC
           DUPLICATES ARE NOT ALLOWED
       KEY IS ALTERNATE DEPT
           DUPLICATES ARE INDEXED.

AREA NAME IS JOBDETAIL
       KEY ID IS CONCATKEY < EMP-ID OF JOBREC
                             SEQ-NO >
           DUPLICATES ARE NOT ALLOWED.

AREA NAME IS DEPARTMENTS
       KEY IS DEPT-NO
           DUPLICATES ARE NOT ALLOWED
       KEY IS ALTERNATE MGR-ID
           DUPLICATES ARE FIRST.

AREA NAME IS PROJECT
       KEY IS PROJECT-ID OF PROJREC
           DUPLICATES ARE NOT ALLOWED
       KEY IS ALTERNATE RESPONSIBILITY
           DUPLICATES ARE ALLOWED.

AREA NAME IS DEVELOPMENT-PRODUCTS
       KEY IS PRODUCT-ID OF DEVREC
           DUPLICATES ARE NOT ALLOWED
       KEY IS ALTERNATE PROJECT-ID OF DEVREC
           DUPLICATES ARE ALLOWED
       KEY IS ALTERNATE EVAL-ID
           DUPLICATES ARE INDEXED.

AREA NAME IS TESTS
       KEY IS TESTNO
           DUPLICATES ARE NOT ALLOWED
       KEY IS ALTERNATE TNAME
           DUPLICATES ARE INDEXED
       KEY IS ALTERNATE TESTER
           DUPLICATES ARE FIRST
       SEQUENCE IS ASCII.

AREA NAME IS CDCSCAT
       KEY IS QUCAT-KEY OF QUCATREC
           DUPLICATES ARE NOT ALLOWED
       SEQUENCE IS DISPLAY.


CONSTRAINT NAME IS MGR-CONST
           MGR-ID OF DEPTREC DEPENDS ON EMP-ID OF EMPREC.

CONSTRAINT NAME IS PROJ-CONST
           PROJECT-ID OF DEVREC DEPENDS ON PROJECT-ID OF PROJREC.


RELATION NAME IS EMP-REL
       JOIN WHERE EMP-ID OF JOBREC EQ EMP-ID OF EMPREC.

RELATION NAME IS TEST-REL
       JOIN WHERE PRDCTNO OF TESTREC EQ PRODUCT-ID OF DEVREC.

RELATION NAME IS DPD-REL
       JOIN WHERE MGR-ID OF DEPTREC EQ RESPONSIBILITY OF PROJREC
                  PROJECT-ID OF PROJREC EQ PROJECT-ID OF DEVREC.
```

Figure H-1. Schema Generation (Sheet 3 of 3)

```
Job statement
USER control statement
CHARGE control statement
DEFINE,C5SSLIB.
ATTACH,MANUFAC.
DDL3,C5,SB=C5SSLIB,SC=MANUFAC.
--EOR--
       TITLE DIVISION.
           SS  C5SS-PRODUCT-MANAGEMENT WITHIN MANUFACTURING-DB.

       ALIAS DIVISION.
           AD  REALM DEVELOPMENT-PRODUCTS BECOMES PRODUCTS.
           AD  RECORD DEVREC BECOMES PRODREC.
           AD  DATA CUM-TEST-AVERAGE BECOMES CUMULATIVE-AVERAGE.

       REALM DIVISION.
           RD DEPARTMENTS, PROJECT, PRODUCTS.

       RECORD DIVISION.
       01  DEPTREC.
           03  DEPT-NO                 PICTURE X(4).
           03  DEPT-NAME               PICTURE X(20).
           03  MGR-ID                  PICTURE X(8).
           03  MGR-NAME                PICTURE X(20).

       01  PROJREC.
           03  PROJECT-ID              PICTURE X(10).
           03  PROJ-DESCR              PICTURE X(40).
           03  RESPONSIBILITY          PICTURE X(8).

       01  PRODREC.
           03  PRODUCT-ID              PICTURE X(10).
           03  PROJECT-ID              PICTURE X(10).
           03  STATUS-CODE             PICTURE A.

       RELATION DIVISION.
           RN IS DPD-REL
               RESTRICT PRODREC WHERE STATUS-CODE EQ "A"
                                  OR  STATUS-CODE EQ "N".
```

Figure H-2. COBOL Subschema Generation

```
Job statement.
USER control statement
CHARGE control statement
DEFINE,F5SSLIB.
ATTACH,MANUFAC.
DDLF,F5,SB=F5SSLIB,SC=MANUFAC.
--EOR--
      SUBSCHEMA F5SS-PRODUCT-MANAGEMENT, SCHEMA=MANUFACTURING-DB

      ALIAS (ITEM) DEPTNO = DEPT-NO
      ALIAS (ITEM) DEPTNAM = DEPT-NAME
      ALIAS (ITEM) MGRID = MGR-ID
      ALIAS (ITEM) MGRNAM = MGR-NAME
      ALIAS (ITEM) PROJID = PROJECT-ID.PROJREC
      ALIAS (ITEM) PROJDES = PROJ-DESCR
      ALIAS (ITEM) BUDGTOT = BUDGET-TOTAL
      ALIAS (REALM) BOSS = RESPONSIBILITY
      ALIAS (REALM) PRODUCT-FILE = DEVELOPMENT-PRODUCTS
      ALIAS (ITEM) PRODUCT = PRODUCT-ID
      ALIAS (ITEM) PRODES = PRODUCT-DESCR
      ALIAS (ITEM) PROJECT = PROJECT-ID.DEVREC
      ALIAS (ITEM) STATUS = STATUS-CODE
      ALIAS (ITEM) YTDCOST = DEV-COST-YTD

      REALM DEPARTMENTS, PROJECT, PRODUCT-FILE

      RECORD DEPTREC
      CHARACTER DEPTNO *4, DEPTNAM *20
      CHARACTER MGRID *8, MGRNAM *20

      RECORD PROJREC
      CHARACTER PROJID *10, PROJDES *40
      REAL BUDGTOT

      RECORD DEVREC
      CHARACTER PRODUCT *10, PRODES *20
      CHARACTER PROJECT *10
      CHARACTER STATUS *1
      REAL YTDCOST

      RELATION DPD-REL
      RESTRICT DEVREC (STATUS .EQ. "A" .OR. STATUS .EQ. "R")

      END
```

Figure H-3. FORTRAN 5 Subschema Generation

```
Job statement
USER control statement
CHARGE control statement
DEFINE,QUSSLIB.
ATTACH,MANUFAC.
DDL3,QC,SB=QUSSLIB,SC=MANUFAC.
--EOR--
       TITLE DIVISION.
           SS  QUCREA6 WITHIN MANUFACTURING-DB.

       ALIAS DIVISION.
           AD  REALM DEPARTMENTS BECOMES DEPTAREA.
           AD  REALM DEVELOPMENT-PRODUCTS BECOMES DEVAREA.

       REALM DIVISION.
           RD  EMPLOYEE, DEPTAREA, PROJECT.
           RD  DEVAREA, TESTS, CDCSCAT.

       RECORD DIVISION.
       01  EMPREC.
           03  EMP-ID                  PICTURE X(8).
           03  EMP-LAST-NAME           PICTURE A(20).
           03  EMP-INITIALS            PICTURE A(4).
           03  DEPT                    PICTURE X(4).

       01  DEPTREC.
           03  DEPT-NO                 PICTURE X(4).
           03  DEPT-NAME               PICTURE X(20).
           03  MGR-ID                  PICTURE X(8).
           03  MGR-NAME                PICTURE X(20).
           03  NUM-ITEM                PICTURE 9(3).
           03  ITEM                    OCCURS 2 TO 25 TIMES
                                           DEPENDING ON NUM-ITEM.
               05  LOC-CODE            PICTURE X(4).
               05  HEAD-COUNT          PICTURE Z(4).
               05  EXPENSES-YTD        PICTURE Z(8).99.
               05  BUDGET              PICTURE Z(9).

       01 PROJREC.
           03  PROJECT-ID              PICTURE X(10).
           03  PROJ-DESCR              PICTURE X(40).
           03  BUDGET-TOTAL            PICTURE Z(9).99.
           03  MONTHLY-BUDGET          PICTURE Z(7).99
                                           OCCURS 12 TIMES.
           03  SCHED-COMPLETE          PICTURE X(10).
           03  RESPONSIBILITY          PICTURE X(8).

       01  DEVREC.
           03  PRODUCT-ID              PICTURE X(10).
           03  CLASS                   PICTURE Z9.
           03  PRICE                   PICTURE Z(5).99
                                           USAGE IS COMP-1.
           03  PROJECT-ID              PICTURE X(10).
           03  NUM-TESTED              PICTURE ZZ9
                                           USAGE IS COMP-1.
           03  CUM-TEST-AVERAGE        PICTURE Z(7)9.9(3)
                                           USAGE IS COMP-2.
           03  STATUS-CODE             PICTURE A.
           03  DEV-COST-YTD            PICTURE Z(8)9.99
                                           USAGE IS COMP-1.
```

Figure H-4. Query Update Subschema Generation (Sheet 1 of 2)

```
01 TESTREC.
    03  TESTNO                 PICTURE 9(14)
                                    USAGE IS COMP-1.
    03  TNAME                  PICTURE X(20).
    03  PRDCTNO                PICTURE X(10).
    03  TESTER                 PICTURE 9(8)
                                    USAGE IS COMP-1.
    03  TOTALCT                PICTURE 9(4)
                                    USAGE IS COMP-1.
    03  N                      PICTURE 9(3)
                                    USAGE IS COMP-1.
    03  PASPROB                PICTURE 9.9999
                                    OCCURS 100 TIMES
                                    USAGE IS COMP-2.
01  QUCATREC.
    03  QUCAT-KEY              PICTURE X(10).
    03  QUCAT-ITEM             PICTURE X(1030).


TITLE DIVISION.
    SS  QUPRODMGT WITHIN MANUFACTURING-DB.

ALIAS DIVISION.
    AD  REALM DEVELOPMENT-PRODUCTS BECOMES PRODAREA.
    AD  RECORD DEVREC BECOMES PRODREC.
    AD  DATA CUM-TEST-AVERAGE BECOMES CUMULATIVE-AVERAGE.
    AD  REALM DEPARTMENTS BECOMES DEPTAREA.

REALM DIVISION.
    RD  DEPTAREA, PROJECT, PRODAREA, CDCSCAT.

RECORD DIVISION.
01  DEPTREC.
    03  DEPT-NO                PICTURE X(4).
    03  DEPT-NAME              PICTURE X(20).
    03  MGR-ID                 PICTURE X(8).
    03  MGR-NAME               PICTURE X(20).

01  PROJREC.
    03  PROJECT-ID             PICTURE X(10).
    03  PROJ-DESCR             PICTURE X(40).
    03  BUDGET-TOTAL           PICTURE Z(9).99.
    03  RESPONSIBILITY         PICTURE X(8).

01  PRODREC.
    03  PRODUCT-ID             PICTURE X(10).
    03  CLASS                  PICTURE Z9.
    03  PRICE                  PICTURE Z(5).99
                                    USAGE IS COMP-1.
    03  PROJECT-ID             PICTURE X(10).
    03  STATUS-CODE            PICTURE A.
    03  DEV-COST-YTD           PICTURE Z(8)9.99
                                    USAGE IS COMP-1.

01  QUCATREC.
    03 QUCAT-KEY               PICTURE X(10).
    03 QUCAT-ITEM              PICTURE X(1030).

RELATION DIVISION.
    RN IS DPD-REL
        RESTRICT PRODREC WHERE STATUS-CODE EQ "A".
```

Figure H-4. Query Update Subschema Generation (Sheet 2 of 2)

```
Job statement
USER control statement
CHARGE control statement
DEFINE,MSTRDIR.
PERMIT,MSTRDIR,DBCNTLX=W.
ATTACH,MANUFAC.
ATTACH,C5SSLIB.
ATTACH,F4SSLIB.
ATTACH,F5SSLIB.
ATTACH,QUSSLIB.
DBMSTRD(NMD=MSTRDIR,LD)
--EOR--
SCHEMA NAME IS MANUFACTURING-DB
        FILE NAME IS MANUFAC


        TRANSACTION RECOVERY FILE
            PFN IS "DB1TRF" UN IS "CDCS23"
            UNIT LIMIT IS 50
            UPDATE LIMIT IS 15
        RESTART IDENTIFIER FILE
            PFN IS "DB1RIF" UN IS "CDCS23"
        JOB CONTROL INFORMATION
            TAPE TYPE IS NT
                 DENSITY IS PE
            UN IS "CDCS23"
            CHARGE IS "1982CHG".

VERSION NAME IS MASTER

    AREA NAME IS EMPLOYEE
        PFN IS "MEMPL" UN IS "CDCS23"
            PW IS "OKCDCS2"
        INDEX FILE ASSIGNED
            PFN "MXEMPL" UN IS "CDCS23".

    AREA NAME IS JOBDETAIL
        PFN IS "MJOBD" UN IS "CDCS23"
            PW IS "OKCDCS2".

        AREA NAME IS DEPARTMENTS
            PFN IS "MDEPT" UN IS "CDCS23"
            INDEX FILE ASSIGNED
                PFN "MXDEPT" UN IS "CDCS23".

        AREA NAME IS PROJECT
            PFN IS "MPROJ" UN IS "CDCS23"
            INDEX FILE ASSIGNED
                PFN "MXPROJ" UN IS "CDCS23".

        AREA NAME IS DEVELOPMENT-PRODUCTS
            PFN IS "MDEVE" UN IS "CDCS23"
            INDEX FILE ASSIGNED
                PFN IS "MXDEVE" UN IS "CDCS23".

        AREA IS TESTS
            PFN IS "MTEST" UN IS "CDCS23"
            INDEX FILE ASSIGNED
                PFN "MXTEST" UN IS "CDCS23".

        AREA NAME IS CDCSCAT
            PFN IS "MQCAT" UN IS "CDCS23".

    VERSION NAME IS BRANCH1.
    VERSION NAME IS BRANCH2.
    VERSION NAME IS BRANCH3.
    VERSION NAME IS BRANCH4.

    SUBSCHEMA NAME IS PRODUCT-PERSONNEL
            FILE NAME IS C5SSLIB.
    SUBSCHEMA NAME IS C5SS-PRODUCT-MANAGEMENT
            FILE NAME IS C5SSLIB.
    SUBSCHEMA NAME IS F4SS-PRODUCT-EVALUATION
            FILE NAME IS F4SSLIB.
    SUBSCHEMA NAME IS F5SS-PRODUCT-MANAGEMENT
            FILE NAME IS F5SSLIB.
    SUBSCHEMA NAME IS QUCREA6
            FILE NAME IS QUSSLIB.
    SUBSCHEMA NAME IS QUPRODMGT
            FILE NAME IS QUSSLIB.
```

Figure H-5. Master Directory Generation

```
Job statement
USER control statement
CHARGE control statement
DEFINE,DB1TRF1.
DEFINE,DB1RIF.
PERMIT,DB1TRF1,DBCNTLX=W.
PERMIT,DB1RIF,DBCNTLX=W.
RETURN,DB1TRF1.
RETURN,DB1RIF.
ATTACH,MSTRDIR.
DBREC.
--EOR--
    SCHEMA NAME IS MANUFACTURING-DB
    ALLOCATE
    TRANSACTION RECOVERY FILE IS DB1TRF1
    RESTART IDENTIFIER FILE IS DB1RIF
```

Figure H-6. Recovery File Initialization by DBREC

```
Job statement
USER control statement
CHARGE control statement
COMMENT.  CREATING DB AREA FILES
DEFINE(MEMPL/PW=OKCDCS2,M=W)
DEFINE(MXEMPL/M=W)
RETURN(MEMPL,MXEMPL)
DEFINE(MDEPT,MXDEPT/M=W)
RETURN(MDEPT,MXDEPT)
DEFINE(MPROJ,MXPROJ/M=W)
RETURN(MPROJ,MXPROJ)
DEFINE(MDEVE,MXDEVE/M=W)
RETURN(MDEVE,MXDEVE)
DEFINE(MTEST,MXTEST/M=W)
RETURN(MTEST,MXTEST)
DEFINE(MQCAT/M=W)
RETURN(MQCAT)
PERMIT,MEMPL,DBCNTLX=W.
PERMIT,MXEMPL,DBCNTLX=W.
PERMIT,MDEPT,DBCNTLX=W.
PERMIT,MXDEPT,DBCNTLX=W.
PERMIT,MPROJ,DBCNTLX=W.
PERMIT,MXPROJ,DBCNTLX=W.
PERMIT,MDEVE,DBCNTLX=W.
PERMIT,MXDEVE,DBCNTLX=W.
PERMIT,MTEST,DBCNTLX=W.
PERMIT,MXTEST,DBCNTLX=W.
PERMIT,MQCAT,DBCNTLX=W.
COMMENT.  CONTROL STATEMENTS FOR QU JOB
QU.
QU.
ATTACH,MQCAT/M=W.
REWIND,ZZZZZQ2.
COPY,ZZZZZQ2,MQCAT.
--EOR--
CREATE EMPLOYEE OF QUCREA6 FROM LIBRARY QUSSLIB (UN=CDCS23)
ACCESS KEY IS $EMP-WRITE$ ON OUTPUT FOR AREA EMPLOYEE
ACCESS KEY IS $EMP-READ$ ON INPUT FOR AREA EMPLOYEE
STORE SETTING EMP-ID, EMP-LAST-NAME, EMP-INITIALS, DEPT
$1460BUNS$ $BUNSEN$ $I.A.$  $M890$
$2330FIND$ $FINDER$ $R.H.$ $M200$
$3650HOWE$ $HOWE$ $L.C.$ $M210$
$4540MENT$ $MENTOR$ $X.P.$ $M570$
$5730GOOD$ $GOODE$ $S.A.$ $M0680$
$6930CARP$ $CARPENTERR$ $B.L.$ $M130$
$7210BYER$ $BYERS$ $C.B.$ $M010$
*END
CREATE DEPTAREA OF QUCREA6 LIBRARY QUSSLIB (UN=CDCS23)
ACCESS KEY IS $VERY*PRIVATE$ ON I-O FOR AREA DEPTAREA
STORE SETTING DEPT-NO, DEPT-NAME, MGR-ID, MGR-NAME, NUM-ITEM
$M010$ $PURCHASING$ $7210BYER$ $C. B. BYERS$ 8
$M130$ $PACKAGING-DESIGN$ $6930CARP$ $B. L. CARPENTER$ 5
$M200$ $RESEARCH$ $2330FIND$ $R. H. FINDER$ 7
$M210$ $DEVELOPMENT$ $3650HOWE$ $L. C. HOWE$ 7
$M570$ $TESTING-EVALUATION$ $4540MENT$ $X. P. MENTOR$ 13
$M680$ $QUALITY CONTROL$ $5730GOOD$ $S. A. GOODE$ 7
$M890$ $CHEM LAB$ $1460BUNS$ $I. A. BUNSEN$ 9
*END
CREATE PROJECT OF QUCREA6 FROM LIBRARY QUSSLIB (UN=CDCS23)
ACCESS KEY $OKAYED-OUTPUT$ ON OUTPUT FOR AREA PROJECT
STORE SETTING PROJECT-ID OF PROJREC, PROJ-DESCR, +
    RESPONSIBILITY, BUDGET-TOTAL
$M130001560$ $ADVANCED BIG BOX$ $6930CARP$ 143000.
$M130001720$ $ADVANCED THERMAL PROTECTOR$ $6930CARP$ 105000.
$M200001570$ $ADVANCED NETWORK$ $2330FIND$ 275000.
$M200001590$ $ARCTIC AGRICULTURE$ $2330FIND$ 192500.
$M210001320$ $GAMMA SPOOK SUPPORT$ $3650HOWE$ 55000.
$M210001322$ $BETA SUPPORT$ $3650HOWE$ 85000.
$M210002540$ $EPSILON SPOOK DEVICES$ $3650HOWE$ 95000.
$M890001550$ $ADVANCED BONDING AGENT-99$ $1460BUNS$ 130000.
$M890001720$ $GENETIC ENGINEERING: NITROGEN FIXING$ $1460BUNS$ 258000.
*END
```

Figure H-7. Data Base File Creation by Query Update Application (Sheet 1 of 3)

```
INVOKE QUCREA6 FROM LIBRARY QUSSLIB (UN=CDCS23)
ACCESS KEY $OKAYED-OUTPUT$ ON OUTPUT FOR AREA PROJECT
ACCESS KEY IS $VERIFIED-INPUT$ ON INPUT FOR AREA PROJECT
MODIFY USING PROJECT-ID OF PROJREC SETTING SCHED-COMPLETE
$M130001560$ $84/06/31$
$M130001720$ $83/12/20$
$M200001570$ $85/07/30$
$M200001590$ $84/09/30$
$M210001320$ $84/02/15$
$M210001322$ $85/08/25$
$M210002540$ $83/10/01$
$M890001550$ $83/03/15$
$M890001720$ $84/11/01$
*END
CREATE DEVAREA OF QUCREA6 FROM LIBRARY QUSSLIB (UN=CDCS23)
ACCESS KEY IS $ACCESS(/)OK$ ON I-O FOR AREA DEVAREA
STORE SETTING PRODUCT-ID, CLASS, PROJECT-ID OF DEVREC, STATUS-CODE, +
    DEV-COST-YTD
$826NAMW019$ 02 $M130001560$ $N$ 4590.
$7684GRD028$ 03 $M130001560$ $R$ 5555.
$537KLPN037$ 03 $M130001560$ $N$ 2030.
$432DRTF043$ 03 $M130001720$ $A$ 4580.
$025CBLE055$ 04 $M130001720$ $A$ 30500.
$826NAMW069$ 08 $M200001570$ $N$ 10850.
$826NAMW070$ 08 $M200001570$ $N$ 12850.
$537KLPN077$ 01 $M200001570$ $A$ 950.
$537KLPN078$ 01 $M200001570$ $A$ 12250.
$537KLPN079$ 02 $M200001570$ $A$ 14400.
$567CRTX081$ 06 $M200001570$ $R$ 3678.
$567CRTX881$ 06 $M200001570$ $R$ 3678.
$567CRTX882$ 07 $M200001570$ $N$ 5400.
$567LINE094$ 07 $M200001570$ $N$ 9400.
$387ARAG322$ 08 $M200001590$ $A$ 35000.
$387ARAG323$ 08 $M200001590$ $N$ 8580.
$387ARAG555$ 06 $M200001590$ $R$ 7500.
$693GSPK020$ 14 $M210001320$ $A$ 2050.
$693GSPK022$ 13 $M210001320$ $A$ 8050.
$280BSPK910$ 10 $M210001322$ $N$ 20500.
$280BSPK950$ 08 $M210001322$ $R$ 10220.
$466EPSD311$ 06 $M210002540$ $A$ 54000.
$138CBND926$ 03 $M890001550$ $A$ 33330.
$138CBND930$ 05 $M890001550$ $A$ 13000.
$138CBND940$ 04 $M890001550$ $A$ 1050.
$268GENE088$ 11 $M890001720$ $R$ 164000.
*END*
CREATE TESTS OF QUCREA6 FROM LIBRARY QUSSLIB (UN=CDCS23)
ACCESS KEY IS $UP$ ON OUTPUT FOR AREA TESTS
ACCESS KEY IS $DOWN$ ON INPUT FOR AREA TESTS
STORE SETTING TESTNO,TNAME,PRDCTNO,TOTALCT,N,PASPROB(ALL)
32555 $LINE-TESTS$ $537KLPN078$  45 100 +
    0.95 0.97 0.99 0.94 0.99 0.98 0.92 0.88 0.92 0.99 +
    0.88 0.99 0.94 0.93 0.99 0.88 0.98 0.87 0.99 0.93 +
    0.97 0.99 0.87 0.99 0.95 0.89 0.92 0.90 0.92 0.88 +
    0.99 0.94 0.92 0.94 0.99 0.91 0.96 0.99 0.91 0.84 +
    0.87 0.88 0.86 0.98 0.99 0.93 0.93 0.92 0.98 0.99 +
    0.99 0.87 0.90 0.97 0.84 0.89 0.90 0.99 0.99 0.93 +
    0.89 0.92 0.85 0.92 0.98 0.99 0.98 0.95 0.88 0.92 +
    0.85 0.76 0.94 0.92 0.99 0.97 0.92 0.98 0.92 0.95 +
    0.87 0.96 0.99 0.92 0.94 0.90 0.83 0.88 0.92 0.98 +
    0.89 0.89 0.99 0.93 0.95 0.92 0.98 0.95 0.99 0.97
18961 $QUALITY-TESTS$ $537KLPN079$  20 15  +
    0.89 0.85 0.72 0.93 0.84 0.93 0.98 0.95 0.88 0.97 +
    0.98 0.65 0.78 0.65 0.93 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
```

Figure H-7. Data Base File Creation by Query Update Application (Sheet 2 of 3)

```
34347 $SMALLPARTS-TEST$ $466EPSD311$ 14 35 +
    0.89 0.88 0.72 0.80 0.71 0.96 0.98 0.95 0.88 0.99 +
    0.98 0.98 0.99 0.93 0.88 0.92 0.91 0.98 0.97 0.83 +
    0.88 0.93 0.97 0.88 0.94 0.99 0.98 0.91 0.99 0.99 +
    0.89 0.94 0.72 0.80 0.71 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
25100 $CHEMICAL-TEST$ $138CBND926$ 134 31  +
    0.89 0.68 0.72 0.80 0.71 0.84 0.98 0.95 0.88 0.66 +
    0.89 0.77 0.98 0.99 0.83 0.52 0.98 0.93 0.99 0.97 +
    0.83 0.96 0.95 0.92 0.98 0.84 0.86 0.83 0.97 0.97 +
    0.50 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
93447 $PACKAGE-TEST$ $826NAMW019$ 114 34 +
    0.89 0.99 0.93 0.96 0.92 0.83 0.73 0.82 0.88 0.99 +
    0.87 0.73 0.93 0.96 0.94 0.92 0.91 0.99 0.99 0.97 +
    0.60 0.71 0.67 0.99 0.91 0.90 0.90 0.92 0.97 0.78 +
    0.77 0.92 0.93 0.99 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
25300 $COLOR-TEST$ $826NAMW070$ 22 18 +
    0.65 0.66 0.78 0.64 0.60 0.66 0.97 0.85 0.67 0.74 +
    0.93 0.92 0.98 0.88 0.79 0.87 0.74 0.92 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
10777 $PRICE-TEST$ $025CBLE055$ 18 43 +
    0.50 0.44 0.47 0.42 0.68 0.66 0.52 0.73 0.91 0.49 +
    0.54 0.77 0.33 0.89 0.93 0.74 0.69 0.99 0.97 0.51 +
    0.55 0.68 0.97 0.54 0.78 0.46 0.97 0.99 0.69 0.93 +
    0.77 0.77 0.98 0.93 0.76 0.44 0.87 0.75 0.99 0.93 +
    0.99 0.97 0.92 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
85475 $ASSEMBLY-TEST$ $537KLPN078$ 180 26 +
    0.93 0.99 0.98 0.93 0.95 0.98 0.99 0.96 0.94 0.99 +
    0.89 0.98 0.97 0.91 0.89 0.95 0.93 0.98 0.99 0.96 +
    0.96 0.99 0.98 0.93 0.92 0.98 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 +
    0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
*END
END
--EOR--
RECORDING DUMMY
DISPLAY DUMMY
RECORDING OFF
*END
END
```

Figure H-7. Data Base File Creation by Query Update Application (Sheet 3 of 3)

COLLATING SEQUENCES FOR DATA BASE FILES

The collating sequence of a data base file is determined by the area control entry in the schema. The following collating sequences can be specified: ASCII, COBOL, and DISPLAY (also called the FORTRAN collating sequence). If no collating sequence is specified in the schema area control entry, the COBOL collating sequence is the default for the file. The collating sequence specified for a specific data base area can be obtained from the data administrator. Collating sequences are shown in table I-1.

The collating sequence specified for a file affects:

The order in which items are retrieved when a file is accessed sequentially.

The order in which items are returned by alternate key retrieval for all file organizations if duplicate alternate keys are allowed.

The order in which items are returned by primary key (or the major portion of a primary key) for files that have indexed sequential file organization.

The collating sequence specified for a file also affects the comparison performed by the FORTRAN DML READ and START statements, and by the COBOL START statement.

The collating sequence of a Query Update catalog file must be the DISPLAY collating sequence.

The collating sequences in this appendix apply only to operations performed on data base files. If collation is used in any other portion of an application program, the collating sequence defined by the program is used.

TABLE I-1. DATA BASE COLLATING SEQUENCE

Collating Sequence		ASCII		COBOL		DISPLAY	
Decimal	Octal	Graphics	Display Code	Graphics	Display Code	Graphics	Display Code
00	00	blank	55	blank	55	:†	00†
01	01	!	66	<	74	A	01
02	02	=	64	%†	63†	B	02
03	03	#	60	[	61	C	03
04	04	$	53	→	65	D	04
05	05	%†	63†	≡	60	E	05
06	06	&	67	∧	67	F	06
07	07	'	70	↑	70	G	07
08	10	(	51	↓	71	H	10
09	11	)	52	>	73	I	11
10	12	*	47	≥	75	J	12
11	13	+	45	¬	76	K	13
12	14	,	56	.	57	L	14
13	15	-	46	)	52	M	15
14	16	.	57	;	77	N	16
15	17	/	50	+	45	O	17
16	20	0	33	$	53	P	20
17	21	1	34	*	47	Q	21
18	22	2	35	-	46	R	22
19	23	3	36	/	50	S	23
20	24	4	37	,	56	T	24
21	25	5	40	(	51	U	25
22	26	6	41	=	54	V	26
23	27	7	42	≠	64	W	27
24	30	8	43	<	72	X	30
25	31	9	44	A	01	Y	31
26	32	:†	00†	B	02	Z	32
27	33	;	77	C	03	0	33
28	34	<	72	D	04	1	34
29	35	=	54	E	05	2	35
30	36	>	73	F	06	3	36
31	37	?	71	G	07	4	37
32	40	@	74	H	10	5	40
33	41	A	01	I	11	6	41
34	42	B	02	∨	66	7	42
35	43	C	03	J	12	8	43
36	44	D	04	K	13	9	44
37	45	E	05	L	14	+	45
38	46	F	06	M	15	-	46
39	47	G	07	N	16	*	47
40	50	H	10	O	17	/	50
41	51	I	11	P	20	(	51
42	52	J	12	Q	21	)	52
43	53	K	13	R	22	$	53
44	54	L	14	]	62	=	54
45	55	M	15	S	23	blank	55
46	56	N	16	T	24	,	56
47	57	O	17	U	25	.	57
48	60	P	20	V	26	≡	60
49	61	Q	21	W	27	[	61
50	62	R	22	X	30	]	62
51	63	S	23	Y	31	%†	63†
52	64	T	24	Z	32	≠	64
53	65	U	25	:†	00†	→	65
54	66	V	26	0	33	∨	66
55	67	W	27	1	34	∧	67
56	70	X	30	2	35	↑	70
57	71	Y	31	3	36	↓	71
58	72	Z	32	4	37	<	72
59	73	[	61	5	40	>	73
60	74	\	75	6	41	≤	74
61	75	]	62	7	42	≥	75
62	76	^	76	8	43	¬	76
63	77	-	65	9	44	;	77

†In installations using a 63-graphic set, the % graphic does not exist. The : graphic is display code 63.

SUMMARY OF DATA DEFINITIONS IN DMS-170 J

This summary indicates the correspondence of the clauses or statements that can be used in defining data items in DMS-170. Table J-1 shows the schema definition required for data items of each schema data class and the subschema definitions that correspond to each schema data class.

For Query Update access to schema-defined data base files in CYBER Record Manager (CRM) data base access mode, the data base must be defined in the Query Update subschema exactly as the data base is defined in the schema. Therefore, every data item must be defined to correspond in size and class to the schema definition of the item.

For COBOL, FORTRAN, and Query Update access to schema-defined areas in CYBER Database Control System (CDCS) data base access mode, the definition of data items in the subschema does not have to correspond exactly to the schema definition of data items. Through mapping, CDCS can generate a record image conforming to the subschema format from a record in schema format, or can perform the conversion of data from subschema format to schema format. Detailed information about the conversions allowed is included in the CDCS 2 Data Administrator reference manual.

TABLE J-1. DATA DEFINITION IN DMS-170

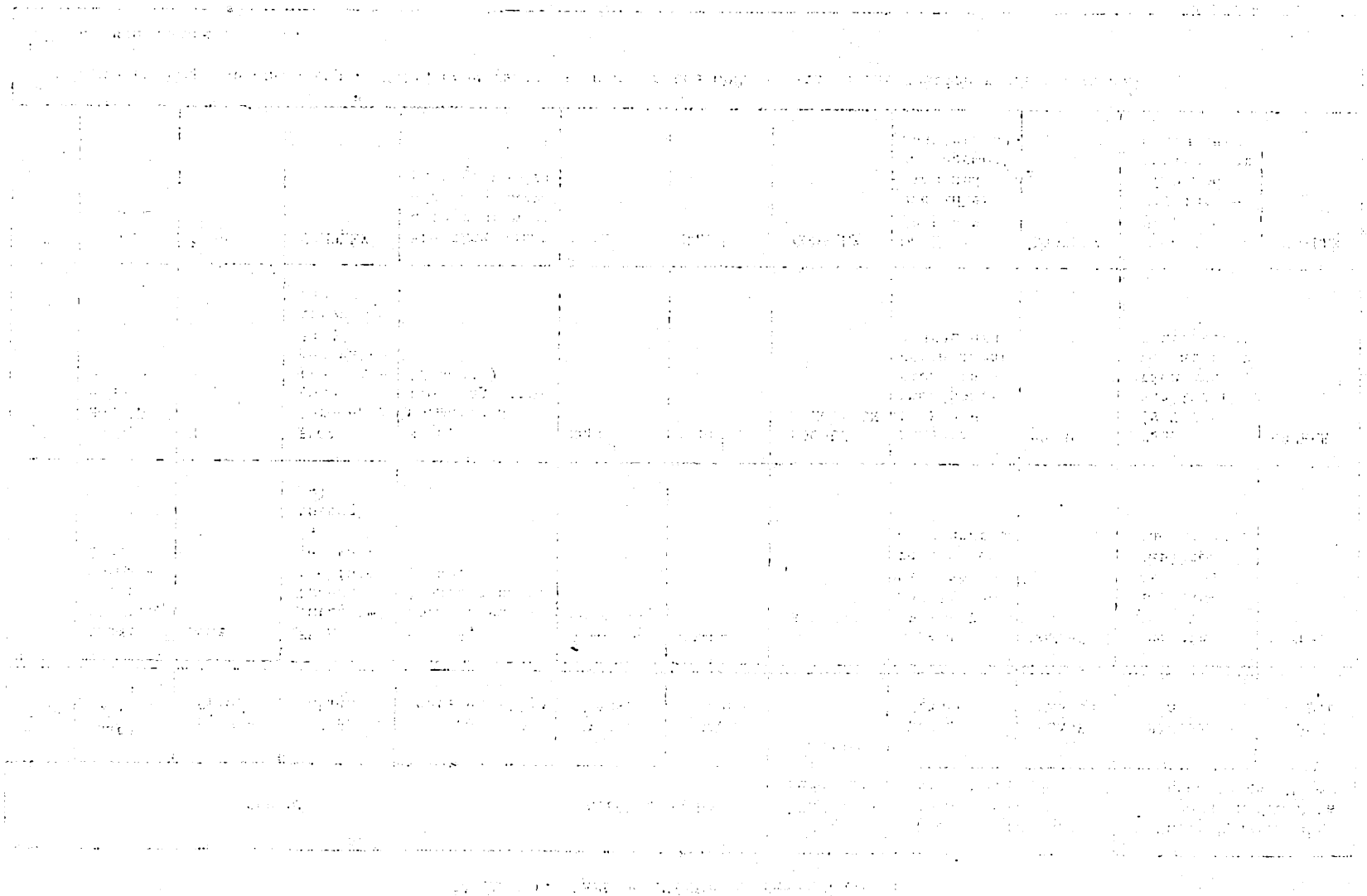
SCHEMA					COBOL Subschema		FORTRAN 5 Subschema Type Statement	Query Update Subschema in CDCS Data Base Access Mode		Query Update Subschema in CRM Data Base Access Mode	
Data Class No.	Data Class Name	PICTURE Clause	TYPE Clause	Internal Representation	PICTURE Clause	USAGE Clause		PICTURE Clause	USAGE Clause	PICTURE Clause	USAGE Clause
0	Display alpha-numeric	Alpha-numeric (A X 9; not all As or 9s; mixed specifi-cation used as all Xs)	CHARACTER	Display code, alphanumeric	Alpha-numeric (A X 9; not all As or 9s; mixed specifi-cation used as all Xs)	DISPLAY (or none)	CHARACTER	Alpha-numeric (A X 9; not all As or 9s; mixed specifi-cation used as all Xs)	DISPLAY (or none)	Alpha-numeric (A X 9; not all As or 9s; mixed specifi-cation used as all Xs)	DISPLAY (or none)
1	Display alpha-betic	Alpha-betic (A)	None	Display code, alphabetic	Alpha-betic (A)	DISPLAY (or none)	CHARACTER	Alpha-betic (A)	DISPLAY (or none)	Alpha-betic (A) (or none)	DISPLAY (or none)
3	Display integer	Numeric (9 T)	None	Display code numeric, can have sign overpunch in last character position	Numeric (9 S)	DISPLAY COMP (or none)	None†	Numeric (9 S and insertion and re-placement characters)	DISPLAY COMP (or none)	Numeric (9 S and insertion and re-placement characters)	DISPLAY COMP (or none)
4	Display fixed	Numeric (9 P T V .)	None	Display code numeric plus implicit or explicit deci-mal or scaling position	Numeric (9 S V P)	DISPLAY COMP (or none)	None†	Numeric (9 S V P and inser-tion and replacement characters)	DISPLAY COMP (or none)	Numeric (9 S V P and inser-tion and replacement characters)	DISPLAY COMP (or none)
10	Coded binary integer	None	FIXED integer-1 integer-2 (where the inte-ger value is 1 thru 18)	Binary integer	Numeric (9 S V P)	COMP-1 INDEX††	INTEGER LOGICAL BOOLEAN	Numeric (9 S V P and inser-tion and replacement characters)	COMP-1 INDEX†† LOGICAL	Numeric (9 S V P and inser-tion and replacement characters)	COMP-1 INTEGER LOGICAL

TABLE J-1. DATA DEFINITION IN DMS-170 (Contd)

SCHEMA					COBOL Subschema		FORTRAN 5 Subschema Type Statement	Query Update Subschema in CDCS Data Base Access Mode		Query Update Subschema in CRM Data Base Access Mode	
Data Class No.	Data Class Name	PICTURE Clause	TYPE Clause	Internal Representation	PICTURE Clause	USAGE Clause		PICTURE Clause	USAGE Clause	PICTURE Clause	USAGE Clause
13	Coded floating point normalized	None	FLOAT integer-1 (where the integer value is 1 through 14)	Signed, normalized floating point (1 word)	Numeric (9 S V P)	COMP-2	REAL BOOLEAN	Numeric (9 S V P and insertion and replacement characters)	COMP-2	Numeric (9 S V P and insertion and replacement characters)	COMP-2
14	Coded double precision	None	FLOAT integer-1 (where the integer value is 15 through 29)	Signed, normalized floating point (2 words)	None†	None†	DOUBLE PRECISION	Numeric (9 S V P and insertion and replacement characters)	DOUBLE	Numeric (9 S V P and insertion and replacement characters)	DOUBLE
15	Coded complex	None	COMPLEX	Floating point with real part and imaginary part (2 words)	None†	None†	COMPLEX	Numeric (9 S V P and insertion and replacement characters)	COMPLEX	Numeric (9 S V P and insertion and replacement characters)	COMPLEX

†No corresponding subschema type. Valid conversion is shown in the CDCS 2 Data Administration reference manual.

††No picture clause allowed.



INDEX

COMMENT SHEET

MANUAL TITLE: CYBER Database Control System Version 2
Application Programming Reference Manual

PUBLICATION NO.: 60485300

REVISION: D

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments on the back (please include page number references).

_____ Please reply _____ No reply necessary

FOLD FOLD

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
P.O. BOX 3492
Sunnyvale, California 94088-3492

CUT ALONG LINE

FOLD FOLD

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND TAPE

NAME:

COMPANY:

STREET ADDRESS:

CITY/STATE/ZIP:

TAPE TAPE