

CA-IDMS®

Navigational DML Programming

15.0



Computer Associates™

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this documentation for their own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

Second Edition, October 2001

© 2001 Computer Associates International, Inc.
All rights reserved.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

How to Use This Manual	ix
Chapter 1. Overview of the CA-IDMS Programming Environment	1-1
1.1 About this chapter	1-3
1.2 Terminology	1-4
1.3 Database access	1-5
1.4 CA-IDMS batch and online environments	1-6
1.5 Navigational DML programs and CA-ADS programs	1-7
Chapter 2. Basic DML Programming Concepts	2-1
2.1 About this chapter	2-3
2.2 Database components	2-4
2.2.1 Schemas	2-4
2.2.2 Subschemas	2-8
2.3 Db-keys and page information	2-11
2.4 Run units, locks, and recovery units	2-13
2.4.1 Run units	2-13
2.4.2 Record locks	2-13
2.4.3 Area locks	2-17
2.4.4 Area usage modes	2-18
2.4.5 Recovery units	2-20
2.5 Basic programming considerations	2-24
2.5.1 Establishing communications with CA-IDMS	2-24
2.5.2 Checking the status of statement execution	2-29
2.5.3 Specifying a dictnode or dictname for a run unit	2-29
2.5.4 Using currency	2-30
2.5.5 Collecting database statistics	2-30
2.6 IDMS communications block	2-32
Chapter 3. Introduction to Database Access with Navigational DML	3-1
3.1 About this chapter	3-3
3.2 Records	3-4
3.2.1 Record name	3-4
3.2.2 Record identification	3-5
3.2.3 Storage mode	3-5
3.2.4 Record length	3-5
3.2.5 Location mode	3-5
3.2.6 Duplicates option	3-6
3.2.7 Area name	3-7
3.3 Sets	3-8
3.3.1 Set name	3-9
3.3.2 Set linkage	3-9
3.3.3 Set membership options	3-10
3.3.4 Set order	3-11
3.3.5 Chained and indexed sets	3-13
3.3.6 Set relationship definition	3-13
3.4 Data structure diagram	3-20

3.5	Currency	3-22
3.5.1	Use and updating of currency by DML verbs	3-23
3.5.2	Updating currencies during DML processing	3-25
3.6	Database access execution sequence	3-27
Chapter 4. Navigational DML Programming Techniques		4-1
4.1	About this chapter	4-3
4.2	Retrieving records	4-4
4.2.1	Accessing CALC records	4-4
4.2.2	Walking a set	4-5
4.2.3	Accessing a sorted set	4-6
4.2.4	Performing an area sweep	4-10
4.2.5	Accessing owner records	4-12
4.2.6	Reestablishing run-unit currency	4-14
4.2.7	Accessing a record by its db-key	4-15
4.2.8	Accessing indexed records	4-18
4.2.9	Moving contents of a record occurrence	4-21
4.3	Saving db-key, page information and bind addresses	4-24
4.3.1	Saving a db-key	4-24
4.3.2	Saving page information	4-28
4.3.3	Saving a record's BIND address	4-30
4.4	Checking for set membership	4-31
4.4.1	Using the IF EMPTY statement	4-31
4.4.2	Using the IF MEMBER statement	4-32
4.5	Updating the database	4-35
4.5.1	Storing records	4-35
4.5.2	Modifying records	4-37
4.5.3	Erasing records	4-39
4.5.4	Connecting records to a set	4-44
4.5.5	Disconnecting records from a set	4-45
4.6	Locking records	4-48
Chapter 5. Advanced DML Programming Topics		5-1
5.1	About this chapter	5-3
5.2	Copying record definitions and their synonyms	5-4
5.3	Accessing bill-of-materials structures	5-6
5.3.1	Storing a bill-of-materials structure	5-6
5.3.2	Retrieving a bill-of-materials structure	5-8
Chapter 6. Introduction to Online Programming		6-1
6.1	About this chapter	6-3
6.2	DC as an operating system	6-4
6.3	Transaction and task processing	6-5
6.4	Pseudoconversational programming	6-6
6.5	Performance considerations	6-9
6.6	Error handling	6-10
6.7	Using the IDMS communications block	6-11
Chapter 7. Terminal Management		7-1
7.1	About this chapter	7-3
7.2	Mapping mode	7-4

7.2.1	Housekeeping	7-5
7.2.2	Displaying screen output	7-5
7.2.3	Reading screen input	7-8
7.2.4	Modifying map options	7-11
7.2.5	Writing and reading in one step	7-12
7.2.6	Suppressing map error messages	7-13
7.2.7	Testing for identical data	7-14
7.3	Using pageable maps	7-15
7.3.1	Pageable map format	7-15
7.3.2	Conducting a map paging session	7-17
7.3.3	How to code a browse application	7-20
7.3.4	How to code an update application	7-23
7.3.5	Overriding automatic mapout for pageable maps	7-27
7.4	Line mode	7-30
7.4.1	Beginning a line mode session	7-30
7.4.2	Writing a line of data	7-30
7.4.3	Reading a line of data	7-32
7.4.4	Ending a line mode session	7-33
7.4.5	3270-type considerations	7-33
Chapter 8. Storage, Scratch, and Queue Management		8-1
8.1	About this chapter	8-3
8.2	Using storage pools	8-4
8.2.1	User storage	8-5
8.2.2	User kept storage	8-6
8.2.3	Shared storage	8-9
8.2.4	Shared kept storage	8-10
8.2.5	Storage pool summary	8-11
8.3	Using scratch records	8-15
8.4	Using queue records	8-22
8.5	Using the terminal screen to transmit data	8-27
Chapter 9. DC Programming Techniques		9-1
9.1	About this chapter	9-3
9.2	Passing program control	9-4
9.2.1	Returning to a higher level program	9-5
9.2.2	Passing control laterally	9-6
9.2.3	Passing control, expecting to return	9-7
9.3	Retrieving task-related information	9-9
9.4	Maintaining data integrity in the online environment	9-11
9.4.1	Setting longterm explicit locks	9-11
9.4.2	Monitoring concurrent database access	9-14
9.5	Managing tables	9-18
9.6	Retrieving the current time and date	9-21
9.7	Writing to the journal file	9-23
9.8	Collecting DC statistics	9-25
9.9	Sending messages	9-28
9.9.1	Sending a message to the current user	9-28
9.9.2	Sending a message to other users	9-29
9.10	Writing to a printer	9-31

9.11	Writing JCL to a JES2 internal reader	9-33
9.12	Modifying a task's priority	9-34
9.13	Initiating nonterminal tasks	9-35
9.13.1	Attaching a task	9-35
9.13.2	Time-delayed tasks	9-36
9.13.3	External requests	9-36
9.13.4	Queue threshold tasks	9-36
9.14	Controlling abend processing	9-37
9.14.1	Terminating a task	9-37
9.14.2	Handling db-key deadlocks	9-37
9.14.3	Performing abend routines	9-39
9.15	Establishing and posting events	9-41
 Chapter 10. Advanced CA-IDMS Programming Topics		10-1
10.1	About this chapter	10-3
10.2	Calling a DC program from a CA-ADS dialog	10-4
10.3	Basic mode	10-6
10.3.1	Reading data from the terminal	10-7
10.3.2	Writing data to the terminal	10-7
10.4	Determining if asynchronous I/O is complete	10-8
10.5	Communicating with database procedures	10-9
10.5.1	BIND PROCEDURE	10-9
10.5.2	ACCEPT PROCEDURE CONTROL LOCATION	10-10
10.6	Managing queued resources	10-12
 Chapter 11. Testing		11-1
11.1	About this chapter	11-3
11.2	Preparing programs for execution	11-4
11.3	Selecting local mode or central version	11-5
11.4	Using SYSIDMS parameters and DCUF SET statements	11-6
11.5	Overriding subschemas (Release 10.2)	11-7
11.5.1	Overriding a batch program's subschema	11-7
11.5.2	Overriding an online program's subschema	11-9
11.6	Setting up an online test application	11-10
 Chapter 12. Debugging		12-1
12.1	About this chapter	12-3
12.2	Debugging batch programs with the CA-IDMS trace facility	12-4
12.3	Using the CA-OLQ menu facility	12-6
12.4	Reading task dumps	12-7
12.4.1	Contents of a snap dump	12-7
12.4.2	How to use the dump	12-9
12.5	Error checking	12-12
 Appendix A. PL/I Considerations		A-1
A.1	About this appendix	A-3
A.2	Transferring control	A-4
A.3	Using the Online Debugger with PL/I	A-5
A.3.1	Computation Phase	A-5
A.3.2	Sample Online Debugger Session	A-6

Appendix B. Assembler Considerations	B-1
B.1 About this appendix	B-3
Appendix C. Batch Access to DC Queues and Printers	C-1
C.1 About this appendix	C-3
Appendix D. XA Considerations	D-1
D.1 About this appendix	D-3
Appendix E. Running a Program Under TCF	E-1
E.1 About this appendix	E-3
E.2 Overview of TCF	E-4
E.3 Defining a TCF task to the DC system	E-6
E.4 Using the UCE for communication under TCF	E-7
E.5 Determining if TCF is active	E-9
E.6 Starting a new session	E-10
E.7 Resuming a suspended session	E-11
E.8 Processing a pseudoconverse	E-12
E.8.1 Suspend processing	E-12
E.8.2 End processing	E-12
E.8.3 Switch processing	E-12
E.9 Displaying error messages	E-14
E.10 Sample application under TCF	E-15
Appendix F. Calls to IDMSIN01	F-1
F.1 About IDMSIN01	F-3
Appendix G. 10.2 Services Batch Interface	G-1
G.1 About the 10.2 services batch interface	G-3
Index	X-1

How to Use This Manual

What this manual is about

This manual discusses the following topics:

- Programming navigational access to a CA-IDMS database
- Programming CA-IDMS applications in COBOL, PL/I, Assembler, Fortran, and RPG II
- Testing and debugging
- Topics of interest to advanced programmers

Who should use this manual

This manual is a guide for the developer of batch applications that access a non-SQL defined CA-IDMS database. It is also for the developer of applications that execute in a DC system and may or may not access a CA-IDMS database.

How information is presented

- **Programming functions** are explained by task.
- **Step-by-step instructions** guide you through the programming operations for each function.
- **Special considerations** for using each function are provided where appropriate.
- **Programming examples** are presented in context with other DML and host language-specific source statements.

All programming examples in this manual are given in COBOL. For specific information regarding PL/I or Assembler, refer to Appendix A, “PL/I Considerations” on page A-1 or Appendix B, “Assembler Considerations” on page B-1.

How product names are referenced

This manual uses the term CA-IDMS to refer to any one of the following CA-IDMS components:

- CA-IDMS/DB — The database management system
- CA-IDMS/DC — The data communications system and proprietary teleprocessing monitor
- CA-IDMS/UCF — The universal communications facility for accessing CA-IDMS database and data communications services through another teleprocessing monitor, such as CICS
- CA-IDMS/DDS — The distributed database system

This manual uses the terms DB, DC, UCF, and DDS to identify the specific CA-IDMS component only when it is important to your understanding of the product. References to DC apply equally to UCF unless otherwise noted.

Related documentation

Language-specific DML reference manuals that include the syntax and syntax rules for each DML statement:

CA-IDMS DML Reference - COBOL

CA-IDMS DML Reference - PL/I

CA-IDMS DML Reference - Assembler

For information that will help you plan, code, and test your application programs:

CA-IDMS Messages and Codes

CA-IDMS Transfer Control Facility

CA-IDMS Utilities

CA-IDMS Mapping Facility

CA-IDMS Online Debugger

For related information on more advanced topics:

CA-IDMS System Generation

CA-IDMS System Operations

CA-IDMS Database Administration

Chapter 1. Overview of the CA-IDMS Programming Environment

- 1.1 About this chapter 1-3
- 1.2 Terminology 1-4
- 1.3 Database access 1-5
- 1.4 CA-IDMS batch and online environments 1-6
- 1.5 Navigational DML programs and CA-ADS programs 1-7

1.1 About this chapter

This chapter provides a high-level description of the environments that CA-IDMS provides for creating and executing application programs. The chapter begins with definitions of key terms used when discussing navigational DML programming.

1.2 Terminology

As you use this manual, you should be familiar with the these terms:

- **Database management system (DBMS)** — The software component of CA-IDMS that accesses the database, handles all database input/output (I/O) and space management functions, and maintains all data and data relationships.
- **Database administrator (DBA)** — The individual or staff responsible for implementing and maintaining the database.
- **CA-IDMS central version** — A CA-IDMS mode of operation that allows multiple application programs to execute concurrently, sharing a single DBMS. Additionally, the central version provides for automatic recovery in the event of a program failure.
- **Host language** — COBOL, PL/I, Assembler.
- **Navigational DML programming** — Programming with CA-IDMS navigational DML statements.

You code DML statements in the program source as if they were a part of the host language (such as COBOL, PL/I, or Assembler) The precompiler converts each DML statement into a subroutine call that requests DB and DC services.

- **Run unit** — That portion of a CA-IDMS program that establishes communication with the DBMS, initiates database requests, and releases database resources.
- **Variable storage** — That portion of storage associated with an application program at runtime; for example, the WORKING-STORAGE and LINKAGE SECTIONS of a COBOL program.

1.3 Database access

Access methods: CA-IDMS provides these ways for a program to access a database:

- Navigational DML statements — Access a non-SQL-defined database and include information about how data is stored
- SQL statements — Access data associated with an SQL schema
- LRF statements — Access data by referencing defined logical record paths

This manual discusses navigational programming techniques.

►► For more information about using SQL in a program to access the database, refer to *CA-IDMS SQL Programming Guide*.

►► For more information about using LRF in a program to access the database, refer to *CA-IDMS Logical Record Facility*.

Navigational programming: Navigational programs access database records and sets one record at a time, checking and maintaining currency in order to assure correct results. To use navigational DML statements, you must have a thorough knowledge of the database structure.

Navigational programming provides:

- **Control over error checking** — You can check the result of each navigational statement as you go, enabling more thorough error detection.
- **Flexibility in choosing database access strategy** — You can enter the database sequentially or by using a symbolic key value, a calculated key value, or a database key (db-key) value.

1.4 CA-IDMS batch and online environments

You design CA-IDMS programs to run in either the batch or the online environment.

Batch program: A batch program typically processes large volumes of sequential input transactions and writes output in the form of files and reports. Errors are detected and held in a suspense file to be corrected and resubmitted with the next batch run. Because batch jobs perform such extensive processing, they are usually run at off-peak hours.

Online program: An online program typically processes transaction requests from terminals connected directly to the computer and displays transaction results at the terminal. Errors are detected immediately; the user is required to correct them before the transaction can be processed.

An online system must efficiently handle multiple requests from multiple sources and be able to manage a variety of concurrent transaction requests. Additionally, online processing is immediate; thus, fast response time is essential in maintaining an efficient work environment for multiple users.

Online programs run under DC, the teleprocessing monitor that is fully integrated with DB and with IDD, the CA-IDMS dictionary.

►► For a list of the teleprocessing monitors supported, refer to the language-specific CA-IDMS DML reference manual.

1.5 Navigational DML programs and CA-ADS programs

CA-ADS: CA-ADS is an application development system that includes a fourth-generation programming language.

►► For more information about CA-ADS, refer to *CA-ADS User Guide*.

Both CA-ADS application programs and navigational DML programs can execute in the CA-IDMS environment. A navigational DML program can be called by a CA-ADS dialog.

►► For more information about calling a navigational DML program from a CA-ADS dialog, see 10.2, “Calling a DC program from a CA-ADS dialog” on page 10-4.

Chapter 2. Basic DML Programming Concepts

2.1 About this chapter	2-3
2.2 Database components	2-4
2.2.1 Schemas	2-4
2.2.2 Subschemas	2-8
2.3 Db-keys and page information	2-11
2.4 Run units, locks, and recovery units	2-13
2.4.1 Run units	2-13
2.4.2 Record locks	2-13
2.4.3 Area locks	2-17
2.4.4 Area usage modes	2-18
2.4.5 Recovery units	2-20
2.5 Basic programming considerations	2-24
2.5.1 Establishing communications with CA-IDMS	2-24
2.5.2 Checking the status of statement execution	2-29
2.5.3 Specifying a dictnode or dictname for a run unit	2-29
2.5.4 Using currency	2-30
2.5.5 Collecting database statistics	2-30
2.6 IDMS communications block	2-32

2.1 About this chapter

This chapter highlights programming considerations common for all navigational DML programmers, regardless of application type (batch or online).

If you are writing database applications, you should be familiar with all the information in this chapter. Programmers whose applications perform no database access need read only the 2.5, “Basic programming considerations” on page 2-24 section of this chapter.

This chapter presents:

- **Database components** — A discussion of schemas and subschemas, including information on areas, records, and the IDMSRPTS utility
- **Run unit, lock, and recovery unit considerations** — A discussion of run units, record and area locks including information on area usage modes, and recovery units
- **Basic programming considerations** — A discussion of DML housekeeping functions, communications blocks, currency, and database statistics

2.2 Database components

The DBA uses CA-IDMS data description language (DDL) to define the three components of a database:

- Schema
- DMCL
- Subschema

Schema: A schema provides the complete logical description of the content and structure of a database, including the names and descriptions of all areas, records, and sets.

DMCL: The DMCL:

- Controls the mapping of the schema-defined database into physical files; specifies the size of the buffers
- Designates which areas of the database are utilized at runtime
- Optionally, describes the files used to journal database activities

►► For more information on the DMCL, refer to the *CA-IDMS Database Administration*.

Subschema: A subschema defines the program's view of the database. It typically defines a subset of the records and record elements contained in the schema.

The subschema also defines restrictions placed on the DML statements that can be used to access that view. Additionally, the subschema can contain DBA-defined database-access paths used in LRF.

►► For information about LRF paths and programming, refer to *CA-IDMS Logical Record Facility*.

2.2.1 Schemas

A schema is the description of the database; one schema is defined for each database. The entities that the DBA defines in the schema are:

- **Areas** are portions of physical storage that map to files on a one-to-one, many-to-one, or one-to-many basis.
- **Records** define categories of information and are contained in areas.
- **Sets** either establish logical relationships between records or place an index on a record.

►► For more information about sets, see Chapter 4, “Navigational DML Programming Techniques” on page 4-1.

Areas: An area is a subdivision of database storage that maps to a direct-access file. The subdivision of a database into areas provides the following advantages:

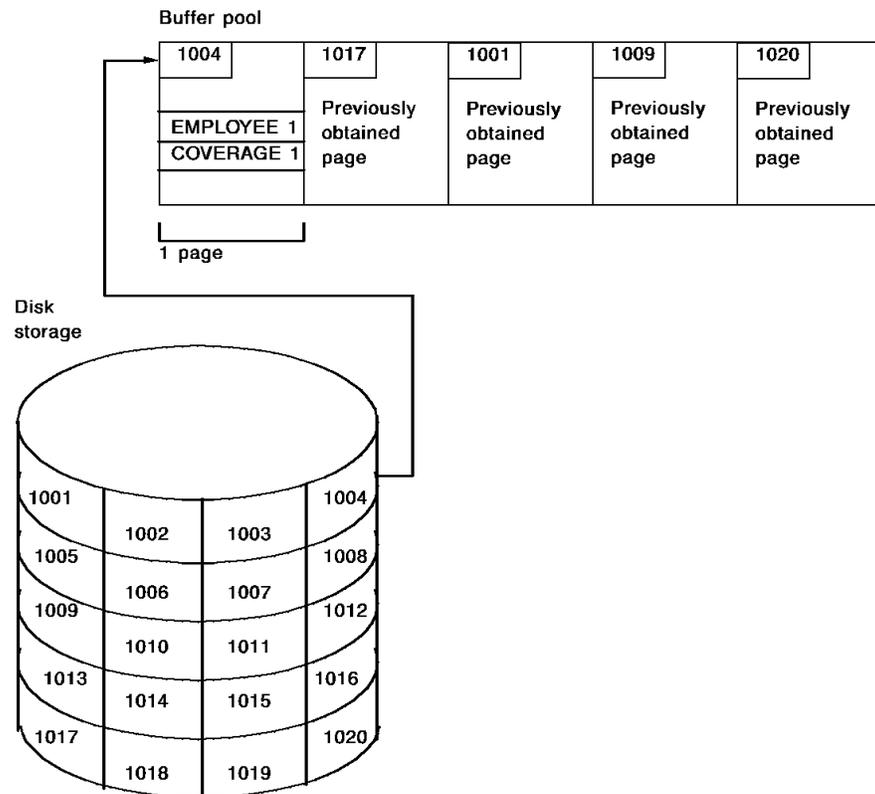
- Since each area is a discrete entity, you need prepare only the areas relevant for use by your application program. This allows concurrently executing programs access to areas that you are not using.
- Your program can restrict the use of an area, thereby preventing concurrently executing programs from accessing or updating records in that area. This allows you to determine the level of control that your program will have over an area (for example, if an application performs extensive updates, you might want more control than in an application that simply retrieves records).
- Assignment of a single record type to a single area enhances runtime efficiency when using either serial sweeps of the database or system-owned indexes.
- The database can be initialized, restructured, backed up, and recovered on an area-by-area basis.

Pages: Each area consists of a given number of contiguous pages within the database, as defined by the DBA. A page is a division of an area that corresponds to a physical block of storage on a direct-access storage device.

Each page in a database area contains record occurrences and database control information (including a unique numeric page identifier). Page limits and page sizes are assigned for each area by the DBA.

Database paging: Since each database page corresponds to a direct-access block on a file, data is transferred from the storage device to the buffer one page at a time. Each transfer results in one physical I/O.

The figure below shows database paging:



Minimizing paging I/O: Since database paging involves I/O overhead, you should minimize database paging in your program. Ways to cut down on database paging include:

- Saving the db-keys of retrieved records to be used later in the program
- Using the RETURN statement to establish currency in an indexed set
- FINDing rather than OBTAINing records that are only used for database positioning
- Using the ACCEPT statement to retrieve the db-key of the next, prior, or owner records

►► For more information on ways to reduce database paging, see Chapter 4, “Navigational DML Programming Techniques” on page 4-1.

Records: Database records are defined by the DBA in the dictionary (or optionally in the schema). Records are composed of elements (also called data items). Individual occurrences of each record type are maintained in the database according to their database keys.

Record type: The description of a record *type* consists of its name, followed by the names and attributes of all elements contained in the record. The example below shows the information associated with the record type named DEPARTMENT. The elements contained in the DEPARTMENT record include a 4-character number (DEPT-ID-0410), a 45-character name (DEPT-NAME-0410), and another 4-character number (DEPT-HEAD-ID-0410). This description is the model for the DEPARTMENT record type wherever it appears in the database.

ELEMENT NAME	USAGE	PICTURE
DEPT-ID-0410	DISPLAY	9(4)
DEPT-NAME-0410	DISPLAY	X(45)
DEPT-HEAD-ID-0410	DISPLAY	9(4)

Record occurrences: A record *occurrence* consists of the actual data stored in the database. Three occurrences of the DEPARTMENT record type might appear as follows:

DEPT-ID-0410	DEPT-NAME-0410	DEPT-HEAD-ID-0410
1000	Personnel	0013
2000	Accounting and Payroll	0011
3100	Brainstorming	0015

Any number of record types can be assigned to a database area, but the number of record occurrences within an area is determined by the total amount of physical storage space available for the area.

Symbolic keys: Symbolic keys, which include CALC, sort, and index keys, are user-supplied record-element values that determine where a record is stored. You can use these keys to identify a particular record to help reduce I/O, thereby increasing overall efficiency.

Symbolic keys can consist of multiple record elements (up to 256 bytes), as shown in this example:

```
01 RECORD-A.
   03 SYMBOLIC-KEY-PART-1 PIC X(3).
   03 SYMBOLIC-KEY-PART-2 PIC X(2).
   03 SYMBOLIC-KEY-PART-3 PIC X(4).
   03 NOT-A-KEY-1         PIC X(30).
   03 NOT-A-KEY-2         PIC X(20).
   03 NOT-A-KEY-3         PIC 9(5).
```

These elements need not be adjacent to one another in the record definition, as in this example:

```
01 RECORD-B.
   03 SYMBOLIC-KEY-PART-1 PIC 9(2).
   03 NOT-A-KEY-1         PIC X(8).
   03 SYMBOLIC-KEY-PART-2 PIC 9(5).
   03 NOT-A-KEY-2         PIC XXX.
   03 SYMBOLIC-KEY-PART-3 PIC X(15).
   03 NOT-A-KEY-3         PIC 9(5)V99.
```

Note: The entire CALC key must be defined in the record in order to access that record as CALC.

2.2.2 Subschemas

A **subschema** is a program's view of the database; it typically defines a subset of the records and record elements contained in the schema. The following rules apply to subschema usage:

- Any number of subschemas can be associated with a single schema.
- Any number of programs can share a subschema.
- A program can have only one subschema.

Comparing subschema and schema: The table below compares the features and characteristics of subschemas and schemas.

Subschema	Schema
One or more per database	One per database
A program's view of the database (subset of records and record elements)	Complete database description (all records and record elements)
Source description resides in the DDL/DML area of the dictionary	Source description resides in the DDL/DML area of the dictionary
Source description used at DML program compile time	Source description not used at DML program compile time
A load module resides in the DDL/DCL/DL area of the dictionary or in a load (core-image) library	No load module
Load module used at runtime	Not used at runtime

Effect of subschema definitions: The subschema defines restrictions placed on the DML statements that can be used to access database records. For example, you may be able to retrieve a record but not modify or erase it. Additionally, the DBA can specify in the subschema that each program must be registered in the dictionary before compilation under one of the precompilers.

Subschema access restrictions: DBA-designated access restrictions, which are defined in the subschema, control program access to the database. Restrictions can be placed on:

- **Areas** — Access restrictions placed on areas prevent programs from readying them in specified usage modes (see 2.4.4, “Area usage modes” on page 2-18 later in this chapter).

For example, a subschema with an update access restriction on the ORG-DEMO-REGION area prevents programs from readying that area in any update mode and enforces read-only access to the area.

- **Records** — Access restrictions placed on records prohibit programs from performing one or more of the following DML functions against the specified record types:

STORE
ERASE
CONNECT
FIND
MODIFY
GET
DISCONNECT
KEEP

For example, ERASE IS NOT ALLOWED for the OFFICE record type prohibits your program from erasing OFFICE record occurrences.

Note: The DML OBTAIN statement is a combination of FIND and GET; access restrictions on either FIND or GET will affect the use of OBTAIN.

- **Sets** — Access restrictions placed on sets prohibit programs from performing one or more of the following DML functions against record occurrences in the specified set:

CONNECT
FIND
DISCONNECT
KEEP

For example, DISCONNECT IS NOT ALLOWED for the JOB-EMPOSITION set prohibits your program from disconnecting EMPOSITION occurrences from the JOB-EMPOSITION set.

If your program issues a DML statement that is prohibited in the subschema, the DBMS returns a status of *nn10* to the ERROR-STATUS field in the IDMS communications block. The IDMSRPTS utility (discussed below) produces listings of any access restrictions that apply to a given subschema.

Program registration: The DBA can specify in the subschema that each program must be defined in the dictionary before compilation under one of the precompilers. If program registration is in effect, you should ensure that the name listed in the PROGRAM-ID statement matches the program name assigned through IDD.

IDMSRPTS utility: The IDMSRPTS utility produces listings that describe the database definition (that is, the schema and all associated subschemas). These reports are useful in all phases of program development; they provide the following information:

- Names of all records, sets, and areas included in the subschema
- Names, attributes, and positions of all elements included in each subschema record
- Storage mode of each record
- Access restrictions

- Set characteristics

IDMSRPTS parameters: The table below lists the parameters of the IDMSRPTS utility that are most useful to applications programmers.

Parameter	Requested information
RECDES	All records and record elements defined in the schema
SETDES	Set name, owner, membership options, and linkage options for all sets defined in the schema
SUBREC	All records and record elements defined in the subschema; access restrictions placed on records
SUBSET	Set name, owner, membership options, and linkage options for all sets defined in the subschema; access restrictions placed on sets
SUBAREA	Usage modes applicable to subschema areas, default usage modes; access restrictions placed on areas

►► For more information on the IDMSRPTS utility, refer to *CA-IDMS Utilities*.

IDMSRPTS utility sample OS/390 JCL: A sample IDMSRPTS JCL stream follows (the SUBREC report lists record information for the specified subschema):

```
//          EXEC PGM=IDMSRPTS,REGION=2048K
//STEPLIB DD DSN=idms.cailib,DISP=SHR
//SYSCTL DD DSN=idms.sysctl,DISP=SHR
//SYSJRNL DD DUMMY
//SYSOUT DD SYSOUT=A
//SYSLST DD SYSOUT=A
//SYSIDMS DD *
DBNAME=dictname
//SYSIPT DD *
SCHEMA=schema-name VERSION=nnnn
SUBSCHEMA=subschemaname
SUBREC
/*
```

2.3 Db-keys and page information

Database keys: Each database record occurrence is identified by a database key (db-key). The db-key is a 4-byte identifier that consists of:

- A page number, which identifies the page on which the record occurrence is stored
- A line number, which identifies the record's location on the page

The DBMS assigns a db-key to a record occurrence when the occurrence is stored in the database; that db-key remains unchanged until the occurrence is erased or the database is unloaded and subsequently reloaded.

Page information: Page numbers are used to identify pages within a database; however, a page number is not necessarily unique across all areas accessible to a CA-IDMS runtime system. If a page number is not unique, you can qualify it with additional information so that it uniquely identifies a page. The additional qualifying information is a 4-byte identifier that consists of:

- A 2-byte page group
- A 2-byte db-key radix

A page group is a number assigned to an area by the DBA for the purpose of making the area's page range unique to the CA-IDMS runtime system. The db-key radix indicates the number of bits within the 4-byte db-key that contain a record's line number. The db-key radix is calculated by CA-IDMS based on the maximum number of record occurrences that can be stored on a page of the area.

►► For more information on page groups and maximum records per page, refer to *CA-IDMS Database Administration*.

Qualifying db-keys: Normally all areas accessed by a run unit have the same page information and so the db-key of a record occurrence uniquely identifies it from all other record occurrences accessible to the run unit. A run unit, however, can access areas with different page groups or db-key radices if it accesses a database defined to allow mixed page group binds. When this happens, a db-key must be qualified either by record type or page information so that it uniquely identifies a record occurrence.

In order to permit qualification, either record type or page information can be specified when retrieving a record occurrence through its db-key. Whenever a record occurrence is retrieved, its record type, db-key and associated page information are returned to the application program. You can save these for later use in retrieval commands. It is also possible to determine the page information associated with a specific record type by issuing an ACCEPT Page-Info command.

►► For more information on retrieving a record by db-key, refer to 4.2.7, "Accessing a record by its db-key" on page 4-15.

►► For more information on saving qualifying information, refer to 4.3.2, “Saving page information” on page 4-28.

►► For more information on mixed page group binds and accessing areas with different page groups or maximum records per page, refer to *CA-IDMS Database Administration*.

Page information and record types: For the duration of a run unit, the page information for all occurrences of a given record type is the same. Similarly, the page information for all record types within an area or all record types associated with a set is the same.

Using page information to interpret db-keys: The format of a db-key value depends on its db-key radix. The db-key radix specifies the number of bits within a db-key that are reserved for a record occurrence's line number. Since the db-key radix is part of the page information associated with a db-key, you can use page information to interpret a 4-byte db-key value. You can use this when displaying db-keys for error reporting purposes or when establishing a target page for storing records whose location mode is direct.

Given a db-key, you can separate its associated page number by dividing the db-key by 2 raised to the power of the db-key radix. For example, if the db-key is 4, you divide the db-key value by 2^{*4} . The resulting value is the page number of the db-key. To separate the line number, you multiply the page number by 2 raised to the power of the db-key radix and subtract this value from the db-key value. The result is the line number of the db-key. You can use the following two formulas to calculate the page and line numbers from a db-key value:

- Page-number = db-key value / ($2^{*db\text{-key radix}}$)
- Line-number = db-key value - (page number*($2^{*db\text{-key radix}}$))

2.4 Run units, locks, and recovery units

A run unit is that portion of CA-IDMS processing during which communications are established with the database. Navigational programs that maintain efficient run units help to maximize the resources of a runtime system. Well-managed record locks, area locks, and recovery units are major considerations in maintaining efficient run units.

Record locks and area in-use locks ensure data integrity by preventing concurrent update of database records. Additionally, your program can specify area usage modes to ensure a particular level of security in database areas to be accessed. You should be familiar with these locks and usage modes, their uses, and their effect on the runtime system, particularly when running under the CA-IDMS central version.

2.4.1 Run units

A run unit begins with the BIND RUN-UNIT statement and (if successful) ends with the FINISH statement. The BIND RUN-UNIT and FINISH statements are analogous to the processing time between OPEN and CLOSE file statements. A program can consist of any number of run units that are executed serially, but typically contains only one.

Note: If your program consists of more than one run unit, you must reinitialize the ERROR-STATUS field in the IDMS communications block to the value 1400 before reissuing the BIND RUN-UNIT and READY statements.

2.4.2 Record locks

In general, record locks prevent concurrent retrieval and update by separate run units operating under the same central version. This statement does not apply when:

- Run units operate in local mode — concurrent update of record occurrences is prevented by *area* locks
- RETRIEVAL NOLOCK has been specified in system generation — the system does not maintain locks for retrieval run units

Exclusive lock: An exclusive lock indicates that no other run unit can access the designated record occurrence in any way. Only one run unit at a time can place an exclusive lock on a record occurrence. A run unit can place an exclusive lock on a record occurrence only if that occurrence has not been assigned any locks (shared or exclusive) by another run unit. A run unit that tries to place an exclusive lock on an occurrence that already has been locked must wait until all other locks on the occurrence are released.

Shared lock: A shared lock indicates that other run units can retrieve the designated record occurrence but cannot update it. Any number of run units can place a shared lock on a record occurrence. A run unit that tries to place a shared lock on an occurrence that has already received an exclusive lock must wait until the exclusive lock is released.

Notify lock: A notify lock is used in the online environment to monitor database access to a specified record occurrence.

►► For more information on notify locks, see 9.4, “Maintaining data integrity in the online environment” on page 9-11.

Implicit and explicit locks: Record locks can be set implicitly by the central version or you can set them explicitly by coding the DML KEEP function in the program.

Implicit locks: Implicit locks are maintained automatically by the central version for every run unit accessing the database in shared update usage mode. The DBA can also specify that implicit locks be maintained for run units accessing the database in shared retrieval or protected update usage mode.

►► For further details about usage modes, see 2.4.4, “Area usage modes” on page 2-18 later in this chapter.

Types of implicit lock: Implicit locks can be shared or exclusive, as follows:

- The central version places **implicit shared locks** on the record occurrences that are current of run unit, record, set, and area. These locks remain in effect until the record occurrences are no longer current, thereby preventing concurrently executing run units from updating the same record.
- The central version places an **implicit exclusive lock** on every record occurrence that is modified by a DML statement (STORE, MODIFY, or ERASE). Additionally, the central version sets implicit exclusive locks for:
 - The next and prior record occurrences for all sets in which the modified record participates
 - Each database page on which the amount of space has been altered as the result of a STORE, MODIFY, or ERASE statement

The central version maintains implicit exclusive locks for the duration of the recovery unit to prevent concurrently executing run units that are maintaining locks from accessing modified records that might have to be rolled back because of an error later in the program.

►► For more information about recovery units and implicit exclusive locks, see 2.4.5, “Recovery units” on page 2-20 later in this chapter.

Explicit locks: Explicit locks, which you set in your program, maintain record locks that would otherwise be released after a change in currency. The KEEP statement and the KEEP clause of the FIND/OBTAIN statement are used to set explicit shared and exclusive locks.

►► For more information about setting explicit locks, see Chapter 4, “Navigational DML Programming Techniques” on page 4-1.

Managing record locks: Setting a large number of implicit or exclusive record locks during a recovery unit will hinder system performance. You can maintain efficient recovery units by regularly issuing the DML COMMIT statement (described later in this chapter).

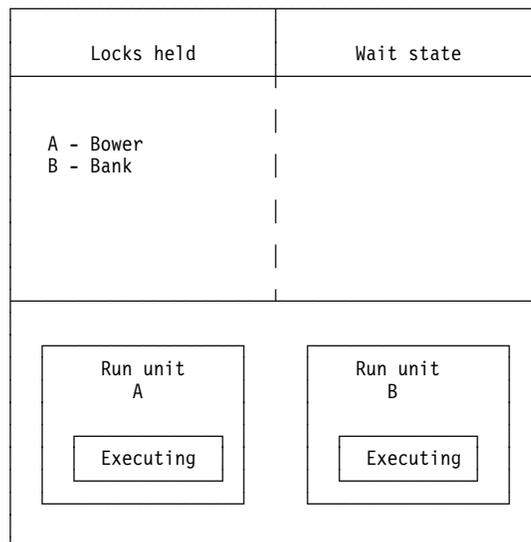
Additionally, certain conditions that result from the use of record locks can cause abnormal termination of run units executing under the central version:

- **Exceeded wait time** — A run unit waiting to set a record lock on a record that is currently held by another run unit abends if it exceeds the internal wait interval specified at central version generation. When this happens, the central version rolls back the recovery unit and returns a value of *nn69* to the ERROR-STATUS field in the IDMS communications block.
- **Deadlock** — If two or more run units would cause a deadlock were they all permitted to wait, one run unit is aborted to avoid the deadlock.

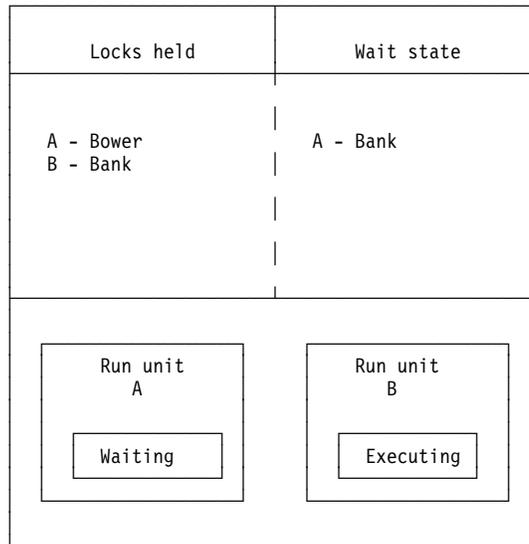
When a run unit is terminated because of a potential deadlock, the central version rolls back the recovery unit, returns a value of *nn29* to the ERROR-STATUS field in the IDMS communications block, and releases all locks held by the aborted run unit.

Deadlock example: The following sequence of figures shows a typical deadlock situation:

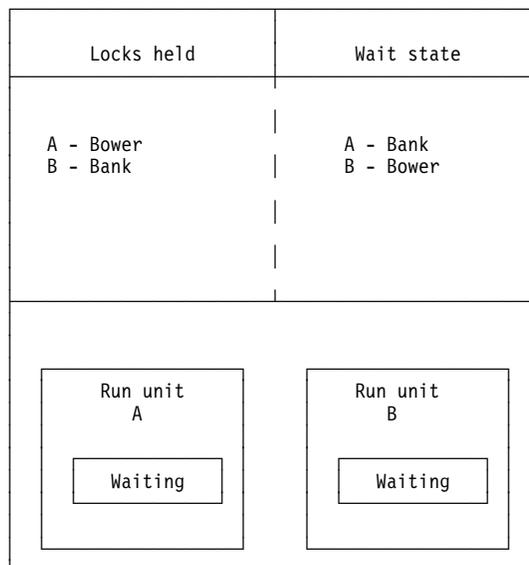
1. Run unit A and run unit B have placed shared locks (implicitly or explicitly) on the Bower EMPLOYEE record occurrence and the Bank EMPLOYEE record occurrence respectively; neither of these locks can be released until processing is complete.



2. Run unit A tries to place an exclusive lock on the Bank record (locked by run unit B); it is placed in a wait state.



3. Run unit B then attempts to place an exclusive lock on the Bower record (locked by run unit A) and deadlock results.



4. CA-IDMS automatically resolves the deadlock through a process of victim selection. In general, the younger of the two run units will be abended and rolled back unless the issuing task has a higher priority than that of the other issuing task involved in the deadlock.

►► For more information about deadlock detection and processing, refer to *CA-IDMS Database Administration*.

2.4.3 Area locks

Area in-use locks are examined whenever an area is opened in an update usage mode. These locks prevent run units originating in multiple regions or partitions (multiple local mode run units, multiple central versions, or a combination of both) from concurrently updating an area. Area in-use locks also prevent any access to an area that requires recovery of incomplete run units due to a local mode or central version abend.

Local mode: In local mode the area lock is checked as each area is readied in an update usage mode. If the lock is already set, a value of 0966 is returned to the ERROR-STATUS field in the IDMS communications block and access to the area is not allowed. If the lock is not set, the local mode run unit causes the lock to be set. If the run unit terminates abnormally (that is, without issuing a FINISH statement), the lock remains set. Further update access by subsequent local mode or central version run units is prevented until the area is recovered manually (using the FIX ARCHIVE and ROLLBACK utilities).

Central version: Each area defined for a central version is associated with an access mode. Access modes determine the availability of each area to run units running under the central version, to other central versions, and to programs running in local mode. The access modes are described below:

- **UPDATE (ONLINE)** indicates that areas are available for update to run units running under the central version. Run units running in local mode or other central versions cannot ready the area in any update usage mode.
- **RETRIEVAL** indicates that areas are available for retrieval to run units running under the central version. Run units running in local mode or under other central versions can ready the area in any usage mode.
- **OFFLINE** indicates that areas are not available for update or retrieval to run units running under the central version. Run units running in local mode or under other central versions can ready the area in any usage mode.

Note: The UPDATE, RETRIEVAL, and OFFLINE central version access modes are operator concerns; they are presented here as background information only. You do not have to address these modes in your program.

System startup: When the central version is started up, it checks the locks in all areas available for update. If any lock is found to be set, a warning message is displayed at the user's console and further access to that area is disallowed. The central version proceeds without the use of that area; any run unit attempting to ready that area receives a value of 0966 returned to the ERROR-STATUS field in the IDMS communications block. If the lock is removed after startup, the user must change the area status from OFFLINE to ONLINE to make the area available to the central version.

Note: Area locks are not set for individual run units running under the central version; run-unit conflicts are avoided by internal means.

2.4.4 Area usage modes

Run units can ready an individual area in a particular usage mode in order to define the scope of operations that can be performed against that area. The area usage modes, which are specified by the DML **READY** statement, are retrieval and update.

Retrieval: Retrieval specifies that the issuing run unit can perform only retrieval functions against records in that area. It can issue only the FIND, OBTAIN, and GET statements; it cannot issue the STORE, MODIFY, ERASE, CONNECT, or DISCONNECT statements.

Update: Update specifies that the issuing run unit can modify as well as retrieve records in that area. That is, it can issue all available DML statements.

Ready options: You can issue a ready option in conjunction with a usage mode to restrict retrieval or update of records by other run units executing concurrently under the same central version in the specified area. The ready options are:

- **Protected** indicates that other run units cannot ready the specified area in update usage mode and must wait until your run unit terminates. You cannot ready an area with the protected option if a concurrently executing run unit has readied the area in update mode.
- **Exclusive** indicates that other run units cannot ready the specified area in *any* usage mode and must wait until your run unit terminates. You cannot ready an area with the exclusive option if a concurrently executing run unit has readied the area in any usage mode.
- **Shared** indicates that more than one run unit running under the same central version can concurrently access the same area.

Note: You cannot explicitly code SHARED in the READY statement (that is, READY UPDATE is functionally the same as READY SHARED UPDATE).

Combinations of usage mode and ready options: The table below summarizes the effect that various combinations of usage modes and ready options have on concurrently executing run units.

The usage mode in which one run unit readies an area restricts the usage mode in which other run units executing under the same central version can ready that area. The usage modes in which run unit B can ready an area are shown, depending on the usage mode in which run unit A has readied the area. Y (yes) signifies that the second run unit can ready the area in the specified usage mode; N (no) signifies that it cannot.

		Run unit B					
		SHARED UPDATE	SHARED RETRIEVAL	PROTECTED UPDATE	PROTECTED RETRIEVAL	EXCLUSIVE UPDATE	EXCLUSIVE RETRIEVAL
Run unit A	SHARED UPDATE	Y	Y	N	N	N	N
	SHARED RETRIEVAL	Y	Y	Y	Y	N	N
	PROTECTED UPDATE	N	Y	N	N	N	N
	PROTECTED RETRIEVAL	N	Y	N	Y	N	N
	EXCLUSIVE UPDATE	N	N	N	N	N	N
	EXCLUSIVE RETRIEVAL	N	N	N	N	N	N

Wait state: When a run unit cannot ready an area because a protected or exclusive restriction is already placed on that area by another run unit running under the same central version, it is placed in a wait state until the first run unit is finished.

Automatic implicit locking: The central version automatically maintains implicit record locks. These record locks are dependent on DBA specifications and on the area usage mode specified. The following record locks can be maintained:

- **Shared update** — The central version always maintains record locks for run units executing in shared update usage mode.
- **Shared retrieval** — The central version maintains record locks for run units executing in shared retrieval mode only if specified by the DBA at system generation. If they are not maintained, a run unit with shared retrieval usage mode may yield unpredictable results if it accesses records being modified by a concurrently executing run unit with a shared update or protected update usage mode.
- **Protected update** — The central version maintains record locks for run units executing in protected update mode only if specified by the DBA at system generation.
- **Protected retrieval, exclusive update, exclusive retrieval** — The central version does not maintain implicit record locks for run units executing in these modes since the usage modes themselves prohibit concurrent update.

Default usage modes: Your DBA can assign default usage modes for subschema areas. The specified default determines the usage mode in which an area will automatically be readied for programs using that subschema. You do not have to code READY statements in programs that use such a subschema; however, if you issue a READY command for one area in the subschema, you must issue READY commands for all database areas to be accessed.

You can use the SUBAREA parameter of the IDMSRPTS utility to determine if the DBA has specified any default usage modes.

2.4.5 Recovery units

Every time your program modifies the database, a before and after image of the affected record occurrence is written to the journal file. These images are used in the event of program or system failure to recover (roll back) all changes made to the database. The database is rolled back; that is, any updates are reversed, to the last checkpoint written to the journal file.

Checkpoints: The following DML statements write checkpoints to the journal file:

- BIND RUN-UNIT
- FINISH
- COMMIT
- ROLLBACK

A recovery unit is that portion of a run unit that falls between two checkpoints.

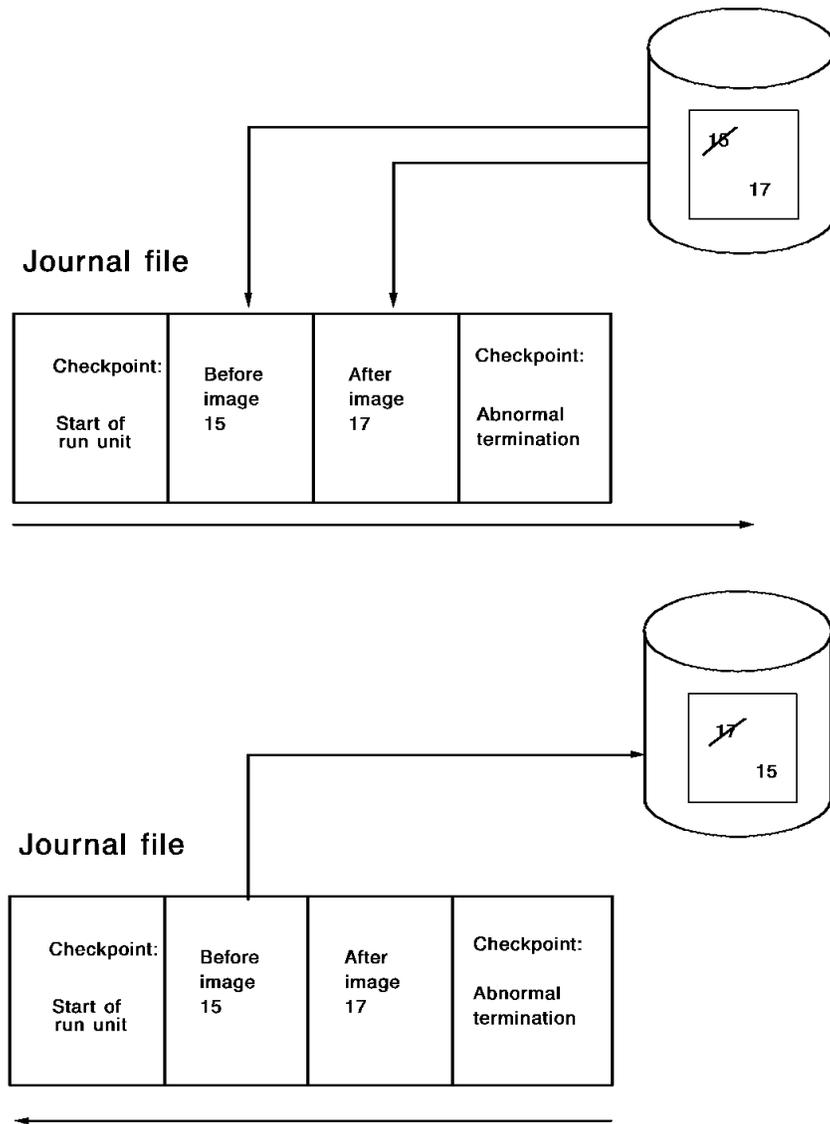
Automatic recovery under central version: Recovery is effected automatically for programs running under the central version. Under local mode you must manually restore the database by running the FIX ARCHIVE and ROLLBACK utilities.

►► For more information on restoring the database under local mode, refer to *CA-IDMS Database Administration*.

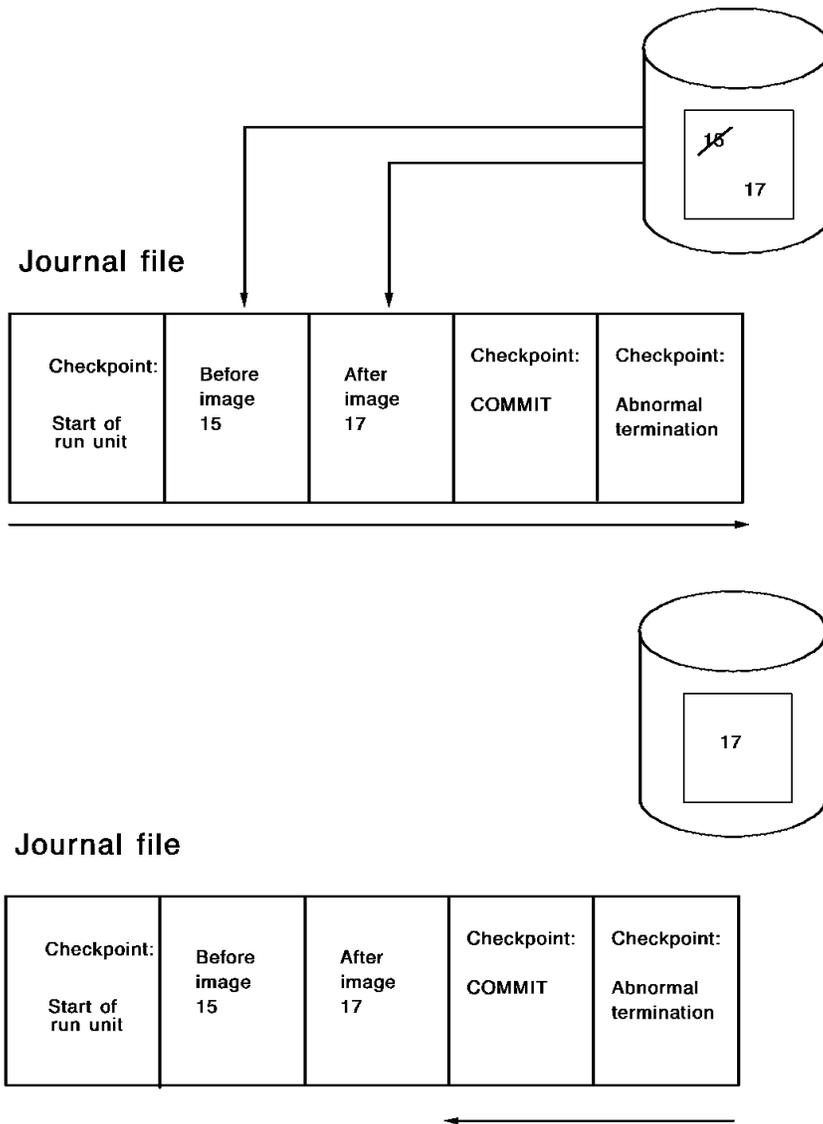
Use of COMMIT: If your application performs database updates, you should issue the COMMIT statement at regular intervals to:

- Release implicit locks held by the run unit
- Prevent needless rollback of valid database updates

Rolling back the database: The figure below shows journaling, checkpoints, and rollback. The BIND RUN-UNIT statement writes the initial checkpoint to the journal file. Before and after images are maintained for every modified record occurrence. In the event of an abend, the central version uses the before images to restore the database back to the last checkpoint. In this figure, run unit and recovery unit are synonymous.



Establishing checkpoints: The figure below shows the use of the COMMIT statement to establish checkpoints. In the event of an abend, the central version restores the database as far back as the last COMMIT checkpoint.



Frequency of COMMIT statements: Since modified records contain implicit exclusive locks, timely use of the COMMIT statement is an important programming consideration.

►► For detailed information on the implicit exclusive record locks maintained for each modified record, refer to *CA-IDMS Database Administration*.

The frequency of issuing COMMITs is a site- and application-specific decision. Some questions to ask when determining the frequency of COMMITs are:

- **Will a logical unit of work be completed?** — You should maintain implicit exclusive locks at least until a logical unit of work is complete. For example, if

you plan to DISCONNECT and subsequently CONNECT a record, you should not issue the COMMIT until after the CONNECT.

- **What is the application's operating environment?** — If there will be a high volume of concurrent online users, you should try to keep recovery units short either by issuing COMMITs more frequently or by maintaining short run units.
- **Is there a potential for growth?** — You might want to insert comments in your program that note logical places to issue COMMITs in anticipation of future program expansion.
- **How many locks will be held for each modified record?** — Additional implicit exclusive record locks (for example, on the NEXT and PRIOR records) are held when modifying symbolic keys, when erasing occurrences from the database, when connecting or disconnecting records, or when modifying variable-length records.

▶▶ For detailed information on the implicit exclusive record locks maintained for each modified record, refer to *CA-IDMS Database Administration*.

- **How many locks is too many?** — This is application- and site-specific, although you probably do not want to maintain implicit exclusive locks on more than 70-100 records at any one time.

Note: You can use the IDMS statistics block (explained in 2.5.5, “Collecting database statistics” on page 2-30 later in this chapter) to obtain lock-related information at runtime.

2.5 Basic programming considerations

There are some programming considerations common to all CA-IDMS navigational DML programmers, including communications blocks, currency, and statistics collection.

2.5.1 Establishing communications with CA-IDMS

To establish communications with CA-IDMS, you must code certain DML statements that perform housekeeping functions. These statements, which include compiler-directive and control statements, perform the following functions:

- Identify the operating mode
- Identify the subschema
- Copy record descriptions
- Define the run unit

Identifying the operating mode: Program operating modes (also called protocols) specify the manner in which the precompiler will generate CALL statements for CA-IDMS services and, optionally, whether the precompiler will generate DML sequence numbers in the output listing. The operating mode is specified by using the MODE parameter.

The standard CA-IDMS operating modes are:

- BATCH
- IDMS-DC
- DC-BATCH

Additionally, the precompiler can generate CALL statements that support teleprocessing monitors other than DC, such as CICS.

►► For a complete list of the teleprocessing monitors supported, refer to the language-specific CA-IDMS DML reference manual.

The DEBUG option: The precompiler option DEBUG specifies that, at runtime, a unique sequence number is placed in the DML-SEQUENCE field of the IDMS communications block after each DML call. You can reference these sequence numbers in the precompiler output listing to assist in program debugging.

The DEBUG option is a very useful debugging tool for test and production programs.

Identifying the subschema: If your program accesses the database, you must specify a subschema name and schema name.

►► For more information on subschema identification, refer to the discussion of compiler-directive statements in the language-specific CA-IDMS DML reference manual.

Copying record descriptions: You can use the COPY IDMS (INCLUDE IDMS in PL/I) compiler-directive statement to copy IDMS or non-IDMS record-description source code from the dictionary into your program. IDMS-related record description source code includes communications blocks, subschema records and names, map request blocks (MRBs), and map records. Optionally, the precompiler can automatically copy all IDMS-related record-description source code into program variable storage.

If your program accesses a large subschema but never references many of the records, you should copy subschema records manually in order to maintain a small load module. If your program accesses a small application-specific subschema, you should let the precompiler automatically copy the subschema records.

The COPY IDMS statement is also used to copy IDD-defined files (batch only), record descriptions, and executable module source.

COBOL programmers: When coding a COBOL OPEN command that names multiple files, you must first name all files defined to the dictionary before naming any files not defined to the dictionary. Alternatively, you can code each OPEN and CLOSE statement separately.

►► For more information about the COPY IDMS statement and source code requirements, refer to the language-specific CA-IDMS DML reference manual.

Defining a run unit: Your program must establish a run unit to access the database. While the run unit is active, your program can issue any number of DML statements and language-specific source statements. Issue the following DML statements to define a run unit:

1. Issue a **BIND RUN-UNIT** statement to establish addressability to the IDMS communications block and name the subschema to be loaded for the run unit. BIND RUN-UNIT can also name the system node under which the run unit will execute and identify the database to be accessed.

The BIND RUN-UNIT statement must be the first functional database DML call passed to the DBMS at execution time; it must logically precede all other database DML statements.

If program registration is in effect (that is, all programs must be registered in system generation before program execution), you must initialize the PROGRAM-NAME field in the IDMS communications block either manually or automatically before issuing the BIND RUN-UNIT statement:

- **Manually** — Explicitly move the program name to the PROGRAM-NAME field before the BIND RUN-UNIT statement is executed. In COBOL, for example:

```
MOVE 'EMPDISP' TO PROGRAM-NAME.
```

- **Automatically** — COPY IDMS SUBSCHEMA-BINDS (discussed later in this chapter) automatically moves the program name given in the PROGRAM-ID compiler directive to the PROGRAM-NAME field.

If your program contains more than one run unit (that is, it issues the BIND RUN-UNIT, READY, FINISH sequence more than once), you must reinitialize the ERROR-STATUS field in the IDMS communications block to the value 1400.

2. Issue one **BIND RECORD** statement for each database record to be accessed. Typically, BIND RECORD statements are issued immediately after the BIND RUN-UNIT statement.

The BIND RECORD statement establishes addressability for a subschema record in program variable storage. You must BIND all records that will be referenced by your application.

3. Issue the **READY** statement to prepare all database areas for access or, optionally, issue one READY statement for each database area to be accessed. Readyng areas individually gives you the following advantages:

- You need ready only those areas to be accessed either explicitly or implicitly.
- You can specify a different usage mode for each area.
 - ▶▶ For more information, see 2.4.4, “Area usage modes” on page 2-18 earlier in this chapter.
- You can perform the IDMS-STATUS routine after each area is readied to ensure that the statement was executed successfully.
 - ▶▶ For more information about performing the IDMS-STATUS routine, see 2.6, “IDMS communications block” on page 2-32 later in this chapter.

Typically, READY statements are issued immediately after the BIND RECORD statements.

Note: The area usage mode specified in the READY statement is an important factor in program and system performance. For example, a program running in exclusive update forces other programs to wait for it to relinquish control of the areas it holds; also, the program will be forced to wait for all other programs to finish using an area before being given exclusive access.

- ▶▶ For more information, see 2.4.4, “Area usage modes” on page 2-18 earlier in this chapter.

Although the READY statement can appear anywhere in your program, it is best to issue all READYS before issuing any other DML statements.

4. When database access is complete, issue a **FINISH** statement. FINISH relinquishes control of all associated database areas, writes statistical information for the database operations performed during run unit execution to the journal file, and defines and logs the end checkpoint for the recovery unit.

Checking the status of statement execution: You should perform the IDMS-STATUS routine after each BIND RUN-UNIT, BIND RECORD, and FINISH statement to ensure that it was executed successfully.

►► For more information about performing the IDMS-STATUS routine, see 2.6, “IDMS communications block” on page 2-32 later in this chapter.

COPY IDMS SUBSCHEMA-BINDS: You can issue the COPY IDMS SUBSCHEMA-BINDS statement instead of explicitly issuing the BIND RUN-UNIT and BIND RECORD statements. This statement automatically initializes the PROGRAM-NAME field in the IDMS communications block and copies a standard BIND RUN-UNIT statement and the appropriate BIND RECORD statements for each subschema record in program variable storage.

When binding several records to the same variable-storage location, COPY IDMS SUBSCHEMA-BINDS issues binds only for those records that are bound to their original location. In this case, code individual BIND RECORD statements for those records being bound to a separate location.

COBOL programmers: If AUTOSTATUS is in effect, a PERFORM IDMS-STATUS statement is automatically included after each BIND statement.

Since COPY IDMS SUBSCHEMA-BINDS does not check the status returned after each BIND, it should be used only for COBOL programs with AUTOSTATUS turned on.

Keeping run units short: It is a good programming practice to keep your run units as short as possible. This can be accomplished in the following ways:

- For **all run units**, issue the FINISH statement as soon as all database processing is complete.
- In a **batch program**, ensure that all input and output files have been opened successfully before issuing the BIND RUN-UNIT and READY statements.
- In an **online program**, perform all map-related processing (including error checking) before issuing the BIND RUN-UNIT and READY statements.

The program excerpt below shows a typical sequence of BIND RUN-UNIT, BIND RECORD, READY, and FINISH statements.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
  01 SWITCHES.
    05 EOF-SW          PIC X    VALUE 'N'.
    88 END-OF-FILE    VALUE 'Y'.
PROCEDURE DIVISION.
  OPEN INPUT DEPT-FILE-IN.
  OPEN OUTPUT DEPT-FILE-OUT.
  OPEN OUTPUT ERR-FILE-OUT.
  READ DEPT-FILE-IN AT END MOVE 'Y' TO EOF-SW.
  IF END-OF-FILE
    PERFORM EMPTY-INPUT-PROCESSING
  ELSE
    NEXT SENTENCE.
*
  MOVE 'DEPTRPT' TO PROGRAM-NAME.
  BIND RUN-UNIT.
  PERFORM IDMS-STATUS.
  BIND EMPLOYEE.
  PERFORM IDMS-STATUS.
  BIND DEPARTMENT.
  PERFORM IDMS-STATUS.
  BIND JOB.
  PERFORM IDMS-STATUS.
  BIND EMPOSITION.
  PERFORM IDMS-STATUS.
  BIND OFFICE.
  PERFORM IDMS-STATUS.
  READY.
  PERFORM IDMS-STATUS.
.
.
.
  DML statements
.
.
.
  FINISH.
  PERFORM IDMS-STATUS.
  GOBACK.
*
  IDMS-ABORT.
  EXIT.
  COPY IDMS IDMS-STATUS.

```

Binding maps and map records: Online programs that use mapping mode terminal-management statements must issue the following BIND statements to inform DC of the location of the map request block (MRB) and to initialize MRB fields:

- **BIND MAP** establishes addressability between DC and an MRB.
- **BIND MAP RECORD** establishes addressability for a map record in program variable storage.

The BIND MAP and BIND MAP RECORD statements are explained in detail in 7.2.1, “Housekeeping” on page 7-5.

2.5.2 Checking the status of statement execution

Communications blocks furnish status information about requested database and data communications services to the application program. The communications block that your program will use depends on the operating mode:

- The **IDMS communications block** is used when the operating mode is BATCH.
 - The **IDMS-DC communications block** is used when the operating mode is either IDMS-DC or DC-BATCH.
- ▶▶ For more information about the IDMS communications block, see 6.7, “Using the IDMS communications block” on page 6-11.

Checking status: The communications blocks are the interface between your program and CA-IDMS software components. Check the ERROR-STATUS field in the appropriate communications block after *every* DML statement.

Handling a status: If an unexpected value is returned to the ERROR-STATUS field, you should terminate run unit processing with a ROLLBACK statement rather than a FINISH. This will prevent incomplete updates from being applied to the database.

Error-checking with IDMS-STATUS: The dictionary provides an error-checking routine called IDMS-STATUS that can be copied into your program. This routine checks for a nonzero value in the ERROR-STATUS field and abends your program if such a value is found.

Before performing this routine, you should check the ERROR-STATUS field for zeros and for any acceptable nonzero values.

- ▶▶ For more information on the IDMS-STATUS routine, refer to:
- 2.6, “IDMS communications block” on page 2-32 later in this chapter
 - 6.7, “Using the IDMS communications block” on page 6-11

2.5.3 Specifying a dictnode or dictname for a run unit

You can specify a dictionary node and a dictionary name in a BIND RUN-UNIT statement to specify the dictionary load area containing the subschema. The dictionary node and name specifications function similarly to the database node and database name specifications. All settings remain in effect for the extent of the run unit.

Defaults: You can specify defaults using DICTNAME and DICTNODE SYSIDMS parameters in the JCL.

Overrides: The dictionary and database settings may occasionally be overridden by components of the runtime system.

The dictnode and dictname settings can be overridden as follows:

- **Online programs** — The setting can be overridden by:
 - A user exit 23, if implemented at your site
- **Batch programs** — The setting can be overridden by:
 - The IDMSOPTI module
 - The SYSCTL option (if ALWAYS is specified) of the system generation SYSTEM statement
 - A user exit 23, if applicable

Naming a dictionary: This COBOL statement names ASFDICT as the dictionary to be accessed by the run unit:

```
BIND RUN-UNIT DICTNAME ASFDICT
```

2.5.4 Using currency

The DBMS keeps track of the database location (db-key) of the most recently accessed record occurrence for the run unit, record type, set, and area. Each of these records is said to be **current** of run unit, record type, set, or area. Currency determines which record occurrences are affected by DML statements.

Advantages of using currency: In navigational DML programming, currency enables you to navigate the database with a minimum of effort.

CA-IDMS maintains currency for scratch and queue records referenced by programs running under the central version. Getting the next queue record is similar to obtaining the next database record.

►► For detailed information on using scratch and queue records, see Chapter 8, “Storage, Scratch, and Queue Management” on page 8-1.

2.5.5 Collecting database statistics

You can collect database runtime statistics with the ACCEPT DATABASE-STATISTICS statement. You can issue this statement any number of times during a run unit. It returns a copy of the IDMS statistics block to a specified location in program variable storage.

Although the ACCEPT DATABASE-STATISTICS statement can be issued any number of times during a run unit, IDMS statistics are cumulative; resetting of IDMS statistics block fields occurs only upon issuing a FINISH statement.

Uses of database statistics: Possible uses of database statistics include:

- Determining whether a variable-length record was stored on one page or fragments were placed in an overflow area
- Obtaining the date and time at the start and end of a run unit

- Keeping track of the number of update locks being held and issuing regular commits based on that statistic

►► For more information on collecting database runtime statistics and individual IDMS statistics block fields, refer to the language-specific CA-IDMS reference manual.

Performance Monitor users: You can use CA-IDMS Performance Monitor to collect statistics about program execution. For more information, refer to *CA-IDMS Performance Monitor User Guide*.

2.6 IDMS communications block

The IDMS communications block is the interface between your program and the DBMS. Whenever your program issues a call to the DBMS for a database operation, the DBMS returns information about the outcome of the requested service to the IDMS communications block.

►► For more information on the IDMS communications block, see 2.5.2, “Checking the status of statement execution” on page 2-29 earlier in this chapter.

Including the IDMS communications block: The data description (identified as SUBSCHEMA-CTRL) of the IDMS communications block is copied from the dictionary into program variable storage. When you submit the program to the precompiler, the IDMS communications block is copied automatically unless you specify that records are to be copied manually. In that case, your program explicitly copies in the data description from the dictionary.

IDMS communications block fields: You should take note of the following IDMS communications block fields:

- **PROGRAM-NAME** contains the name of the current program. It is a good programming practice to initialize this field at the beginning of every program.
- **ERROR-STATUS** contains a value that indicates the status of the last DML call.
- **DBKEY** contains the db-key of the last database record accessed by the program.
- **DML-SEQUENCE** contains the sequence number of the last DML statement executed by the program (only if the precompiler option DEBUG is in effect).

Your program should examine the ERROR-STATUS field in the IDMS communications block after *every* navigational DML call (except IF).

IDMS-STATUS routine: COBOL and PL/I programs check the ERROR-STATUS field by using the IDMS-STATUS routine, which can be copied in from the dictionary. You should perform the IDMS-STATUS routine after first checking for zeros and for any anticipated nonzero ERROR-STATUS values. Under an operating mode of BATCH, this routine checks the ERROR-STATUS field for zeros. If the routine finds a nonzero value, it displays selected values in the IDMS communications block and terminates the program.

Because IDMS-STATUS is a COBOL SECTION, you should copy it into the program using at least one of the following considerations in order to avoid runtime errors:

- Place IDMS-STATUS at the end of the program.
- Start a new SECTION after IDMS-STATUS.
- Perform IDMS-STATUS THRU ISABEX.

All examples in this manual assume that IDMS-STATUS is at the end of the program.

IDMS-STATUS in COBOL programs: COBOL programmers must include a paragraph entitled IDMS-ABORT, which is referenced by IDMS-STATUS to allow for further error processing.

Any operating mode that includes the AUTOSTATUS protocol automatically performs the IDMS-STATUS routine after every DML statement (except the IF statement). You can include one ON clause per DML statement to check for any anticipated nonzero return code.

►► For more information on AUTOSTATUS, refer to *CA-IDMS DML Reference - COBOL*.

Status checking in Assembler programs: The IDMS-STATUS routine is not available in Assembler. An Assembler program must have its own explicitly coded error-checking routines. For more information, see Appendix B, “Assembler Considerations” on page B-1.

Example of IDMS-STATUS routine: The example below shows the IDMS-STATUS routine used in a batch COBOL program to check the ERROR-STATUS field in the IDMS communications block for a value of zero. If a nonzero value is returned, the routine displays program status information, rolls back any changes made to the database, and aborts the program.

```

*****
IDMS-STATUS                                SECTION.
*****
      IF DB-STATUS-OK GO TO ISABEX.
      PERFORM IDMS-ABORT.
      DISPLAY '*****'
              ' ABORTING - ' PROGRAM-NAME
              ', '          ERROR-STATUS
              ', '          ERROR-RECORD
              ' **** RECOVER IDMS ****'
      UPON CONSOLE.
      DISPLAY 'PROGRAM NAME ---- ' PROGRAM-NAME.
      DISPLAY 'ERROR STATUS ---- ' ERROR-STATUS.
      DISPLAY 'ERROR RECORD ---- ' ERROR-RECORD.
      DISPLAY 'ERROR SET ----- ' ERROR-SET.
      DISPLAY 'ERROR AREA ----- ' ERROR-AREA.
      DISPLAY 'LAST GOOD RECORD - ' RECORD-NAME.
      DISPLAY 'LAST GOOD AREA --- ' AREA-NAME.
      DISPLAY 'DML SEQUENCE ---- ' DML-SEQUENCE.
      ROLLBACK.
ISABEX. EXIT.

```

►► For more information on IDMS-STATUS, refer to the language-specific CA-IDMS DML reference manual.

Chapter 3. Introduction to Database Access with Navigational DML

3.1 About this chapter	3-3
3.2 Records	3-4
3.2.1 Record name	3-4
3.2.2 Record identification	3-5
3.2.3 Storage mode	3-5
3.2.4 Record length	3-5
3.2.5 Location mode	3-5
3.2.6 Duplicates option	3-6
3.2.7 Area name	3-7
3.3 Sets	3-8
3.3.1 Set name	3-9
3.3.2 Set linkage	3-9
3.3.3 Set membership options	3-10
3.3.4 Set order	3-11
3.3.5 Chained and indexed sets	3-13
3.3.6 Set relationship definition	3-13
3.4 Data structure diagram	3-20
3.5 Currency	3-22
3.5.1 Use and updating of currency by DML verbs	3-23
3.5.2 Updating currencies during DML processing	3-25
3.6 Database access execution sequence	3-27

3.1 About this chapter

Navigational DML programming is associated with network database structures. You use currency to **navigate** the database records and sets, one record at a time. Because of this, navigational DML programming is also referred to as navigational programming.

This chapter discusses the following topics related to navigational DML programming:

- **Records** — A discussion of database records
- **Sets** — A discussion of chained and indexed sets, including system-owned indexes, set representation, set linkage, set membership options, and set order
- **Data structure diagram** — A discussion of data structure diagrams, including an illustration of the data structure diagram for the sample EMPLOYEE database
- **Currency** — A discussion of currency, including the currencies maintained for sets, records, areas, and run units
- **Database access execution sequence** — A discussion of the steps taken by the DBMS in accessing database records
- **IDMS communications block** — A discussion of the IDMS communication block and how your program interacts with it

3.2 Records

A record, which is a named collection of one or more elements, is assigned certain characteristics by the DBA. (For more information about record types and record occurrences, see 2.2, “Database components” on page 2-4.) These DBA-assigned characteristics are important considerations in navigational DML programming, particularly as a guide for determining database-access strategy.

Schematic representation: A record is represented in a data structure diagram by a rectangular box, as shown in the figure below. The contents of this box define record characteristics that you use to develop strategies for accessing and manipulating the record. Each of these characteristics is described separately in the following pages.

Record name			
Record identification	Storage mode	Record length	Location mode
Calc-key or VIA set name			Duplicates options
Area name			

3.2.1 Record name

The data name of a record in program variable storage is known as a record name. The record name identifies the record description to be copied from the dictionary into variable storage, either automatically or under program control during execution of the precompiler.

Denoting record type: *Record name* is used in DML statements to reference a record type. For example, you can add a new occurrence of the DEPARTMENT record type to the database by using the STORE statement:

```
STORE DEPARTMENT.
```

This statement causes the contents of DEPARTMENT in program variable storage to be stored in the database.

Assembler programmers: To access database records whose names contain more than eight characters or characters not included in the Assembler character set, see your DBA for language-specific synonyms. For more information on synonyms, refer to either 5.2, “Copying record definitions and their synonyms” on page 5-4 or the *CA-IDMS DML Reference - Assembler*.

3.2.2 Record identification

The number that serves as an internal identifier for the record type is the record identification. This number is supplied by the DBA and is used internally by the DBMS. You do not use the record identification when coding DML statements.

3.2.3 Storage mode

The DBMS can store occurrences of a record type with a fixed or variable length. It can also store occurrences in compressed format. The following codes represent a record's storage mode:

- **F** stands for fixed length.
- **V** stands for variable length.
- **C** stands for compressed length.

For example, a storage mode of FC indicates that records of the type being described are fixed length and compressed.

When modifying or storing variable-length records, you should keep in mind that implicit exclusive record locks are held for each page on which a fragment of a variable-length record is stored. This should be a factor in calculating the frequency of COMMITs.

3.2.4 Record length

You can use the record length field to determine the actual number of bytes in a fixed-length record or the maximum or average number of bytes in a variable-length record.

To minimize load module (phase) size, you may not want to copy large subschema records that are not referenced by your program into variable storage.

3.2.5 Location mode

How the DBMS stores a record in a database area is determined by its location mode. The location modes are: CALC, VIA, or DIRECT.

CALC: A record with a location mode of CALC is stored on a page calculated by the DBMS. The value of a designated record element or concatenation of record elements called a CALC key determines the placement of the record. The word CALC in the location mode entry and the name of the CALC key in the record representation signify that the location mode of a record is CALC.

Use of the CALC location mode evenly distributes records over an area by means of a randomization routine. This distribution enables direct retrieval of a record with a single access.

Suppose you need to *store* an occurrence of the EMPLOYEE record in the database and EMP-ID is the CALC key. The DBMS uses the value of EMP-ID to determine the database page on which to store the record.

You can *retrieve* a given EMPLOYEE occurrence by moving a value into the EMP-ID field in variable storage before executing the DML FIND/OBTAIN statement. The DBMS uses the value of EMP-ID to determine the database page number on which to start its search for the specified EMPLOYEE record.

►► For more information on using CALC records, see 4.2.1, “Accessing CALC records” on page 4-4.

VIA: A record with a location mode of VIA is stored relative to another database record. A VIA record is stored as close as possible to the current record of set.

►► For more information on sets, see 3.3, “Sets” on page 3-8 later in this chapter.

The word VIA in the location mode entry and the name of the VIA set in the record representation signify that the location mode of a record is VIA.

The VIA configuration groups records that are likely to be accessed at the same time, either on the same page or on as few pages as possible, thereby minimizing the number of disk accesses needed to retrieve the records.

DIRECT: A record with a location mode of DIRECT is placed on or near a user-specified page. DIRECT location mode is designated in the record representation with the word DIRECT in the location mode entry.

VSAM records: Native VSAM records have a location mode of VSAM; if the records are to be accessed by means of a CALC key, they must have a location mode of VSAM CALC.

►► For more information on how the DBMS stores records, refer to *CA-IDMS Database Administration*.

3.2.6 Duplicates option

You can store CALC records with identical keys based on the duplicates option. This option specifies whether records with duplicate CALC keys are allowed and whether they are to be positioned logically before (FIRST) or after (LAST) duplicate records already existing in the area. The following codes represent the duplicates options:

DF option: DUPLICATES FIRST (DF) indicates that duplicates are positioned before previously existing records with the same CALC key.

DL option: DUPLICATES LAST (DL) indicates that duplicates are positioned after previously existing records with the same CALC key.

DN option: DUPLICATES NOT ALLOWED (DN) indicates that duplicates are not allowed.

DU option: DUPLICATES UNORDERED (DU) indicates that duplicates are stored in first-in first-out (FIFO) order (VSAM CALC only).

VSAM users only: Native VSAM records with a VSAM CALC location mode are assigned a duplicates option of DN or DU. DU signifies that record occurrences with duplicate CALC keys are allowed and are always retrieved in the order in which they were stored.

3.2.7 Area name

To determine the database area in which all occurrences of a record type are stored, refer to the area name field. You can use this name to determine the areas to read in your program and when retrieving records by using an area sweep. For information on performing an area sweep, see 4.2.4, “Performing an area sweep” on page 4-10.

3.3 Sets

Sets either establish relationships between record types or place an index on a record type. A set consists of an owner record type and a member record type.

A set occurrence consists of one occurrence of the owner record type, an index (indexed sets only), and any number of member record occurrences. When an owner occurrence exists with no member occurrences, the set is said to be empty.

Types of set: CA-IDMS supports the following types of sets:

- **Chained** sets consist of an owner record type and one or more member record types that are linked together by pointers.
- **Indexed** sets consist of an owner record type, an index, and a member record type. The owner of an indexed set can be one of the following types of records:
 - A **user-defined record** can be any record type defined in the schema.
 - A **system-owner record** is a system record that acts as an owner record for each indexed set that does not have a user-defined owner (an index with a system-owner record is called a **system-owned index**).

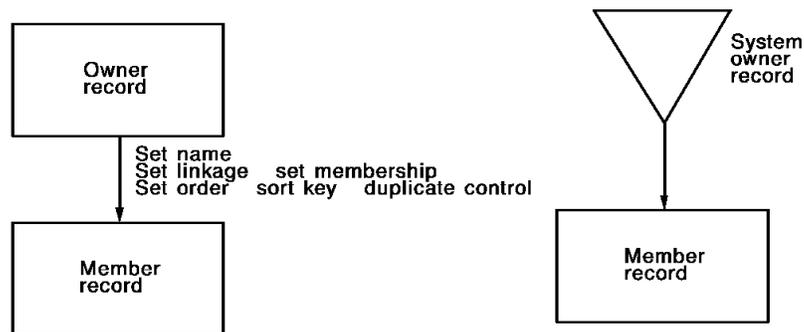
Walking a set: In chained and indexed sets, you can access the owner record and sequentially access each member record until you come to the owner record again. This process is referred to as walking a set. You do not have to access the owner record for a system-owned index before accessing the member record occurrences.

Schematic representation: A set relationship is represented in a data structure diagram by an arrow.

►► For the data structure diagram of the sample EMPLOYEE database, see 3.4, “Data structure diagram” on page 3-20 later in this chapter.)

The tail of the arrow representing a set touches the owner record type; the point of the arrow touches the member record type. Each set representation consists of the set name, its linkage (that is, index, next, prior, or owner), membership options, and the order in which member records are connected to the set.

The figure below shows these set characteristics:



3.3.1 Set name

A unique set type in the database is identified by its set name. A set name typically consists of the owner record name followed by the member record name. This set name is used whenever a set name is required in a DML statement.

3.3.2 Set linkage

The types of pointers that are present in a set's records are identified by the set linkage. Set linkage is represented in the set diagram by one or more of the following codes:

- I (indexed)
- N (next)
- P (prior)
- O (owner)

I (indexed): I specifies that the set is indexed; the index facilitates all access to member record occurrences. The owner of the index can be either a system record (designated by a triangle) or a database record (index between two record types).

The DML FIND/OBTAIN NEXT/PRIOR WITHIN SET statement can be used to access member record occurrences in any indexed set occurrence (for details on accessing indexed sets, see 4.2.8, "Accessing indexed records" on page 4-18).

N (next): N specifies that each record in the set contains a pointer that identifies the next record occurrence. These pointers allow access to member records in a set occurrence in the next direction but not in the prior direction.

Next pointers are required for chained sets.

The DML FIND/OBTAIN NEXT WITHIN SET statement can be used to access member record occurrences in any set occurrence (for details on accessing chained

sets, see 4.2.2, “Walking a set” on page 4-5). By starting with the owner record, you can access the first member record through the next pointer in the owner record. Each subsequent member record is accessed by using the next pointer in the current member until the owner record is encountered.

P (prior): P specifies that each record in the set contains a pointer that identifies the prior record occurrence. This type of set linkage permits retrieval of records in the set in the prior direction, through use of the FIND/OBTAIN PRIOR WITHIN SET statement. Additionally, prior pointers can help minimize logically deleted records, thus saving I/O in database update applications.

Prior pointers are not used with indexed sets.

O (owner): O specifies that each record in the set contains a pointer that identifies the owner record occurrence.

Owner linkage permits direct access to the owner record of a set from a member record through use of the FIND/OBTAIN OWNER statement. If an owner pointer is not present in each member record, the DBMS must access all records in the set iteratively until the owner of the set is encountered. Use of owner pointers improves performance, especially when the set contains a large number of member occurrences.

Valid linkage-option combinations: Valid linkage-option combinations are: I, IO, N, NP, NO, and NPO. Note that I and N are mutually exclusive: I indicates an indexed set; N indicates a chained set.

3.3.3 Set membership options

The manner in which a member record is connected to or disconnected from a set occurrence is indicated by its set membership options. This specification affects the use of the DML STORE, CONNECT, DISCONNECT, and ERASE statements.

►► For more information about using DML verbs that involve the set membership option, see 4.5, “Updating the database” on page 4-35.

Set membership is illustrated by a two-letter symbol. The first letter indicates the manner in which a record is *disconnected* from a set. The second letter indicates the manner in which a record is *connected* to a set.

Disconnect options

- **Mandatory (M)** — A record occurrence can be disconnected from the set only if it is erased from the database. A DISCONNECT statement issued against member records in a mandatory set will cause a non-zero error status to be returned.
- **Optional (O)** — A record occurrence can be disconnected from a set without being erased from the database. The occurrence can remain connected to other sets and can be reconnected to other occurrences of the same set.

Connect options

- **Automatic (A)** — The DBMS automatically connects a member record to a set when the member record occurrence is stored in the database.
- **Manual (M)** — The DBMS does not automatically connect a member record to a set when the member record occurrence is stored. Membership must be established explicitly with the DML CONNECT statement.

Combinations of set membership options: Disconnect and connect options are combined to form set membership options. The following codes represent the various combinations:

MA indicates that the set is mandatory automatic.

MM indicates that the set is mandatory manual.

OA indicates that the set is optional automatic.

OM indicates that the set is optional manual.

3.3.4 Set order

The logical order in which a member record occurrence is connected within a set occurrence is indicated by its set order.

Set order is independent of the physical placement of the records themselves. For example, a record could be physically first on a database page, but logically last in the set.

Set order options: The set order options are as follows:

- FIRST
- LAST
- PRIOR
- SORTED (ASCending or DEScending)

FIRST: FIRST means each new member record occurrence is positioned in a set immediately following (in the next direction) the owner record. Using the FIND/OBTAIN NEXT WITHIN SET statement, the last member record stored becomes the first record accessed. This is equivalent to last in, first out (LIFO).

LAST: LAST means each new member record occurrence is positioned in a set immediately before (in the prior direction) the owner record. This is equivalent to first in, first out (FIFO). Prior pointers are required for a chained set whose set order is LAST.

NEXT: NEXT means each new member record occurrence is connected immediately following (in the next direction) the record occurrence last accessed by the program.

PRIOR: PRIOR means each new member record occurrence is connected immediately before (in the prior direction) the record occurrence last selected by the program. Prior pointers are required for a chained set whose set order is PRIOR. The NEXT and PRIOR set order options allow you to assign the exact logical position of new records in a set.

SORTED: SORTED means each new member record occurrence is connected to the set in ascending or descending sequence based on the value of a designated element or group of elements contained in the record occurrence. The designated element (or group of elements) is called the sort key.

Document convention: Throughout this manual, references to sort key apply equally to a single sort-control element or a group of elements.

When a record is connected to the set, the DBMS examines the sort key in each member occurrence of the same record type to determine the logical position of the new record in the set.

Note: Sets are sorted based on EBCDIC sequence.

Sort sequence: The order of a sorted set is represented by the words **ASC** (ascending) or **DES** (descending), followed by the name of the sort-control element or elements.

Duplicates option for a sorted set: When the value of the named sort key in a record to be connected to a set occurrence matches the sort-key value in an existing record occurrence that is already a member of the set, the duplicate-control option indicates the action to be taken by the DBMS.

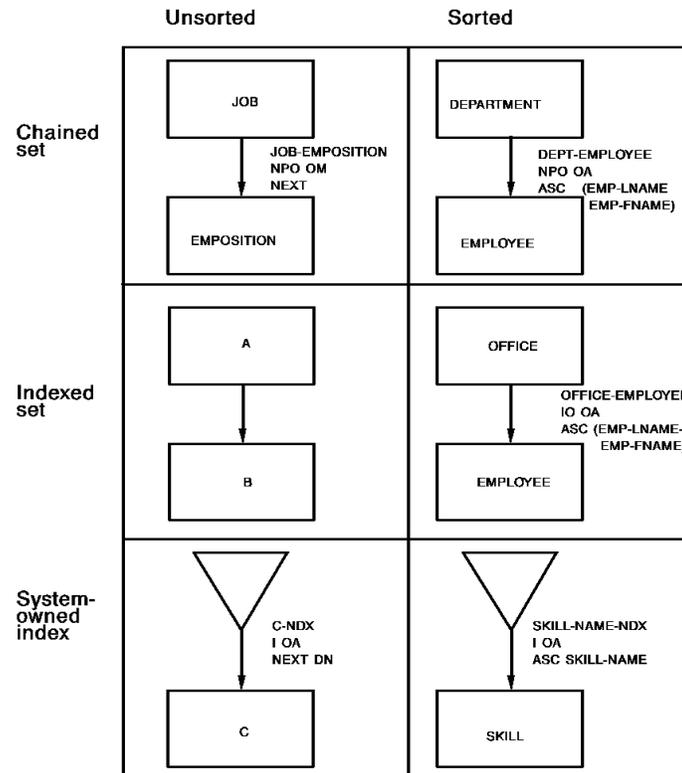
The duplicates options for sorted sets are:

- **DUPLICATES FIRST (DF)** — A record with a duplicate sort key is stored immediately before the existing duplicate record in the logical sequence of the set. The first duplicate record to be encountered in the next direction is always the duplicate most recently stored.
- **DUPLICATES LAST (DL)** — A record with a duplicate sort key is stored immediately after the existing duplicate record in the logical sequence of the set. When accessing records in the next direction of the set, the last duplicate record is the duplicate most recently stored.
- **DUPLICATES NOT ALLOWED (DN)** — A record cannot be stored in a set occurrence when an existing record already contains the same sort key. If a program attempts to store a duplicate record, the DBMS returns an error code to the ERROR-STATUS field in the IDMS communications block.

Native VSAM users: Native VSAM sorted sets are specified as **DUPLICATES NOT ALLOWED (DN)** or **DUPLICATES UNORDERED (DU)**. DU means that record occurrences are always retrieved in the order in which they were stored regardless of the direction in which the set is searched.

3.3.5 Chained and indexed sets

The figure below shows chained and indexed set relationships, both sorted and unsorted.



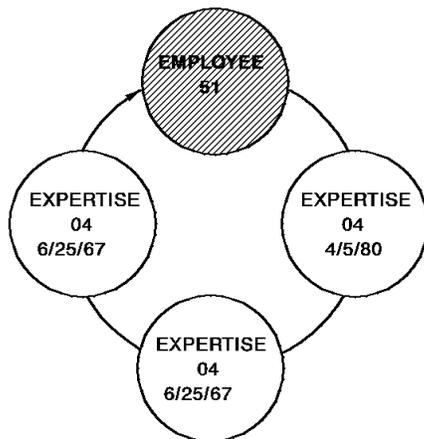
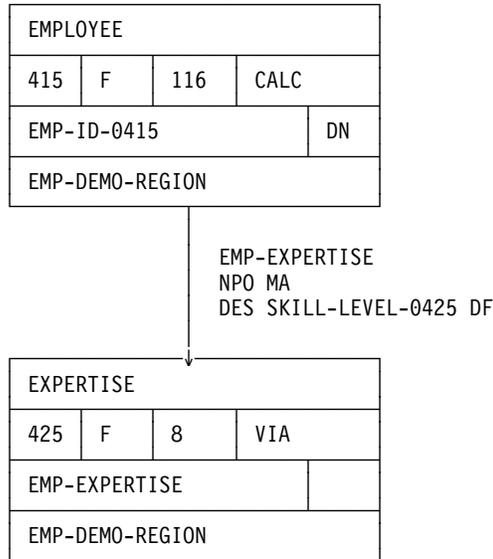
3.3.6 Set relationship definition

Set relationships are defined according to the following rules:

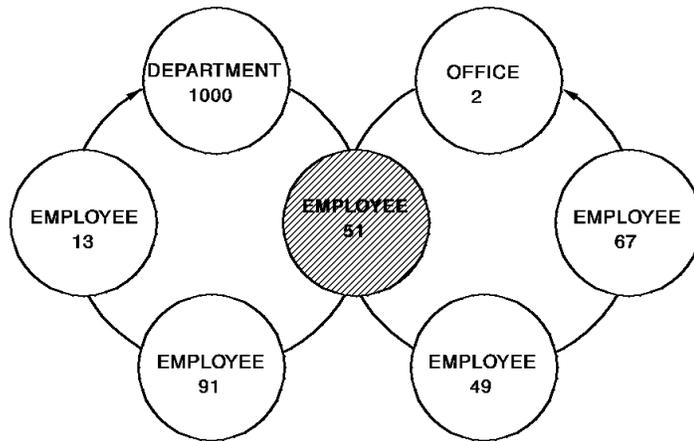
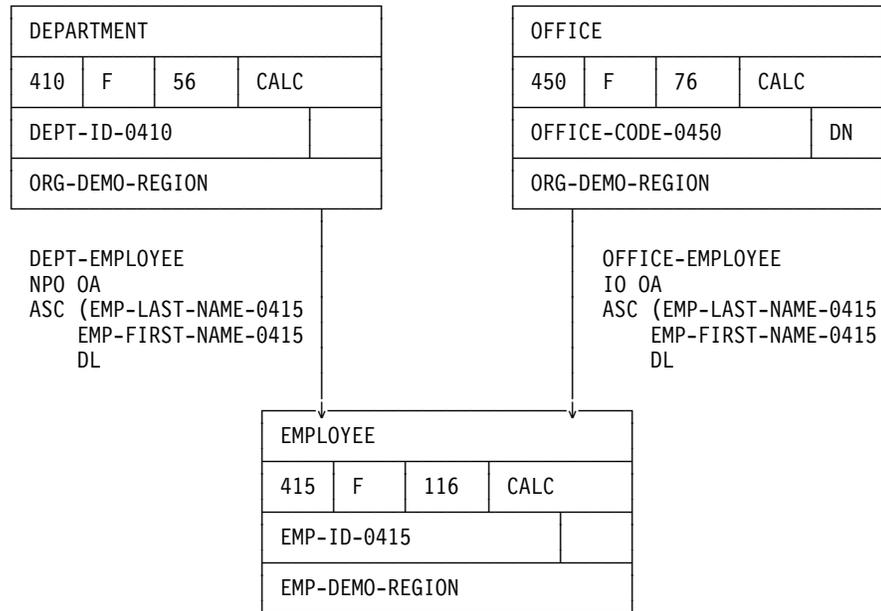
- Any record type can participate as a member in one or more sets.
- Any record type can be the owner of one or more sets.
- Any record type can participate as a member in one set and the owner in another.
- An owner record can own the same member record in more than one set (for example, in a bill-of-materials structure).
- A record cannot be both owner and member in the same set.
- A record need not participate in any set.

The figures that follow illustrate typical types of set relationships.

A record as the owner of a single set: In the figure below, the EMPLOYEE record is the owner of the EMP-EXPERTISE set:

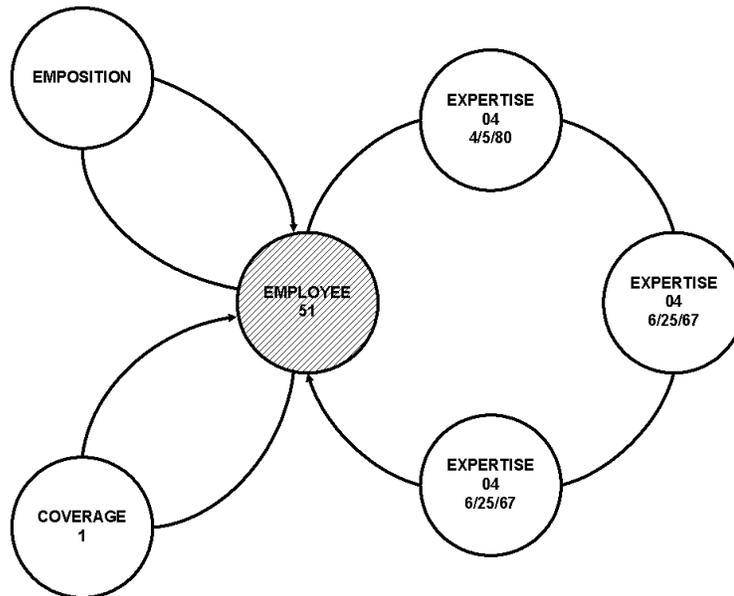
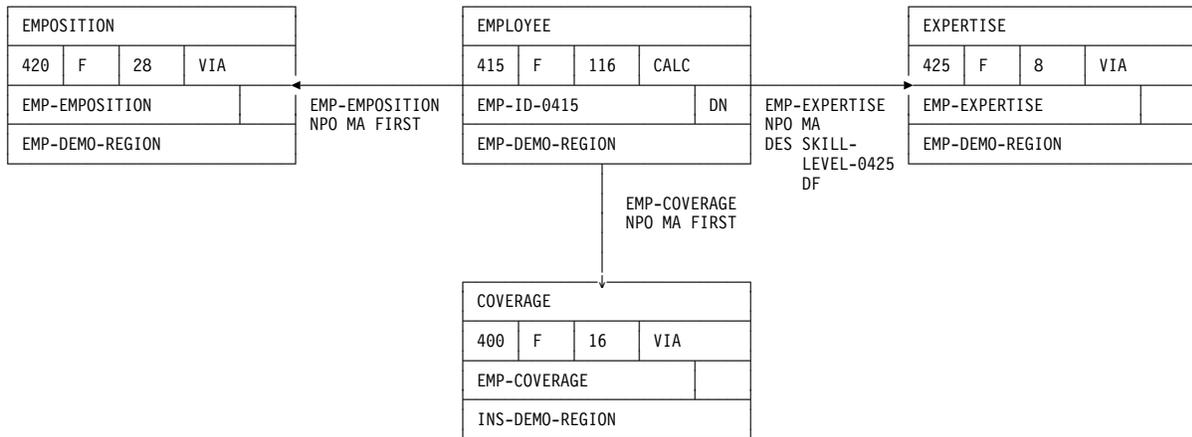


A record as a member in two sets: In the figure below, the EMPLOYEE record participates as a member in the DEPT-EMPLOYEE and OFFICE-EMPLOYEE sets:

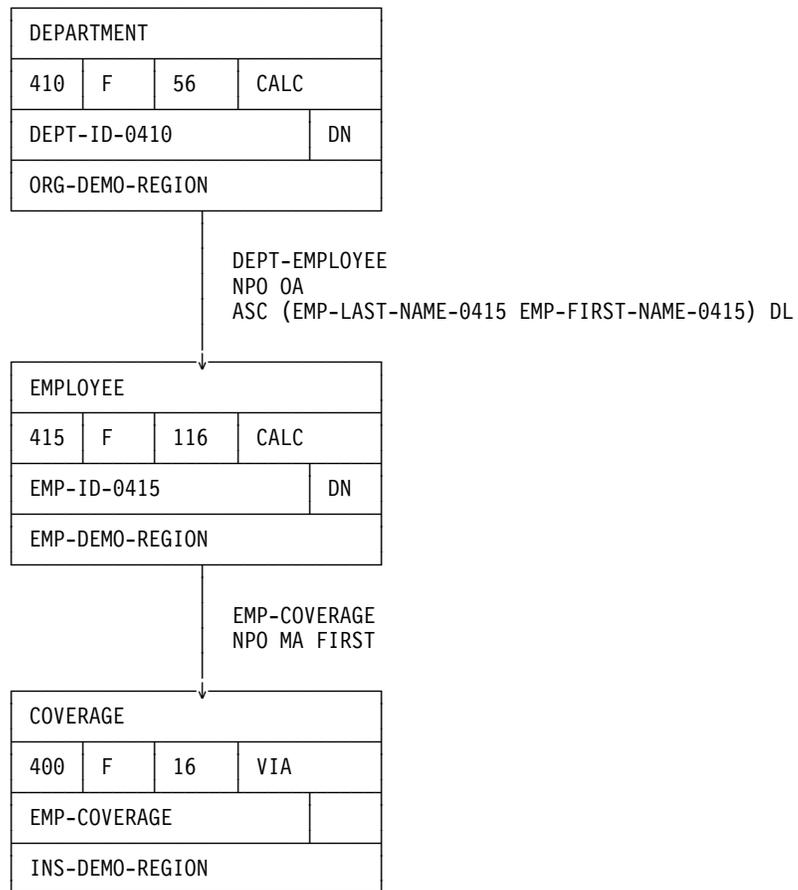


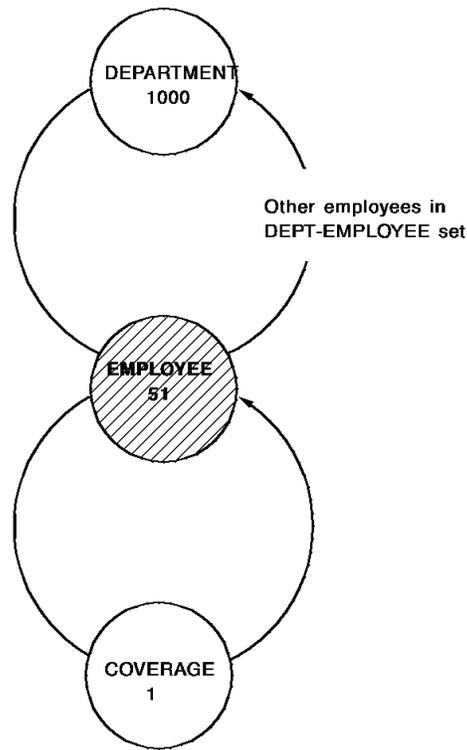
A record as the owner of multiple sets: In the figure below, the EMPLOYEE record is the owner of the EMP-EMPOSITION, EMP-COVERAGE, and EMP-EXPERTISE sets:

3.3 Sets

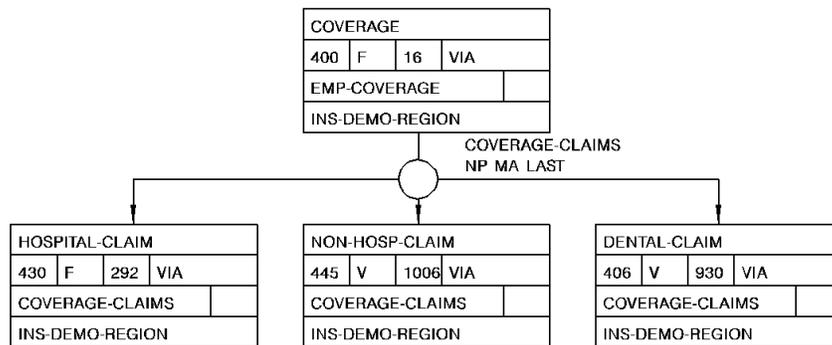


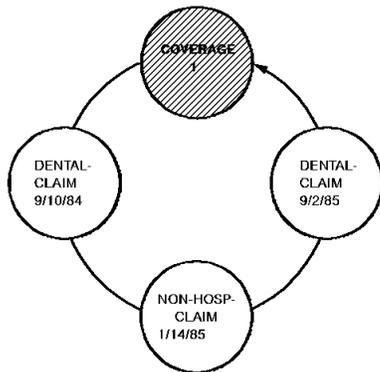
A record as a member and an owner: The EMPLOYEE record is a member in the DEPT-EMPLOYEE set and the owner of the EMP-COVERAGE set:



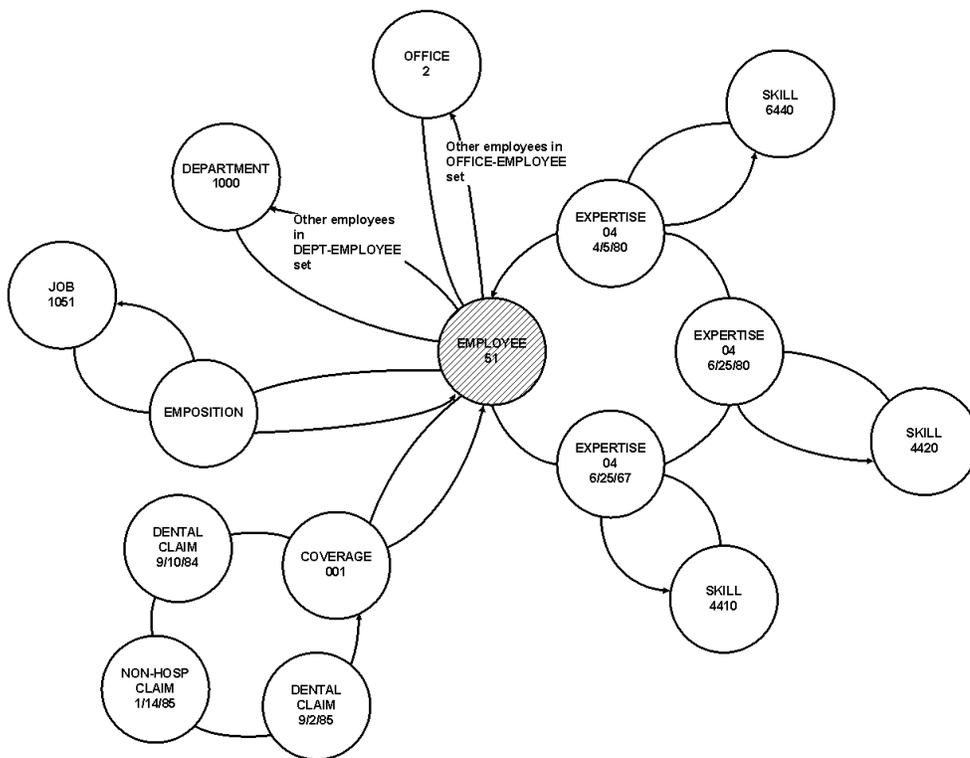


Multiple record types as members of one set: In the figure below, the COVERAGE record is the owner of the COVERAGE-CLAIMS set, which contains HOSPITAL-CLAIM, NON-HOSP-CLAIM, and DENTAL-CLAIM member records:





All record occurrences related to one occurrence: The figure below shows all database record occurrences that are related to one occurrence (EMPLOYEE 51) of the EMPLOYEE record type:



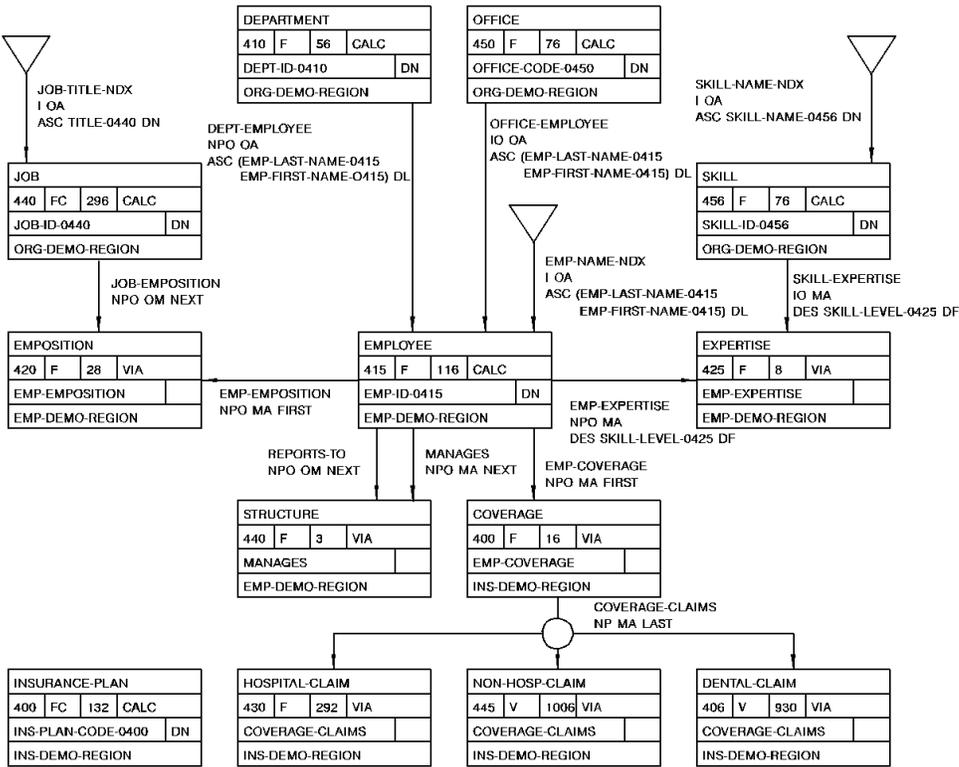
3.4 Data structure diagram

When all records, sets, and set relationships in the database have been defined, the database can be represented in a data structure diagram. This diagram:

- Contains a representative box for each database record defined to the subschema
- Lists set characteristics for all sets defined to the subschema
- Serves as a resource when designing and coding your application program

Employee information subschema: The data structure diagram for the employee information subschema, appearing below, is the basis for most of the examples in this manual. This subschema features:

- **System-owned indexes;** for example, the SKILL-NAME-NDX set
- **Indexes between two record types;** for example, the OFFICE-EMPLOYEE set
- **A Bill-of-materials structure;** for example, the MANAGES and REPORTS-TO sets
- **Sorted chained sets;** for example, the DEPT-EMPLOYEE set
- **A set that consists of multiple record types;** for example, the COVERAGE-CLAIMS set
- **A stand-alone record;** for example, the INSURANCE-PLAN record



3.5 Currency

During the execution of your application program, the DBMS uses **currency** to keep track of the database location (db-key) of the most recently accessed record occurrences for the run unit, record type, set, and area. By keeping track of the most recently accessed records, currency enables you to navigate the database with a minimum of effort. Currency values determine which record occurrences are affected by DML functions requested by an application program. Upon successful execution of a DML statement, the DBMS automatically updates currency values, as appropriate.

A record occurrence can be:

- Current of run unit
- Current of record type
- Current of set
- Current of area

Current of run unit: The record occurrence that was the object of the most recent successful FIND, OBTAIN, CONNECT, STORE, MODIFY, or ERASE function is current of run unit. Only one current record of run unit exists at any given time during program execution. That record's db-key, record type and qualifying page information are placed in the DBKEY, RECORD-NAME and PAGE-INFO fields of the IDMS communications block.

►► For more information on the IDMS communications block, see 2.5.2, “Checking the status of statement execution” on page 2-29 earlier in this chapter.

Current of record type: The most recently accessed occurrence of each record type is current of that record type. At any given time during program execution, one current record can exist for each record type defined in the program's subschema. For example, when your program successfully retrieves JOB 2215, that record becomes current of the JOB record type. If you then successfully obtain EMPLOYEE 466, that record becomes current of the EMPLOYEE record type; currency for the JOB record type remains unchanged.

Current of set: The most recently accessed record occurrence in each set is current of set for that set. At any given time during program execution, one current record can exist for each set defined in the program's subschema.

Because a successfully accessed record becomes the current record of all sets in which it participates as either owner or member, a given record occurrence can be the current record of any number of sets.

Current of area: The most recently accessed record occurrence in each area is current of area for that area. At any given time during program execution, one current record can exist for each area defined to the program's subschema.

When currency is established: At the beginning of a program, all currencies are null. Currency is established by the DML FIND, OBTAIN, RETURN, or STORE function. Currency is updated following each successful execution of a FIND, OBTAIN, CONNECT, DISCONNECT, ERASE, RETURN, MODIFY, or STORE statement.

How the DBMS uses currency: The DBMS uses currency to:

- Establish a starting point for the execution of a DML retrieval statement by using the current position in the database with respect to run unit, record, set, or area
 - Establish proper set occurrences for CONNECT and DISCONNECT functions
 - Determine the target record for a MODIFY statement
 - Determine the physical placement in the database of records stored with a location mode of VIA
 - Provide the basis for saving the db-keys of located records for subsequent use by the program
 - Set locks on the current of record, set, or area to prevent concurrent retrieval or update of records by different run units
- For further information about setting locks, see 4.6, “Locking records” on page 4-48.

3.5.1 Use and updating of currency by DML verbs

The table below outlines the currency required to execute each DML verb and the changes to currency following the successful execution of that verb. The bullet symbol (•) indicates currency used in command execution.

Note: BIND and READY do not use or update currency, but both verbs must be issued before any database access is attempted.

DML verb	Run unit	Record	Set	Area	Currency updated by successful execution
ACCEPT*	•	•	•	•	None
IF*	•		•		None
FIND/OBTAIN DB-KEY					All
FIND/OBTAIN CURRENT*	•	•	•	•	All
FIND/OBTAIN WITHIN SET ₁			•		All
FIND/OBTAIN WITHIN AREA				**	All

3.5 Currency

DML verb	Run unit	Record	Set	Area	Currency updated by successful execution
FIND/OBTAIN OWNER			•		All
FIND/OBTAIN CALC					All
FIND/OBTAIN DUPLICATE		•			All
FIND/OBTAIN USING SORT KEY ₂			•		All
GET	•				None
RETURN ₃			•		Set
STORE			***		All
MODIFY	•				None ₄
ERASE	•				Nullifies currencies of all record types and sets involved
CONNECT		•	•		Run unit, set
DISCONNECT		•			Nullifies currency of object set; updates current of run unit and area
KEEP*	•	•	•	•	None
COMMIT					None
COMMIT ALL					Nullifies all currencies
ROLLBACK					Nullifies all currencies
ROLLBACK CONTINUE					Nullifies all currencies
FINISH					Nullifies all currencies

* Uses only one currency as determined by command format.

** Required for NEXT and PRIOR formats only.

*** All in which record type participates as an automatic member.

₁ Currency is not required if the statement specifies FIRST, LAST or sequence-number for a system-owned indexed set.

₂ Currency is not required for a system-owned indexed set.

₃ Currency is not required if the statement specifies FIRST, LAST, or USING index-key.

₄ Except in the case of a sorted set.

3.5.2 Updating currencies during DML processing

The figure below shows the updating of currencies in the sample employee database following successful execution of a series of DML statements.

►► For further details on the use and update of currencies by each DML statement, refer either to Chapter 4, “Navigational DML Programming Techniques” on page 4-1 or to the language-specific CA-IDMS DML reference manual.

As record occurrences are accessed, run unit, record, set, and area currencies are established and updated. Boxes containing an asterisk (*) indicate changed currencies. When ERROR-STATUS contains an 0307 status code (end-of-set), the owner of the specified set becomes current of run unit, area, and its record and set types.

►► For more information on the records, sets, and areas involved, refer to the EMPLOYEE database data structure diagram in 3.4, “Data structure diagram” on page 3-20 earlier in this chapter.

3.5 Currency

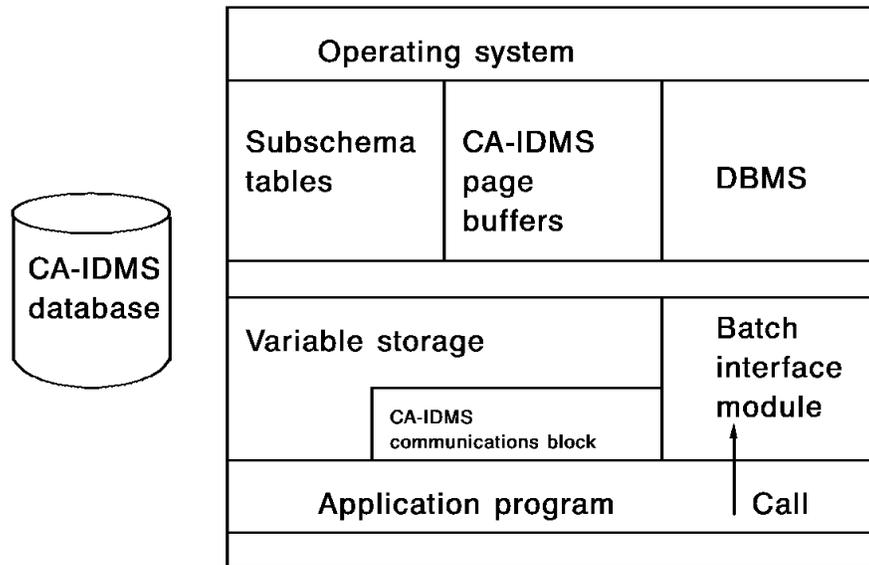
		R. U. curr.	Record currencies					Set currencies				Area currencies		
	ERROR STATUS	RUN UNIT	DEPARTMENT	EMPLOYEE	EMPOSITION	JOB	OFFICE	DEPT-EMPLOYEE	EMP-POSITION	JOB-EMPOSITION	OFFICE-EMPLOYEE	ORG-DEMO-REGION	EMP-DEMO-REGION	INS-DEMO-REGION
BIND RUN UNIT.	0000													
READY SHARED UPDATE.	0000													
OBTAIN FIRST DEPARTMENT WITHIN ORG-DEMO-REGION.	0000	5200*	5200*					5200*				5200*		
OBTAIN NEXT EMPLOYEE WITHIN DEPT-EMPLOYEE.	0000	479*	5200	479*				479*	479*		479*	5200	479*	
OBTAIN OWNER WITHIN OFFICE-EMPLOYEE.	0000	5*	5200	479			5*	479	479		5*	5*	479	
DISCONNECT EMPLOYEE FROM OFFICE-EMPLOYEE.	0000	479*	5200	479			5	479	479		NPO*	5	479	
MOVE 8 TO OFF-CODE-0450.														
FIND CALC OFFICE.	0000	8*	5200	479			8*	479	479		8*	8*	479	
CONNECT EMPLOYEE TO OFFICE-EMPLOYEE.	0000	479*	5200	479			8	479	479		479*	8	479	
OBTAIN NEXT EMPOSITION WITHIN EMP-EMPOSITION.	0000	43*	5200	479	43*		8	479	43*	43*	479	8	43*	
OBTAIN OWNER WITHIN JOB-EMPOSITION.	0000	5035*	5200	479	43	5035*	8	479	43	5035*	479	5035*	43	
MODIFY JOB.	0000	5035	5200	479	43	5035	8	479	43	5035	479	5035	43	
OBTAIN NEXT EMPLOYEE WITHIN DEPT-EMPLOYEE.	0000	329*	5200	329*	43	5035	8	329*	329*	5035	329*	5035	329*	
FIND LAST EMPOSITION WITHIN EMP-EMPOSITION.	0000	52*	5200	329	52*	5035	8	329	52*	52*	329	5035	52*	
ERASE EMPOSITION.	0000	52#	5200	329	Null*	5035	8	329	NPO*	NPO*	329	5035	52#	
FIND LAST EMPOSITION WITHIN EMP-EMPOSITION.	0307	329*	5200	329	Null	5035	8	329	329*	NPO*	329	5035	329*	
STORE EMPOSITION.	0000	53*	5200	329	53*	5035	8	329	53*	NPO	329	5035	52*	
CONNECT EMPOSITION TO JOB-EMPOSITION.	0000	53	5200	329	53	5035	8	329	53	53*	329	5035	53	
FINISH.	0000	Null*	Null*	Null*	Null*	Null*	Null*	Null*	Null*	Null*	Null*	Null*	Null*	

Record is erased, but area and unit currencies are maintained.

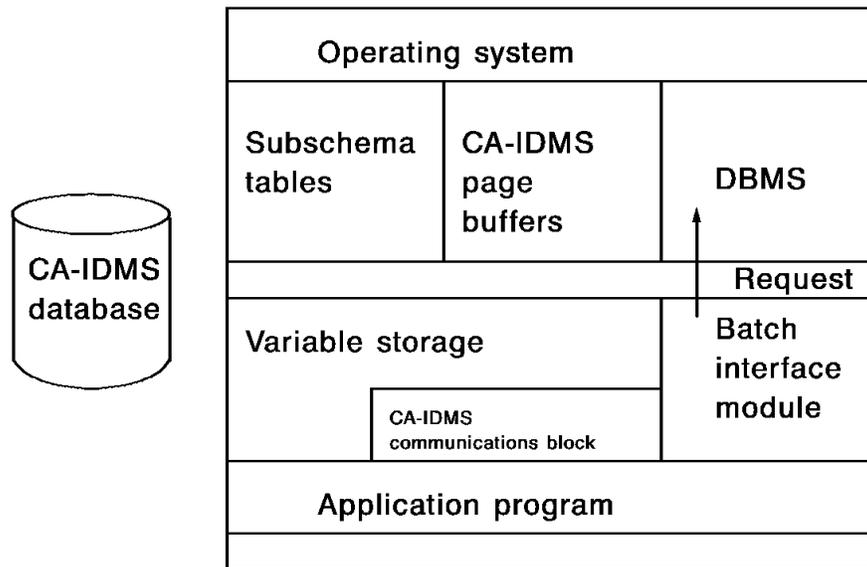
3.6 Database access execution sequence

In navigational DML programming, you access database records one record at a time. At runtime, the following steps are executed during a DML call:

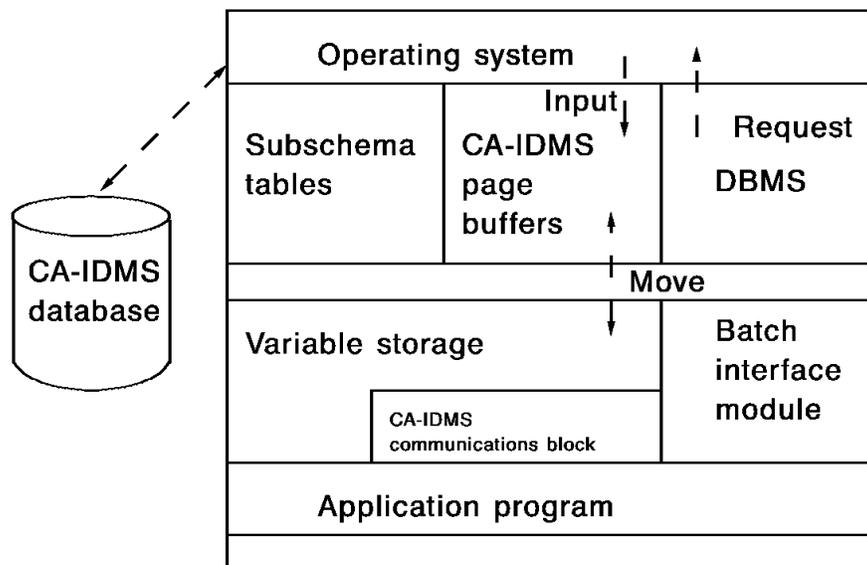
1. The application program calls the batch interface module, which identifies the requested database service, and provides information (that is, record, set, and area names) required to interpret the request.



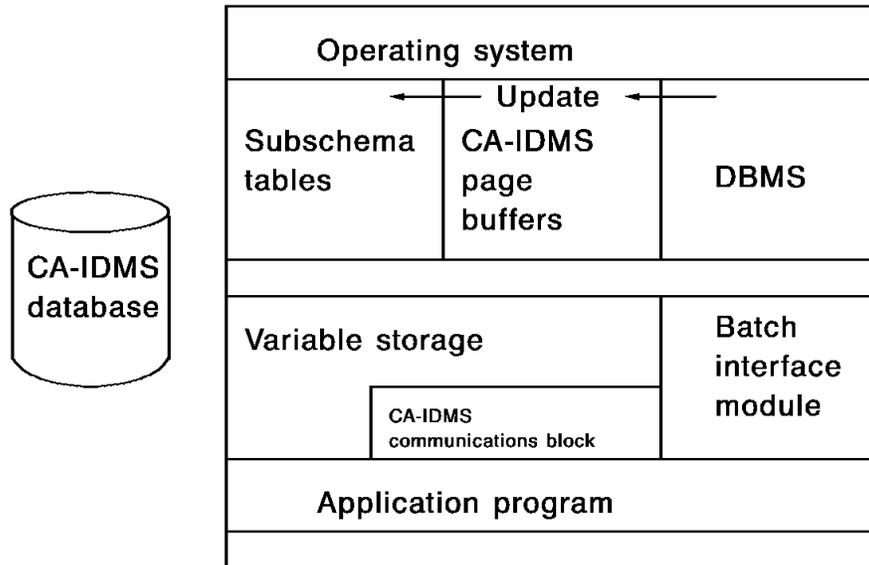
2. The batch interface module transfers control to the DBMS, which then checks the subschema tables for database access information. These tables contain:
 - Record, set, and area definitions
 - Information on currency and access restrictions
 - Database operation statistics



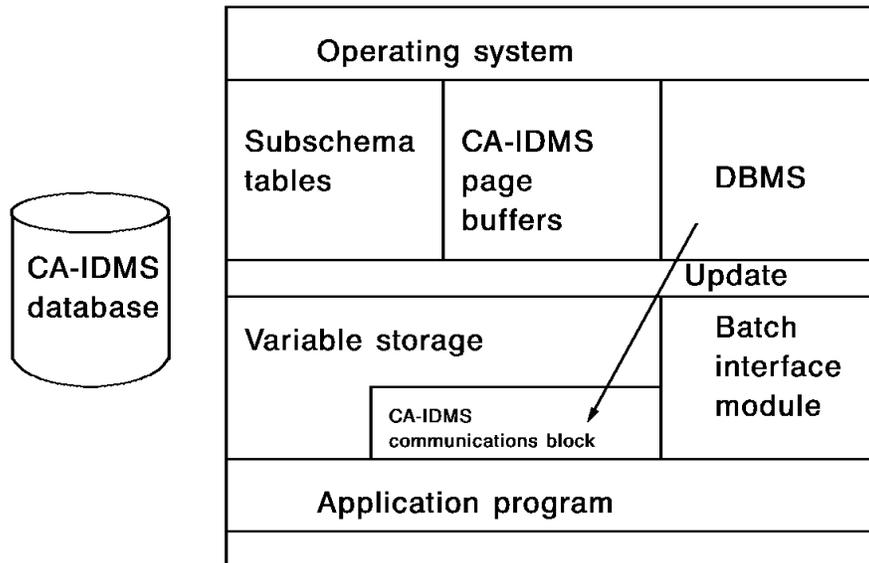
- When the DBMS receives a request for access to a record, it first looks in the page buffers for the requested record occurrence. If the occurrence is present in the buffers, no input operation occurs. If the occurrence is not in the buffers, the DBMS requests the operating system to input the appropriate database page from direct-access storage to the page buffers. If the request to the DBMS specifies movement of the contents of a record to variable storage (for example, an OBTAIN), data is moved from the page buffers to the area in program variable storage associated with the record. The request to the DBMS can also specify the reverse data movement (that is, from variable storage to the page buffers).



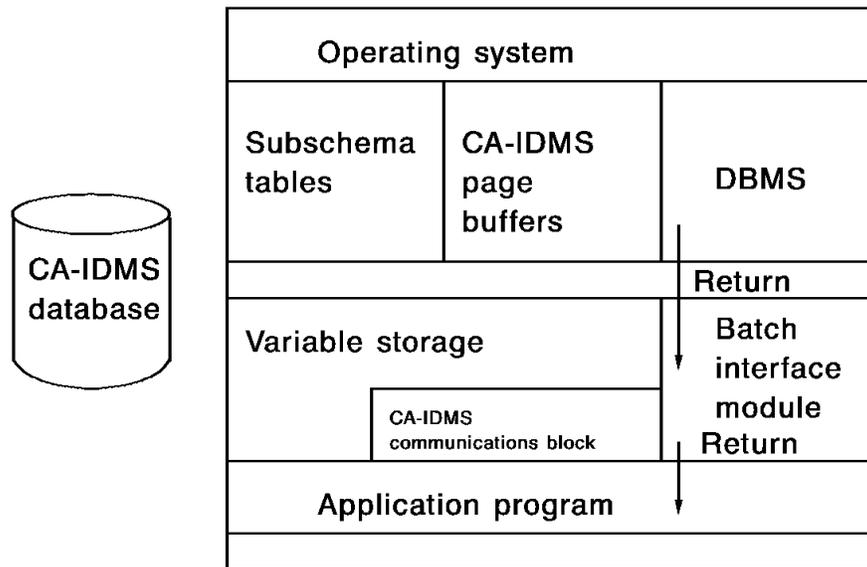
4. After the specified record occurrence is located, the DBMS moves the db-key and other information from the page buffers to the subschema tables. This information includes the currency status of the run unit as well as of the area, sets, and record type of the located record occurrence.



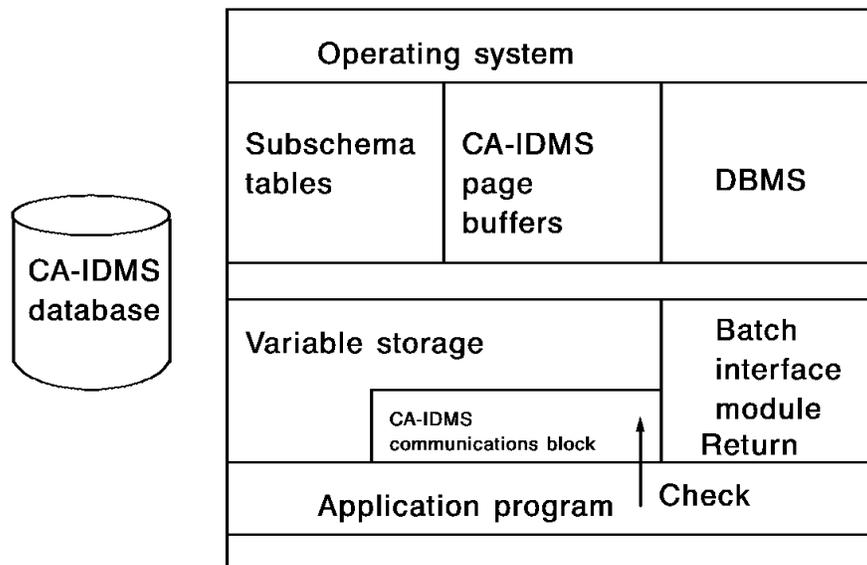
5. The DBMS moves status information regarding the outcome of the DML call to fields within the program's IDMS communications block.



6. The DBMS returns control to the batch interface module, which passes control back to the application program at the statement following the DML statement just executed.



- The program checks the ERROR-STATUS field in the IDMS communications block to determine the outcome of the database request. If the DBMS returns an unexpected value to the ERROR-STATUS field, the program issues a ROLLBACK statement (central version only) to ensure that incomplete updates are not written to the database and terminates processing.



Chapter 4. Navigational DML Programming Techniques

4.1	About this chapter	4-3
4.2	Retrieving records	4-4
4.2.1	Accessing CALC records	4-4
4.2.2	Walking a set	4-5
4.2.3	Accessing a sorted set	4-6
4.2.4	Performing an area sweep	4-10
4.2.5	Accessing owner records	4-12
4.2.6	Reestablishing run-unit currency	4-14
4.2.7	Accessing a record by its db-key	4-15
4.2.8	Accessing indexed records	4-18
4.2.9	Moving contents of a record occurrence	4-21
4.3	Saving db-key, page information and bind addresses	4-24
4.3.1	Saving a db-key	4-24
4.3.2	Saving page information	4-28
4.3.3	Saving a record's BIND address	4-30
4.4	Checking for set membership	4-31
4.4.1	Using the IF EMPTY statement	4-31
4.4.2	Using the IF MEMBER statement	4-32
4.5	Updating the database	4-35
4.5.1	Storing records	4-35
4.5.2	Modifying records	4-37
4.5.3	Erasing records	4-39
4.5.4	Connecting records to a set	4-44
4.5.5	Disconnecting records from a set	4-45
4.6	Locking records	4-48

4.1 About this chapter

This chapter discusses programming techniques used to access the database in navigational DML programs. Functionally similar DML statements are presented together; sample code that demonstrates typical usage of each statement is included. The navigational DML functions are divided into these categories:

- **Retrieving records** — Retrieving information from the database by using navigational DML statements
- **Saving db-key and address information** — Saving db-keys and bind addresses
- **Checking for set membership** — The two forms of the DML IF statement, used to obtain set membership information without performing any I/O
- **Updating the database** — Modifying, storing, erasing, connecting, and disconnecting database records
- **Locking records** — Restricting access to database records

4.2 Retrieving records

In the navigational environment, you can use the following navigational DML statements to retrieve database records:

- **FIND** locates a record occurrence in the database.
- **GET** moves the data associated with a record occurrence from the page buffers to program variable storage.
- **OBTAIN** locates a record occurrence in the database and moves the data associated with that occurrence to program variable storage (the equivalent of a FIND followed by a GET).
- **RETURN** retrieves the db-key and the symbolic key for an indexed record without retrieving the record itself.

The DML retrieval functions listed below are discussed on the following pages:

- Accessing CALC records
- Walking a set
- Accessing a sorted set
- Performing an area sweep
- Accessing owner records
- Reestablishing run-unit currency
- Accessing a record by its db-key
- Accessing indexed records
- Moving contents of a record occurrence

4.2.1 Accessing CALC records

To access a record occurrence based on its CALC-key value, perform the following steps:

1. Move the CALC-key value to the CALC-key field in the database record in variable storage.
2. Issue the FIND/OBTAIN CALC command.
3. Check the ERROR-STATUS field for the value 0326 (DB-REC-NOT-FOUND).
4. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value other than 0326.

Example of retrieving CALC records: The program excerpt below shows retrieval of CALC records.

The MOVE statement initializes the CALC-key field before the database access is performed. If the DBMS returns an ERROR-STATUS of 0326 (condition

DB-REC-NOT-FOUND), the program prints a message and goes on to the next input record.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 SWITCHES.
   05 EOF-SW                PIC X    VALUE 'N'.
   88 END-OF-FILE           VALUE 'Y'.
PROCEDURE DIVISION.
.
.
  READ EMP-FILE-IN
    AT END MOVE 'Y' to EOF-SW.
  IF NOT END-OF-FILE
    PERFORM A400-GET-EMP-REC THRU A400-EXIT
    UNTIL END-OF-FILE.
  FINISH.
  GOBACK.
A400-GET-EMP-REC.
  *** INITIALIZE CALC KEY ***
  MOVE EMP-ID-IN TO EMP-ID-0415.
  *** RETRIEVE RECORD ***
  OBTAIN CALC EMPLOYEE.
  *** CHECK FOR ERROR-STATUS = 0326 ***
  IF DB-REC-NOT-FOUND THEN
    DISPLAY 'EMPLOYEE ID: ' EMP-ID-IN ' NOT FOUND'
  *** CHECK FOR ERROR-STATUS = 0000 ***
  ELSE IF DB-STATUS-OK
    PERFORM B100-WRITE-EMP-REPORT
  ELSE
    PERFORM IDMS-STATUS.
  READ EMP-FILE-IN
    AT END MOVE 'Y' to EOF-SW.
A400-EXIT.
EXIT.

```

4.2.2 Walking a set

To access a record occurrence based on its logical position within a set, perform the following steps:

1. Establish the current of set for the specified set type (for example, by issuing an OBTAIN CALC).
2. Issue the FIND/OBTAIN WITHIN SET command.
3. Check the ERROR-STATUS field for the value 0307 (DB-END-OF-SET).
4. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value other than 0307.

Example of walking a set: The program excerpt below shows the procedure for retrieving all member records in a set.

The program enters the database on the CALC-key field DEPT-ID-0410 and establishes currency on the DEPARTMENT record. It then walks the DEPT-EMPLOYEE set until the DBMS returns an ERROR-STATUS of 0307 (DB-END-OF-SET).

```

WORKING-STORAGE SECTION.
01 SWITCHES.
   05 EOF-SW                PIC X    VALUE 'N'.
   88 END-OF-FILE          VALUE 'Y'.
PROCEDURE DIVISION.
.
   READ DEPT-RECORD-IN
   AT END MOVE 'Y' TO EOF-SW.
   PERFORM A300-GET-DEPT-SET THRU A300-EXIT
   UNTIL EOF-SW = 'Y'.

   FINISH.
   GOBACK.
A300-GET-DEPT-SET.
   MOVE DEPT-ID-IN TO DEPT-ID-0410.
   OBTAIN CALC DEPARTMENT.
   IF DB-REC-NOT-FOUND
      DISPLAY 'DEPT: ' DEPT-ID-IN ' NOT FOUND'
      GO TO A300-GET-NEXT
   ELSE IF DB-STATUS-OK
      NEXT SENTENCE
   ELSE
      PERFORM IDMS-STATUS.
      MOVE DEPT-NAME-0410 TO DEPT-NAME-OUT.
      PERFORM U0900-WRITE-LINE.
A300-SET-WALK.
   *** RETRIEVE NEXT EMPLOYEE IN SET ***
   OBTAIN NEXT EMPLOYEE WITHIN DEPT-EMPLOYEE.
   *** CHECK FOR ERROR-STATUS = 0307 ***
   IF DB-END-OF-SET
      GO TO A300-GET-NEXT
   *** CHECK FOR ERROR-STATUS = 0000 ***
   ELSE IF DB-STATUS-OK
      NEXT SENTENCE
   ELSE
      PERFORM IDMS-STATUS.
      MOVE EMP-NAME-0415 TO EMP-NAME-OUT.
      MOVE EMP-ID-0415 TO EMP-ID-OUT.
      PERFORM U0900-WRITE-LINE.
      GO TO A300-SET-WALK.
A300-GET-NEXT.
   READ DEPT-RECORD-IN
   AT END MOVE 'Y' TO EOF-SW.
A300-EXIT.
EXIT.

```

4.2.3 Accessing a sorted set

To access a record occurrence in a sorted set based on its sort key, use the FIND/OBTAIN WITHIN SET USING SORT KEY statement. The elements that make up a sort key need not be adjacent to one another (that is, they can be contiguous or noncontiguous).

To access a record that has either a **single** or a **contiguous** sort key, perform the following steps:

1. Establish the current of set for the specified set type.
2. Initialize the sort-key field of the database record in program variable storage with the sort-key value; for example:

```
MOVE 77 TO SORT-KEY-1.
```

3. Issue the FIND/OBTAIN WITHIN SET USING SORT KEY statement; for example:
OBTAIN RECORD-B WITHIN A-B USING SORT-KEY-1.
4. Check the ERROR-STATUS field for the value 0326 (DB-REC-NOT-FOUND).
5. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value other than 0326.

Sorted set with a noncontiguous sort key: To access a record that has a **noncontiguous** sort key, perform the following steps:

1. Establish a work field in program variable storage that consists of the record's multiple sort-key elements stored as contiguous data items:

Subschema Record

```
02 RECORD-B.
   05 SORT-KEY-1           PIC 9(2).
   05 NOT-A-KEY-1         PIC X(8).
   05 SORT-KEY-2         PIC 9(5).
   05 NOT-A-KEY-2         PIC XXX.
   05 SORT-KEY-3         PIC X(15).
   05 NOT-A-KEY-3         PIC 9(5)V99.
```

Work Record

```
02 SORT-RECORD-B
   05 S-KEY-1             PIC 9(2).
   05 S-KEY-2             PIC 9(5).
   05 S-KEY-3             PIC X(15).
```

2. Move the sort key values into the work record; for example:

```
MOVE 77           TO S-KEY-1.
MOVE 12345        TO S-KEY-2.
MOVE 'PROGRAMMER' TO S-KEY-3.
```

3. Establish the current of set for the specified set type.
4. Issue the FIND/OBTAIN statement, using the work record:
OBTAIN RECORD-B WITHIN A-B
 USING SORT-RECORD-B.
5. Check the ERROR-STATUS field for the value 0326 (DB-REC-NOT-FOUND).
6. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value other than 0326.

Batch programmers: Sorted sets can be processed more efficiently by sorting the input transactions in the same order as the set before program execution.

Example of retrieval using a sort key: The program excerpt below retrieves an EMPLOYEE record through its sort key.

This example retrieves insurance records for all specified employees. It enters the database through the EMP-NAME-NDX set using the sort key, which is composed of the employee's last name and first name. This example eliminates the need to initialize the sort key elements in the record by using the input file as the sort-control element.

```

DATA DIVISION.
FILE SECTION.
FD SORTED-EMP-FILE-IN.
01 INS-INQ-EMP-REC-IN.
   02 EMP-SORT-NAME.
       04 LAST-IN                PIC X(15).
       04 FIRST-IN               PIC X(10).
WORKING-STORAGE SECTION.
01 SWITCHES.
   05 EOF-SW                     PIC X    VALUE 'N'.
   88 END-OF-FILE                 VALUE 'Y'.
PROCEDURE DIVISION.

   READ INS-INQ-EMP-REC-IN
     AT END MOVE 'Y' TO EOF-SW.
   PERFORM A300-GET-EMP-NDX THRU A300-EXIT
     UNTIL EOF-SW = 'Y'.

   FINISH.
   GOBACK.
A300-GET-EMP-NDX.
   *** RETRIEVE EMPLOYEE USING SORT KEY ***
   OBTAIN EMPLOYEE WITHIN EMP-NAME-NDX
     USING EMP-SORT-NAME.
   *** CHECK FOR ERROR-STATUS = 0326 ***
   IF DB-REC-NOT-FOUND
     THEN DISPLAY
       'EMPLOYEE ' INS-INQ-EMP-REC ' NOT FOUND'
     GO TO A300-GET-NEXT
   *** CHECK FOR ERROR-STATUS = 0000 ***
   ELSE IF DB-STATUS-OK
     NEXT SENTENCE
   ELSE
     PERFORM IDMS-STATUS.
   PERFORM A400-GET-INS-INFO.
A300-GET-NEXT.
   READ INS-INQ-EMP-REC-IN
     AT END MOVE 'Y' TO EOF-SW.
A300-EXIT.
   EXIT.
A400-GET-INS-INFO.
   *** RETRIEVE ALL INSURANCE CLAIM RECORDS THROUGH THE ***
   *** EMP-COVERAGE AND COVERAGE-CLAIMS SETS                ***

```

Sorted set considerations: You should be aware of the following considerations related to the processing of sorted sets:

- The selected record occurrence has a key value equal to the value of the sort-control element. If more than one occurrence contains a sort key equal to the key value in variable storage, the first such record is selected.
- The search for the specified record begins with the owner of the current of set unless the CURRENT option is specified. When CURRENT is specified, the search begins with the currencies already established for the specified set.

Note: If duplicates are allowed, iterative use of CURRENT continually returns the same occurrence; in this case, use OBTAIN NEXT WITHIN SET.

- The search always proceeds in the next direction. The next of set is the member record with the next higher sort-key value (next lower for descending sets) than

the requested value; the prior of set is the member record with the next lower value (higher for descending sets).

Generic key searches: If a member occurrence with the requested sort-key value is not found, the current of set is nullified but the next and prior of set are maintained.

You can use this feature to perform generic key searches. For example, to retrieve all employees whose last names start with the letter **N** or greater, you can establish the appropriate currency by issuing the following statements:

```
MOVE 'N                ' TO EMP-LAST-NAME-0415.  
FIND EMPLOYEE WITHIN EMP-LNAME-NDX USING EMP-LAST-NAME-0415.  
IF ERROR-STATUS = '0326'  
  NEXT SENTENCE  
ELSE  
  PERFORM IDMS-STATUS.
```

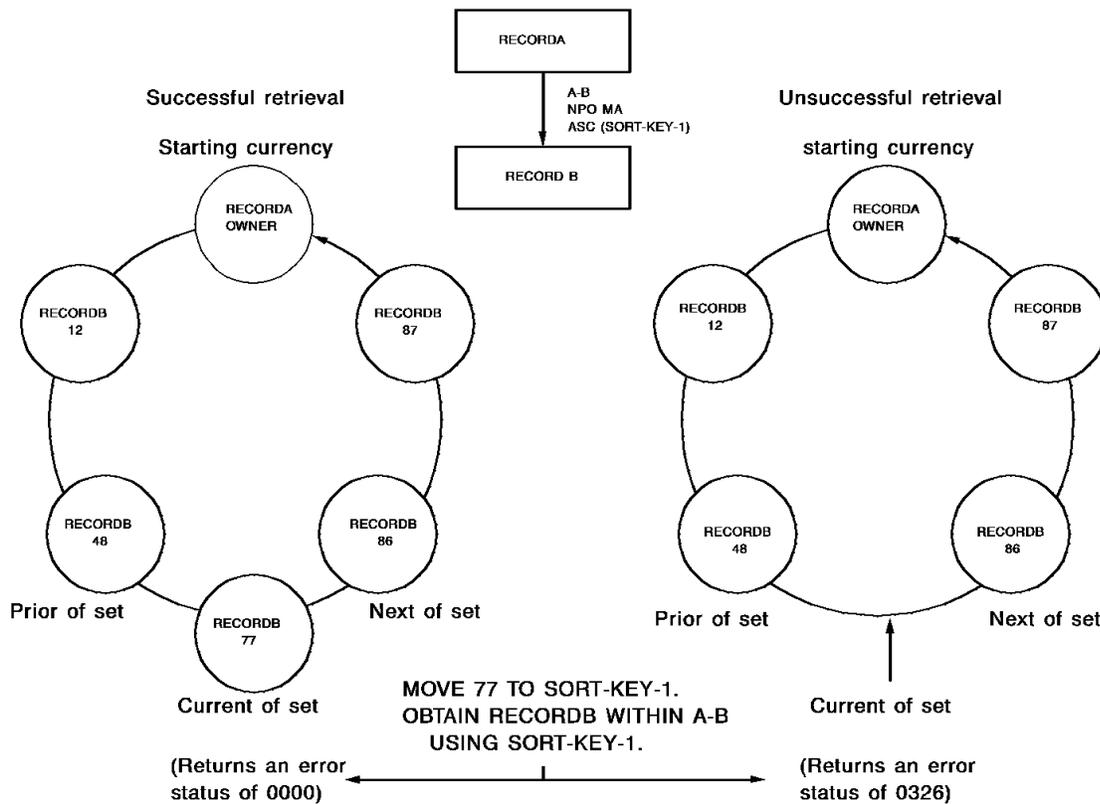
To return the first record containing the partial key value followed by characters other than blanks, you issue this statement:

```
OBTAIN NEXT EMPLOYEE WITHIN EMP-LNAME-NDX USING EMP-LAST-NAME-0415.
```

Continue to issue this OBTAIN until all records within the range you want have been returned.

Example of retrieving occurrences of sorted sets: The figure below shows the currencies maintained by successful and unsuccessful retrieval within a sorted set.

Following successful retrieval within the A-B set, member occurrence 77 is established as current. Following unsuccessful retrieval, a status of 0326 is returned and current of set is nullified, but the next and prior of set are maintained; this enables you to continue accessing that set by using the FIND/OBTAIN WITHIN SET command.



4.2.4 Performing an area sweep

To access a record occurrence based on its logical position within an area, perform the following steps to establish the correct starting position:

1. Issue the FIND/OBTAIN FIRST/LAST/nth WITHIN *area-name* statement.
2. Check the ERROR-STATUS field for the value 0307 (DB-END-OF-SET).
3. Perform the IDMS-STATUS routine if a value other than 0307 is returned.

Accessing subsequent records: To retrieve subsequent record occurrences within an area, perform the following steps:

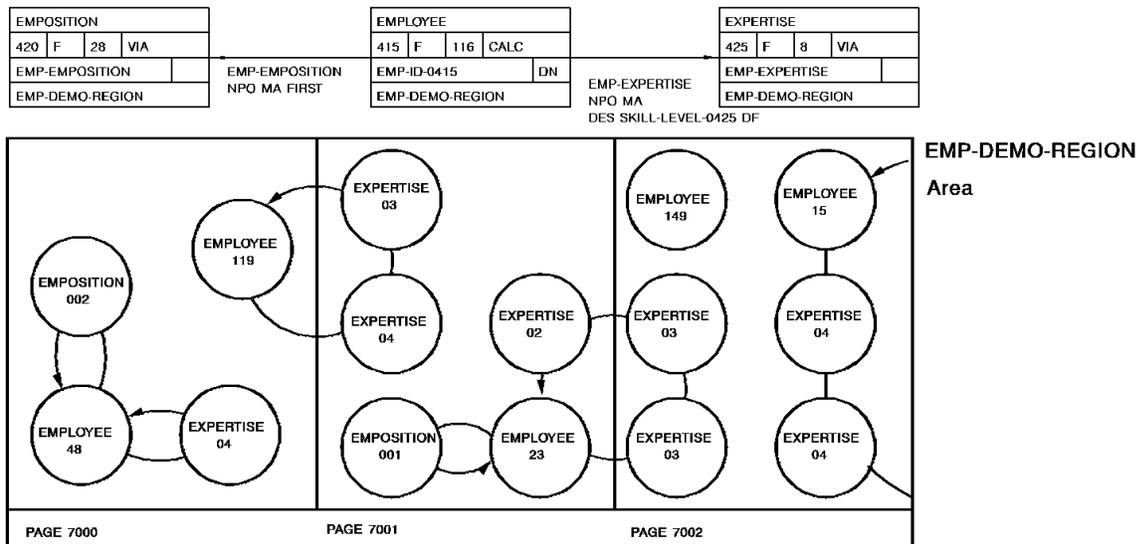
1. Issue the FIND/OBTAIN NEXT/PRIOR WITHIN *area-name* statement.
2. Check the ERROR-STATUS field for the value 0307 (DB-END-OF-SET).
3. Perform the IDMS-STATUS routine if a value other than 0307 is returned.

Relative db-key values: The first record occurrence in an area is the one with the lowest db-key; the last record has the highest db-key. The next record occurrence in an area is the one with the next higher db-key relative to the current record of the named area; the prior record is the one with the next lower db-key relative to the current of area.

Accessing multiple record types: When accessing multiple records types while sweeping an area, you must be sure to reestablish area currency by issuing the FIND CURRENT *record-name* statement each time before reissuing the OBTAIN NEXT WITHIN AREA statement. Failure to reestablish area currency can cause your program to loop or skip records during retrieval.

The figure below shows retrieval of records within an area that contains multiple record types.

In this example, a sweep of the EMP-DEMO-REGION is performed, retrieving sequentially each EMPLOYEE record and all records in the associated EMPLOYEE-EXPERTISE set. The first command retrieves EMPLOYEE 119. Subsequent OBTAIN WITHIN SET statements retrieve the associated EXPERTISE records and establish currency on EXPERTISE 03. The FIND CURRENT statement is used to reestablish the proper position before retrieving EMPLOYEE 48. If FIND CURRENT EMPLOYEE is not specified, an attempt to retrieve the next EMPLOYEE record in the area would return EMPLOYEE 23.



	RUN UNIT	EMPLOYEE	EXPERTISE	EMP-EXPERTISE	EMP-DEMO-REGION
OBTAIN FIRST EMPLOYEE WITHIN EMP-DEMO-REGION.	119	119		119	119
OBTAIN FIRST EXPERTISE WITHIN EMP-EXPERTISE.	04	119	04	04	04
OBTAIN NEXT EXPERTISE WITHIN EMP-EXPERTISE.	03	119	03	03	03
FIND CURRENT EMPLOYEE.	119	119	03	119	119
OBTAIN NEXT EMPLOYEE WITHIN EMP-DEMO-REGION.	48	48	03	48	48

Area sweep of the EMP-DEMO-REGION: The program excerpt below shows a program that retrieves all occurrences of the EMPLOYEE record in the EMP-DEMO-REGION. The program sequentially retrieves each EMPLOYEE record in the EMP-DEMO-REGION area.

```

A000-MAIN-LINE.
.
A400-GET-FIRST.
*** RETRIEVE FIRST EMPLOYEE IN AREA ***
    OBTAIN FIRST EMPLOYEE WITHIN EMP-DEMO-REGION.
*** CHECK FOR ERROR-STATUS = 0307 ***
    IF DB-END-OF-SET
        DISPLAY 'AREA EMPTY'
        FINISH
        GOBACK
    ELSE IF DB-STATUS-OK
        PERFORM A400-AREA-LOOP THRU A400-EXIT
        UNTIL DB-END-OF-SET
    ELSE
        PERFORM IDMS-STATUS.
    FINISH.
    GOBACK.
A400-AREA-LOOP.
    DISPLAY 'EMPLOYEE: ' EMP-ID-0415
           'FIRST NAME: ' EMP-FIRST-NAME-0415
           'LAST NAME: ' EMP-LAST-NAME-0415.
*** RETRIEVE NEXT EMPLOYEE IN AREA ***
    OBTAIN NEXT EMPLOYEE WITHIN EMP-DEMO-REGION.
*** CHECK FOR ERROR-STATUS = 0307 ***
    IF DB-END-OF-SET
        GO TO A400-EXIT
    ELSE IF DB-STATUS-OK
        NEXT SENTENCE
    ELSE
        PERFORM IDMS-STATUS.
A400-EXIT.
EXIT.

```

4.2.5 Accessing owner records

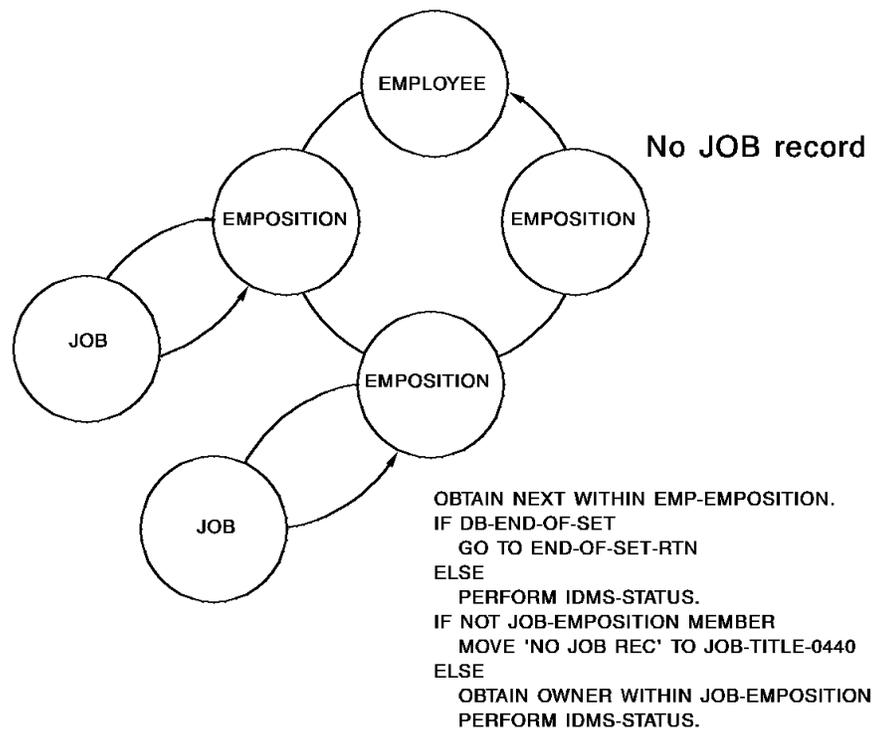
To access the owner record of the current record of set, perform the following steps:

1. Establish the current of set for the specified set type (for example, by issuing an OBTAIN CALC).
2. If the set is defined with either the optional or the manual set membership option, issue the IF MEMBER statement to determine set membership status.
3. Issue the FIND/OBTAIN OWNER command.
4. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

Checking for set membership: If a member record is declared with either the optional or the manual set membership option, you should use the IF statement (explained in 4.4, “Checking for set membership” on page 4-31 later in this chapter) to determine whether the current record of set is presently connected to the specified set. An optional or manual record is *not* established as current of set if it is not presently connected to an occurrence of the specified set. For example, a manual record may not have been connected to the requested set or an optional record may have been disconnected.

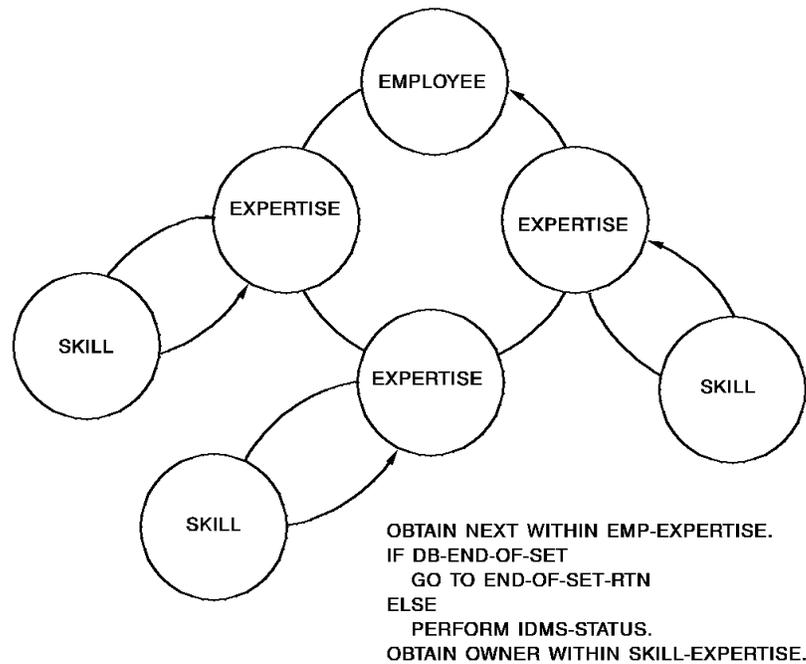
How FIND/OBTAIN OWNER works: FIND/OBTAIN OWNER uses the *current of set*. If the current of record is eligible for set membership, but is not connected to an occurrence of the requested set, the record occurrence retrieved is the owner of the current record of set, not the owner of the current of record.

OWNER retrieval in optional or manual sets: The program excerpt and the figure below illustrate OWNER retrieval for sets with either the optional or the manual membership option. Records defined to sets with either the optional or the manual option may not be connected to a set occurrence; you can use the DML IF statement to test for set membership.



Owner retrieval for mandatory automatic sets: A member record declared as a *mandatory automatic* member of a set (see 3.3.3, “Set membership options” on page 3-10) must be connected to an owner record. The program excerpt and figure illustrate OWNER retrieval for mandatory automatic sets.

Records defined to mandatory automatic sets are connected to a set occurrence when they are stored; they cannot be disconnected. Therefore, you need not test for the existence of owner record occurrences.



4.2.6 Reestablishing run-unit currency

Certain navigational DML statements operate on previously established currencies. You may need to reestablish a previously obtained record as current of run unit in order to execute one of these statements.

►► For information on the currencies required by each navigational DML statement, see Chapter 3, “Introduction to Database Access with Navigational DML” on page 3-1.

How you reestablish currency: To reestablish the current record of record type, set, or area as the current record of run unit, perform the following steps:

1. Establish currency for the named record, set, or area.
2. Perform processing that alters run-unit currency.
3. Issue the FIND/OBTAIN CURRENT statement.
4. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

Using FIND/OBTAIN CURRENT: The FIND/OBTAIN CURRENT statement is an efficient means of establishing the appropriate record as current of run unit, set, or area before executing a DML statement that utilizes run-unit, set, or area currency (for example, ACCEPT, IF, GET, MODIFY, ERASE).

The program excerpt below shows two typical uses of the FIND/OBTAIN CURRENT statement.

This application performs an area sweep of the EMP-DEMO-REGION, looking for former EMPLOYEES to be deleted from the database. Former employees have either no EMPOSITION records or no JOB records.

```

A400-ERASE-NOJOBS.
  OBTAIN FIRST EMPLOYEE WITHIN EMP-DEMO-REGION.
*** CHECK FOR ERROR-STATUS = 0307 ***
  IF DB-END-OF-SET
    DISPLAY 'AREA EMPTY'
*** CHECK FOR ERROR-STATUS = 0000 ***
  ELSE IF DB-STATUS-OK
    PERFORM A400-AREA-LOOP THRU A400-EXIT
    UNTIL DB-END-OF-SET
  ELSE
    PERFORM IDMS-STATUS.
  FINISH.
  GOBACK.
A400-AREA-LOOP.
*** ERASE IF EMPLOYEE HAS NO EMPOSITION RECORDS ***
  IF EMP-EMPOSITION IS EMPTY
    ERASE EMPLOYEE PERMANENT
    PERFORM IDMS-STATUS
    GO TO A400-FIND-NEXT.
  FIND FIRST EMPOSITION WITHIN EMP-EMPOSITION.
  PERFORM IDMS-STATUS.
*** ALSO ERASE IF EMPLOYEE HAS NO JOB RECORDS ***
  IF NOT JOB-EMPOSITION MEMBER
*** USE #1 REESTABLISH RUN UNIT CURRENCY FOR ERASE ***
    FIND CURRENT EMPLOYEE
    PERFORM IDMS-STATUS
    ERASE EMPLOYEE PERMANENT
    PERFORM IDMS-STATUS
  ELSE
*** USE #2 REESTABLISH RUN UNIT CURRENCY FOR OBTAIN WITHIN AREA ***
    FIND CURRENT EMPLOYEE
    PERFORM IDMS-STATUS.
A400-FIND-NEXT.
  OBTAIN NEXT EMPLOYEE WITHIN EMP-DEMO-REGION.
*** CHECK FOR ERROR-STATUS = 0307 ***
  IF DB-END-OF-SET
    GO TO A400-EXIT
  ELSE IF DB-STATUS-OK
    NEXT SENTENCE
  ELSE
    PERFORM IDMS-STATUS.
A400-EXIT.
EXIT.

```

4.2.7 Accessing a record by its db-key

The DBMS assigns a db-key to each record occurrence in the database. This key identifies the database page and line number where the record is located. The db-key can be qualified by record type or page information to ensure that it identifies a unique record occurrence. While always allowed, qualification is necessary only under the following circumstances:

- the subschema includes areas with different page information values

- the page information associated with the current of run unit is different than that of the record to be retrieved

►► For more information about qualifying db-keys, refer to 2.3, “Db-keys and page information” on page 2-11.

Steps to access a record by its db-key: To access a record directly by using its db-key, perform the following steps:

1. Save the db-key of the record to be retrieved in a field defined as a binary fullword (COBOL PIC S9(8) COMP SYNC). Optionally save its record type or page information to use to qualify the db-key. For more information, refer to 4.3.1, “Saving a db-key” on page 4-24 later in this chapter.
2. Perform processing as required.
3. Issue the FIND/OBTAIN DB-KEY command using the saved db-key and qualifying information.
4. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

When to use access by db-key: Using a record's db-key provides for the most efficient form of database retrieval. For example, if you know that your program will need to use a record more than once, it is best to save the record's db-key and reaccess the record by using FIND/OBTAIN DB-KEY. Any subschema record can be accessed by its db-key, regardless of location mode. Currency is not used to determine the target record of the FIND/OBTAIN DB-KEY statement; the record is identified by its db-key and, optionally, by its record type or page information.

Native VSAM users: The FIND/OBTAIN DB-KEY statement cannot be used to access records in a native VSAM key-sequenced data set (KSDS).

Example of record access by db-key and page-info: The program excerpt below shows using a db-key and page-info to reestablish currency.

Note: This application walks the DEPT-EMPLOYEE set, printing a report of all employees and their managers. After accessing the manager's EMPLOYEE record, the FIND DB-KEY statement is used to reestablish the correct EMPLOYEE record as current of the DEPT-EMPLOYEE set.

```

WORKING-STORAGE SECTION.
01 SAVED-DBKEYS.
   05 SAVE-EMP-DBKEY          PIC S9(8) COMP SYNC.
PROCEDURE DIVISION.

A200-GET-EMP-MANAGER.
*** RETRIEVE EMPLOYEES SEQUENTIALLY WITHIN SET ***
   OBTAIN NEXT WITHIN DEPT-EMPLOYEE.
*** CHECK FOR ERROR-STATUS = 0307 ***
   IF DB-END-OF-SET
     GO TO A200-EXIT
*** CHECK FOR ERROR-STATUS = 0000 ***
   ELSE IF DB-STATUS-OK
     NEXT SENTENCE
   ELSE
     PERFORM IDMS-STATUS.
*** SAVE EMPLOYEES' DB-KEY ***
   MOVE DBKEY TO SAVE-EMP-DBKEY.
   PERFORM IDMS-STATUS.
   MOVE EMP-ID-0415 TO EMP-ID-OUT.
   MOVE EMP-FIRST-NAME-0415 TO EMP-FIRST-OUT.
   MOVE EMP-LAST-NAME-0415 TO EMP-LAST-OUT.
   IF REPORTS-TO IS EMPTY
     DISPLAY 'EMPLOYEE ' EMP-ID-0415 'HAS NO MANAGER'
     GO TO A200-EXIT.
   FIND FIRST WITHIN REPORTS-TO.
   PERFORM IDMS-STATUS.
*** ACCESS MANAGER'S EMPLOYEE RECORD ***
   OBTAIN OWNER WITHIN MANAGES.
   PERFORM IDMS-STATUS.
   MOVE EMP-FIRST-NAME-0415 TO MANAGER-FIRST-OUT.
   MOVE EMP-LAST-NAME-0415 TO MANAGER-LAST-OUT.
*** REESTABLISH EMPLOYEE CURRENCY TO      ***
*** CONTINUE WALKING THE DEPT-EMPLOYEE SET ***
   FIND EMPLOYEE DB-KEY IS SAVE-EMP-DBKEY.
   PERFORM IDMS-STATUS.
A200-EXIT.
EXIT.

```

Example of record access by db-key and page-info: The program excerpt below shows using a db-key and page-info to reestablish currency.

Note: Use this example only when the subschema includes areas that have mixed page groups.

This application walks the DEPT-EMPLOYEE set, printing a report of all employees and their managers. After accessing the manager's EMPLOYEE record, the FIND DB-KEY statement is used to reestablish the correct EMPLOYEE record as current of the DEPT-EMPLOYEE set.

```

WORKING-STORAGE SECTION.
01  SAVED-DBKEY-PAGEINFO.
    05  SAVE-EMP-DBKEY          PIC S9(8) COMP SYNC.
    05  SAVE-EMP-PAGEINFO      PIC S9(8) COMP SYNC.
PROCEDURE DIVISION.

A200-GET-EMP-MANAGER.
*** RETRIEVE EMPLOYEES SEQUENTIALLY WITHIN SET ***
    OBTAIN NEXT WITHIN DEPT-EMPLOYEE.
*** CHECK FOR ERROR-STATUS = 0307 ***
    IF DB-END-OF-SET
        GO TO A200-EXIT
*** CHECK FOR ERROR-STATUS = 0000 ***
    ELSE IF DB-STATUS-OK
        NEXT SENTENCE
    ELSE
        PERFORM IDMS-STATUS.
*** SAVE EMPLOYEES' DB-KEY and PAGE-INFO ***
    MOVE DBKEY TO SAVE-EMP-DBKEY.
    MOVE PAGE-INFO TO SAVE-EMP-PAGEINFO.
    PERFORM IDMS-STATUS.
    MOVE EMP-ID-0415 TO EMP-ID-OUT.
    MOVE EMP-FIRST-NAME-0415 TO EMP-FIRST-OUT.
    MOVE EMP-LAST-NAME-0415 TO EMP-LAST-OUT.
    IF REPORTS-TO IS EMPTY
        DISPLAY 'EMPLOYEE ' EMP-ID-0415 'HAS NO MANAGER'
        GO TO A200-EXIT.
    FIND FIRST WITHIN REPORTS-TO.
    PERFORM IDMS-STATUS.
*** ACCESS MANAGER'S EMPLOYEE RECORD ***
    OBTAIN OWNER WITHIN MANAGES.
    PERFORM IDMS-STATUS.
    MOVE EMP-FIRST-NAME-0415 TO MANAGER-FIRST-OUT.
    MOVE EMP-LAST-NAME-0415 TO MANAGER-LAST-OUT.
*** REESTABLISH EMPLOYEE CURRENCY TO ***
*** CONTINUE WALKING THE DEPT-EMPLOYEE SET ***
    FIND DB-KEY IS SAVE-EMP-DBKEY PAGE-INFO SAVE-EMP-PAGEINFO.
    PERFORM IDMS-STATUS.
A200-EXIT.
EXIT.

```

4.2.8 Accessing indexed records

Indexes provide an efficient means of accessing member record occurrences. You can retrieve member records in indexed sets as if they were member records in nonindexed sets.

The table below lists the retrieval statements that you can use with indexed records.

Retrieval statement	Restrictions
FIND/OBTAIN CURRENT	WITHIN SET option only
FIND/OBTAIN RECORD	WITHIN SET option only
FIND/OBTAIN USING SORT KEY	Sorted indexed sets only
FIND/OBTAIN OWNER	OBTAIN not allowed for system-owned indexes
RETURN	Db-key and symbolic key only

Example of accessing an indexed record: The program excerpt below shows retrieval of all records in the EMP-NAME-NDX (a system-owned indexed set).

The EMP-NAME-NDX set is sorted in ascending order on EMP-LAST-NAME and EMP-FIRST-NAME; this program produces an alphabetical list of all employees.

```

PROCEDURE DIVISION.
A000-MAIN-LINE.
.
.
.
        PERFORM A000-GET-NDX-SET THRU A000-EXIT
                UNTIL DB-END-OF-SET.
        PERFORM END-PROCESSING.
        GOBACK.

A000-GET-NDX-SET.
*** SEQUENTIALLY RETRIEVE EMPLOYEES INDEXED BY LAST NAME ***
        OBTAIN NEXT EMPLOYEE WITHIN EMP-NAME-NDX.
*** CHECK FOR ERROR-STATUS = 0307 ***
        IF DB-END-OF-SET
                GO TO A000-EXIT
*** CHECK FOR ERROR-STATUS = 0000 ***
        ELSE IF DB-STATUS-OK
                NEXT SENTENCE
        ELSE
                PERFORM IDMS-STATUS.
        DISPLAY EMP-ID-0415
                EMP-LAST-NAME-0415
                EMP-FIRST-NAME-0415.
A000-EXIT.
EXIT.

```

Retrieving the key without the record: To retrieve the db-key and symbolic key of an indexed record without retrieving the record itself, perform the following steps:

1. Initialize variable storage fields, as required.
2. Issue the RETURN statement.
3. If you are issuing the RETURN statement iteratively, check for an ERROR-STATUS of 1707; if you are doing a generic-key search, check for an ERROR-STATUS of 1726.
4. Perform the IDMS-STATUS routine if 1726, 1707, or 0000 is not returned.

Using the RETURN statement: The RETURN statement establishes currency in the indexed set and moves the record's symbolic key into the data fields within the record in program variable storage. Alternatively, you can move the record's symbolic key into some other specified variable-storage location.

Example of using RETURN: The program excerpt below uses the RETURN statement to establish indexed set currency.

This program establishes currency in the EMP-NAME-NDX set by using the RETURN statement to perform a generic-key search. It checks for the ERROR-STATUS 1726 (record not found), and retrieves all employees whose last name begins with the letter N or greater.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 INDEX-ITEMS.
   03 DB-KEY-V          PIC S9(8) COMP SYNC.
   03 INDEX-START-POINT PIC X(15)  VALUE 'N          '.
PROCEDURE DIVISION.
A000-MAIN-LINE.
.
.
.
   MOVE INDEX-START-POINT TO INDEX-KEY-VALUE.
   RETURN DB-KEY-V FROM EMP-NAME-NDX
       USING INDEX-START-POINT.
   IF ERROR-STATUS = '1726' THEN
       NEXT SENTENCE
   ELSE IF DB-STATUS-OK
       NEXT SENTENCE
   ELSE
       PERFORM IDMS-STATUS.
       PERFORM A000-GET-NDX-SET THRU A000-EXIT
           UNTIL DB-END-OF-SET.
       FINISH.
       GOBACK.
A000-GET-NDX-SET.
   OBTAIN NEXT EMPLOYEE WITHIN EMP-NAME-NDX.
*** CHECK FOR ERROR-STATUS = 0307 ***
   IF DB-END-OF-SET
       GO TO A000-EXIT
*** CHECK FOR ERROR-STATUS = 0000 ***
   ELSE IF DB-STATUS-OK
       NEXT SENTENCE
   ELSE
       PERFORM IDMS-STATUS.
       DISPLAY EMP-ID-0415
           EMP-LAST-NAME-0415
           EMP-FIRST-NAME-0415.
A000-EXIT.
EXIT.

```

4.2.9 Moving contents of a record occurrence

To move the contents of a specified record occurrence from the page buffer to program variable storage, perform the following steps:

1. Ensure that the following currencies are established:
 - The specified record occurrence must have been established as current of record type by a previous FIND statement.
 - The record must be established as current of run unit.
2. Issue the GET statement.
3. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

Variable-length records: In most cases, using OBTAIN to retrieve a record's data or using FIND (with no GET) to establish position is enough to satisfy your processing needs. However, using FIND followed by GET can save I/O in the case of variable-length records. FINDing a variable-length record fills the page buffers with only the root segment, thus saving the additional I/O needed to access all of its fragments. If necessary, you can issue the GET statement later in the program to transfer the data to program variable storage. You should only use the FIND/GET combination to retrieve a record that you may not need.

Example of moving record contents: The program excerpt below uses the GET statement to move the contents of the EMPOSITION record from the page buffers to program variable storage. This application uses the FIND/GET combination to access only those EMPOSITION records owned by the specified JOB record.

```
WORKING-STORAGE SECTION.  
01 JOB-DBKEY          PIC S9(8)  COMP.  
01 MATCH-DBKEY       PIC S9(8)  COMP.  
.  
.  
PROCEDURE DIVISION.  
.  
.  
    PERFORM A100-GET-EMP-JOB THRU A100-EXIT  
        UNTIL END-OF-FILE.  
.  
.  
A100-GET-EMP-JOB.  
    MOVE GETEMP-ID-IN TO EMP-ID-0415.  
    OBTAIN CALC EMPLOYEE.  
*** CHECK FOR ERROR-STATUS = 0326 ***  
    IF DB-REC-NOT-FOUND  
        DISPLAY 'EMP NOT FOUND: ' GETEMP-ID-IN  
        GO TO A100-GET-NEXT  
*** CHECK FOR ERROR-STATUS = 0000 ***  
    ELSE IF DB-STATUS-OK  
        NEXT SENTENCE  
    ELSE  
        PERFORM IDMS-STATUS.  
        MOVE GETJOB-ID-IN TO JOB-ID-0440.  
        OBTAIN CALC JOB.  
*** CHECK FOR ERROR-STATUS = 0326 ***  
    IF DB-REC-NOT-FOUND  
        DISPLAY 'JOB NOT FOUND: ' GETJOB-ID-IN  
        GO TO A100-GET-NEXT  
*** CHECK FOR ERROR-STATUS = 0000 ***  
    ELSE IF DB-STATUS-OK  
        NEXT SENTENCE  
    ELSE  
        PERFORM IDMS-STATUS.  
*** SAVE JOB DB-KEY ***  
    MOVE DBKEY TO JOB-DBKEY.  
    IF EMP-EMPOSITION IS EMPTY  
        DISPLAY 'EMP-EMPOSITION IS EMPTY FOR: ' GETEMP-ID-IN  
        GO TO A100-GET-NEXT  
    ELSE  
        PERFORM A200-LOOP THRU A200-EXIT.  
A100-GET-NEXT.  
    READ GET-FILE-IN AT END MOVE 'Y' TO EOF-SW.  
A100-EXIT.  
    EXIT.  
A200-LOOP.
```

```
FIND NEXT WITHIN EMP-EMPOSITION.
*** CHECK FOR ERROR-STATUS = 0307 ***
  IF DB-END-OF-SET
    GO TO A200-EXIT
*** CHECK FOR ERROR-STATUS = 0000 ***
  ELSE IF DB-STATUS-OK
    NEXT SENTENCE
  ELSE
    PERFORM IDMS-STATUS.
*** ACCESS DB-KEY OF OWNER IN JOB-EMPOSITION SET ***
  ACCEPT MATCH-DBKEY FROM JOB-EMPOSITION
    OWNER CURRENCY.
  IF DB-STATUS-OK
    NEXT SENTENCE
  ELSE
    PERFORM IDMS-STATUS.
*** IF DB-KEYS ARE NOT EQUAL, LOOP AND TRY AGAIN ***
  IF JOB-DBKEY NOT = MATCH-DBKEY
    THEN
      GO TO A200-LOOP
    ELSE
      NEXT SENTENCE.
*** IF DB-KEYS ARE EQUAL, ACCESS THE EMPOSITION DATA ***
  GET EMPOSITION.
  IF NOT DB-STATUS-OK
    PERFORM IDMS-STATUS
  ELSE NEXT SENTENCE.
  PERFORM A300-PRINT-DATA.
A200-EXIT.
EXIT.
.
.
.
```

4.3 Saving db-key, page information and bind addresses

Retrieving a record by using its db-key is the most efficient form of retrieval. If you know that you will use a record later in your program, you should save its db-key in order to reaccess the record by using db-key retrieval. In certain circumstances a db-key used to access a record may require qualification by record type or page information. You can save page information when saving a db-key or by issuing a DML request.

►► For information about direct access to a record, see 4.2.7, “Accessing a record by its db-key” on page 4-15 earlier in this chapter.

►► For more information about db-key qualification, see 2.3, “Db-keys and page information” on page 2-11.

ACCEPT statements (also called save statements) transfer db-keys, page information and storage-addresses from the DBMS to program variable storage. These statements are an efficient means of obtaining information at runtime since they cause no database I/O.

Saving a db-key, page information and a bind address are explained below.

4.3.1 Saving a db-key

You can retrieve a db-key using one of these methods:

- **Accepting the db-key of a current record.** You can retrieve the db-key of the record that is current of run unit, record type, set, or area using the ACCEPT DB-KEY FROM CURRENCY statement.

Note: You can also retrieve the db-key of the record that is current of run unit from the DBKEY field of the IDMS communications block. You can also retrieve the page information of the record that is current of run unit from the PAGE-INFO field of the IDMS communications block.

- **Accepting a db-key relative to the current record.** You can use an ACCEPT DB-KEY RELATIVE TO CURRENCY statement to retrieve the db-key of the NEXT, PRIOR, or OWNER record relative to the current record of set.

Steps in saving a db-key: To save a db-key, perform the following steps:

1. Establish the appropriate currency for the required record.
2. Perform one of the following steps:
 - **If the required record is current of run unit,** move the DBKEY field in the IDMS communications block to a variable storage field defined as a binary fullword (COBOL PIC S9(8) COMP SYNC).
 - **If the required record is relative to the current of run unit,** issue either the ACCEPT DBKEY FROM CURRENCY or the ACCEPT DBKEY RELATIVE TO CURRENCY statement, storing the saved db-key in a

variable storage field defined as a binary fullword (COBOL PIC S9(8) COMP SYNC).

3. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

CAUTION:

You should not save db-keys or page information outside of the program because these values can change if the database is unloaded and reloaded, if record occurrences are erased or if an area is assigned to a different group.

Example of using db-keys: The program excerpt below shows a program that compares db-keys. The first db-key is acquired from the IDMS communications block, the second by using an ACCEPT DB-KEY statement.

This application compares the db-key of each JOB record with JOB owner db-keys in EMPOSITION records in the JOB-EMPOSITION set. When the db-keys match, the program accesses the EMPOSITION information by issuing a GET statement.

```
WORKING-STORAGE SECTION.  
01 JOB-DBKEY          PIC S9(8)  COMP.  
01 MATCH-DBKEY       PIC S9(8)  COMP.  
.  
.  
PROCEDURE DIVISION.  
.  
.  
    PERFORM A100-GET-EMP-JOB THRU A100-EXIT  
        UNTIL END-OF-FILE.  
.  
.  
A100-GET-EMP-JOB.  
    MOVE GETEMP-ID-IN TO EMP-ID-0415.  
    OBTAIN CALC EMPLOYEE.  
*** CHECK FOR ERROR-STATUS = 0326 ***  
    IF DB-REC-NOT-FOUND  
        DISPLAY 'EMP NOT FOUND: ' GETEMP-ID-IN  
        GO TO A100-GET-NEXT  
*** CHECK FOR ERROR-STATUS = 0000 ***  
    ELSE IF DB-STATUS-OK  
        NEXT SENTENCE  
    ELSE  
        PERFORM IDMS-STATUS.  
        MOVE GETJOB-ID-IN TO JOB-ID-0440.  
        OBTAIN CALC JOB.  
*** CHECK FOR ERROR-STATUS = 0326 ***  
    IF DB-REC-NOT-FOUND  
        DISPLAY 'JOB NOT FOUND: ' GETJOB-ID-IN  
        GO TO A100-GET-NEXT  
*** CHECK FOR ERROR-STATUS = 0000 ***  
    ELSE IF DB-STATUS-OK  
        NEXT SENTENCE  
    ELSE  
        PERFORM IDMS-STATUS.  
*** SAVE JOB DB-KEY ***  
    MOVE DBKEY TO JOB-DBKEY.  
    IF EMP-EMPOSITION IS EMPTY  
        DISPLAY 'EMP-EMPOSITION IS EMPTY FOR: ' GETEMP-ID-IN  
        GO TO A100-GET-NEXT  
    ELSE  
        PERFORM A200-LOOP THRU A200-EXIT.
```

```

A100-GET-NEXT.
  READ GET-FILE-IN AT END MOVE 'Y' TO EOF-SW.
A100-EXIT.
  EXIT.
A200-LOOP.
  FIND NEXT WITHIN EMP-EMPOSITION.
*** CHECK FOR ERROR-STATUS = 0307 ***
  IF DB-END-OF-SET
    GO TO A200-EXIT
*** CHECK FOR ERROR-STATUS = 0000 ***
  ELSE IF DB-STATUS-OK
    NEXT SENTENCE
  ELSE
    PERFORM IDMS-STATUS.
*** ACCESS DB-KEY OF OWNER IN JOB-EMPOSITION SET ***
  ACCEPT MATCH-DBKEY FROM JOB-EMPOSITION
    OWNER CURRENCY.
  IF DB-STATUS-OK
    NEXT SENTENCE
  ELSE
    PERFORM IDMS-STATUS.
*** IF DB-KEYS ARE NOT EQUAL, LOOP AND TRY AGAIN ***
  IF JOB-DBKEY NOT = MATCH-DBKEY
    THEN
      GO TO A200-LOOP
    ELSE
      NEXT SENTENCE.
*** IF DB-KEYS ARE EQUAL, ACCESS THE EMPOSITION DATA ***
  GET EMPOSITION.
  IF NOT DB-STATUS-OK
    PERFORM IDMS-STATUS
  ELSE NEXT SENTENCE.
  PERFORM A300-PRINT-DATA.
A200-EXIT.
  EXIT.
.
.
.

```

Inferring information: For indexed sets and chained sets with prior pointers, the ACCEPT DB-KEY RELATIVE TO CURRENCY statement can also be used to infer information, as shown in the program excerpt below.

This application erases all DEPARTMENT records that contain less than two EMPLOYEE records. The first ACCEPT statement tests for zero EMPLOYEE records; the second ACCEPT statement tests for one.

```

WORKING-STORAGE SECTION.
01 SAVED-DBKEYS.
   05 NEXT-DEPT-EMP-DBKEY    PIC S9(8) COMP SYNC.
   05 PRIOR-DEPT-EMP-DBKEY   PIC S9(8) COMP SYNC.
PROCEDURE DIVISION.
A100-LEAN-AND-FAST.
   OBTAIN FIRST DEPARTMENT WITHIN ORG-DEMO-REGION.
*** CHECK FOR ERROR-STATUS = 0307 ***
   IF DB-END-OF-SET THEN
     GO TO EMPTY-AREA
   ELSE
     PERFORM IDMS-STATUS.
     PERFORM A200-ACCEPT-AND-TEST THRU A200-EXIT
       UNTIL DB-END-OF-SET.
   FINISH.
   GOBACK.
A200-ACCEPT-AND-TEST.
*** RETRIEVE NEXT DB-KEY ***
   ACCEPT NEXT-DEPT-EMP-DBKEY FROM
     DEPT-EMPLOYEE NEXT CURRENCY.
   PERFORM IDMS-STATUS.
*** CHECK FOR EMPTY SET ***
*** IF DB-KEYS ARE THE SAME, THE SET IS EMPTY ***
   IF NEXT-DEPT-EMP-DBKEY = DBKEY THEN
     ERASE DEPARTMENT PERMANENT
     PERFORM IDMS-STATUS
     GO TO A200-GET-NEXT.
*** CHECK FOR ONE-MEMBER SET ***
   ACCEPT PRIOR-DEPT-EMP-DBKEY FROM
     DEPT-EMPLOYEE PRIOR CURRENCY.
   PERFORM IDMS-STATUS.
*** IF DB-KEYS ARE THE SAME, THE SET HAS ONE MEMBER ***
   IF NEXT-DEPT-EMP-DBKEY =
     PRIOR-DEPT-EMP-DBKEY THEN
     ERASE DEPARTMENT PERMANENT
     PERFORM IDMS-STATUS
     GO TO A200-GET-NEXT
   ELSE
     GO TO A200-GET-NEXT.
A200-GET-NEXT.
   OBTAIN NEXT DEPARTMENT WITHIN ORG-DEMO-REGION.
*** CHECK FOR ERROR-STATUS = 0307 ***
   IF DB-END-OF-SET THEN
     GO TO A200-EXIT
   ELSE
     PERFORM IDMS-STATUS.
A200-EXIT.
EXIT.

```

4.3.2 Saving page information

You can retrieve page information using one of these methods:

- Moving the page information of the record that is current of run unit from the PAGE-INFO field of the IDMS communications block.
- Accepting page information for a record type by using an ACCEPT PAGE-INFO statement.

Steps in saving page information: To save page information, perform the following steps:

1. If the required record is current of run unit, move the PAGE-INFO field of the IDMS communications block to a variable storage field.
2. If you know the record type for which page information is desired:
 - Issue the ACCEPT PAGE-INFO statement, storing the output in a variable storage field.
 - Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

The variable storage field used to hold page information can either be defined as a binary fullword field (COBOL PIC S9(8) COMP SYNC) or as a group item consisting of two contiguous binary halfwords. In COBOL this might look as:

```
01 <group-field-name>.  
  02 <page-group-field-name> PIC S9(4) COMP SYNC.  
  02 <dbkey-radix-field-name> PIC S9(4) COMP SYNC.
```

The latter definition enables the components of the page information to be accessed independently.

CAUTION:

You should not save page information outside of the program because the value can change if the database is unloaded and reloaded or if an area is assigned to a different page group.

Example of using page information: The following example retrieves the page information for the DEPARTMENT record and uses the db-key radix to separate a db-key's page and line numbers.

```
WORKING-STORAGE SECTION.  
01 W-DBKEY          PIC S9(8) COMP SYNC.  
01 W-PAGE           PIC S9(8) COMP SYNC.  
01 W-LINE           PIC S9(8) COMP SYNC.  
01 W-VAL            PIC S9(8) COMP SYNC.  
01 W-PG-INFO.  
    02 W-GRP-NUM     PIC S9(4) COMP SYNC.  
    02 W-DBK-FORMAT PIC S9(4) COMP SYNC.  
.br/>.br/>.br/>PROCEDURE DIVISION.  
.br/>.br/>*** SAVE PAGE INFORMATION FOR THE DEPARTMENT RECORD TYPE ***  
    ACCEPT W-PG-INFO FOR DEPARTMENT.  
    IF DB-STATUS-OK  
        NEXT SENTENCE  
    ELSE  
        PERFORM IDMS-STATUS.  
*** COMPUTE WORK VALUE ***  
    MOVE 2 TO W-VAL.  
    SUBTRACT 1 FROM W-DBK-FORMAT.  
    PERFORM COMP-WORK W-DBK-FORMAT TIMES.  
*** COMPUTE PAGE AND LINE NUMBERS ***  
    COMPUTE W-PAGE = W-DBKEY / W-VAL.  
    COMPUTE W-LINE = W-DBKEY - (W-PAGE*W-VAL).  
.br/>.br/>COMP-WORK SECTION.  
    MULTIPLY W-VAL BY 2.
```

4.3.3 Saving a record's BIND address

To access a database record from a subprogram, you may need to know its storage address. You can use the `ACCEPT BIND ADDRESS` statement to acquire the storage address of a record that was bound in the calling program.

Because the `ACCEPT BIND ADDRESS` statement returns a storage address, it is typically used with subroutines written in either assembler or PL/I.

►► For more information on the `ACCEPT BIND ADDRESS` statement, refer to the language-specific CA-IDMS DML reference manual.

4.4 Checking for set membership

When accessing the database, you may find it necessary to obtain information about an owner or member record's set-membership status. To obtain set-specific information for a record occurrence, use the IF statement. By using the IF statement, you can determine:

- If the current occurrence of a specified set contains any member record occurrences (Is it empty?)
- If the current record of run unit participates as a member in a specified set defined with either the optional or the manual set membership option (Is it currently connected to an occurrence of the specified set?)

Each IF statement contains a conditional phrase and an imperative statement that specifies further action based on the outcome of the evaluation.

Native VSAM users: The IF statement is not allowed for sets defined with member records that are stored in native VSAM data sets.

Use of the IF EMPTY statement and the IF MEMBER statement is discussed below.

4.4.1 Using the IF EMPTY statement

After you have retrieved the owner record in a set, you can issue the IF EMPTY statement to determine if the set owns any member record occurrences. This allows you to control processing based on whether the set is empty.

Steps in determining if a set is empty: To determine if a set is empty, perform the following steps:

1. Establish currency for the set.
2. Issue the IF EMPTY statement.
3. Perform further processing as specified.

Note: The IF EMPTY statement always performs I/O to determine if the first record in the set has been logically deleted.

How to avoid an OBTAIN: You can also use the IF EMPTY statement to eliminate the need for using an OBTAIN FIRST WITHIN SET statement to walk a set, as illustrated in the program excerpt below.

Because the IF EMPTY statement determines that the set is not empty, you can be assured that the program can read at least one EMPLOYEE record before an ERROR-STATUS of 0307 (DB-END-OF-SET) is returned.

```

.
.
.
  IF DEPT-EMPLOYEE IS EMPTY
    MOVE NO-EMP-MESSAGE TO TITLE-OUT
  ELSE
    PERFORM A100-DEPT-EMP-WALK THRU A100-EXIT
      UNTIL DB-END-OF-SET.
A100-DEPT-EMP-WALK.
  OBTAIN NEXT WITHIN DEPT-EMPLOYEE.
.
.
.

```

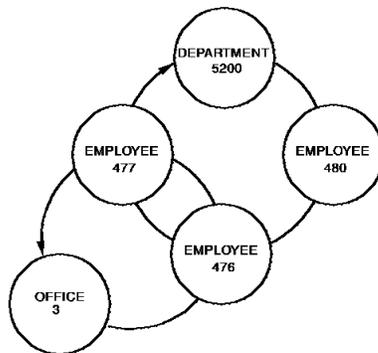
4.4.2 Using the IF MEMBER statement

If the current record of run unit participates as a member in a set defined with either the optional or the manual set membership option, you cannot assume that that record occurrence is also current of set. For example, an optional record may never have been connected to the set, or a manual record may have been disconnected from the set.

►► For more information about optional and manual set membership, see 3.3.3, “Set membership options” on page 3-10.

Failure to test for set membership: The figure below shows the invalid conclusion that can result from not testing for set membership.

Since EMPLOYEE 480 is not currently connected to an occurrence of the OFFICE-EMPLOYEE set, the OBTAIN OWNER statement retrieves the owner of the current record of set (OFFICE 3). This leads to the invalid assumption that EMPLOYEE 480 works in OFFICE 3.



	R. U. curr.	Record currencies			Set currencies		Area currencies	
	R U N U N I T	D E P A R T M E N T	E M P L O Y E E	O F F I C E	D E P T - E M P L O Y E E	O F F I C E - E M P L O Y E E	O R G - D E M O - R E G I O N	E M P - D E M O - R E G I O N
PREVIOUSLY ESTABLISHED CURRENCIES	5200	5200	477	3	5200	477	477	5200
OBTAIN FIRST WITHIN DEPT-EMPLOYEE.	480	5200	480	3	480	477	480	5200
OBTAIN OWNER WITHIN OFFICE-EMPLOYEE.	3	5200	480	3	480	3	480	3

Steps in testing for set membership: You can issue the IF MEMBER statement to ensure that a record occurrence currently participates as a member of a specified set.

To determine if a record participates as a member in a set, perform the following steps:

1. Establish run unit currency for the specified member record.
2. Issue the IF MEMBER statement.
3. Perform further processing, as specified

The program excerpt below uses the IF EMPTY and the IF MEMBER statements to facilitate database navigation.

The IF EMPTY statement determines if the DEPT-EMPLOYEE set is empty; the IF MEMBER statement determines whether the current of run unit (EMPLOYEE) participates as a member in the OFFICE-EMPLOYEE set.

```
PROCEDURE DIVISION.  
.  
.  
.  
A100-EMP-DEPT-OFF.  
    MOVE DEPT-ID-IN TO DEPT-ID-0410.  
    OBTAIN CALC DEPARTMENT  
    IF DB-REC-NOT-FOUND  
        GO TO GET-NEXT  
    ELSE IF DB-STATUS-OK  
        NEXT SENTENCE  
    ELSE  
        PERFORM IDMS-STATUS.  
*** TEST TO SEE IF SET IS EMPTY ***  
    IF DEPT-EMPLOYEE IS EMPTY  
        MOVE NO-EMP-MESSAGE TO TITLE-OUT  
    ELSE  
        PERFORM A100-WALK THRU A100-EXIT  
        UNTIL DB-END-OF-SET.  
A100-WALK.  
    OBTAIN NEXT WITHIN DEPT-EMPLOYEE.  
*** CHECK FOR ERROR-STATUS = 0307 ***  
    IF DB-END-OF-SET  
        GO TO A100-EXIT  
    ELSE IF DB-STATUS-OK  
        NEXT SENTENCE  
    ELSE  
        PERFORM IDMS-STATUS.  
*** TEST TO SEE IF EMPLOYEE IS CURRENTLY CONNECTED TO THE SET ***  
    IF NOT OFFICE-EMPLOYEE MEMBER  
        MOVE NO-OFF-MESSAGE TO TITLE-OUT  
    ELSE  
        OBTAIN OWNER WITHIN OFFICE-EMPLOYEE  
        PERFORM IDMS-STATUS  
        MOVE OFFICE-ADDRESS-0450 TO ADDRESS-OUT.  
    PERFORM U100-PRINT.  
A100-EXIT.  
EXIT.
```

4.5 Updating the database

DML modification statements update record occurrences in the database. By using these statements, which are discussed separately on the following pages, you can:

- Store a new record occurrence in the database
- Modify the contents of an existing record
- Erase a record from the database
- Connect a member record to a set
- Disconnect a member record from a set

4.5.1 Storing records

To add a record occurrence in the database, perform the following steps:

1. Specify a subschema that includes:
 - All sets in which the stored record is defined as an automatic member
 - The owner record of each of the required automatic sets

Note: Sets for which the stored record is defined as a manual member need not be defined in the subschema because the STORE statement does not access those sets. (An automatic member is connected automatically to the selected set occurrence when the record is stored; a manual member is not connected automatically to the selected set occurrence.)

2. Ready all affected areas in one of the update usage modes (for more information, see 2.4.4, “Area usage modes” on page 2-18).

Areas should be readied whether they are affected explicitly or implicitly (for example, as owner of a mandatory automatic set whose members are being stored).

3. Initialize the following variable storage fields:
 - All CALC, index, sort-key, and data fields
 - If the record being stored has a location mode of DIRECT, initialize the contents of the DIRECT-DBKEY field in the IDMS communications block with a suggested db-key value or a null db-key value of -1.
 - If the record is to be stored in a native VSAM relative-record data set (RRDS), initialize the contents of the DIRECT-DBKEY field with the relative record number that represents the location within the data set where the record is to be stored.
4. Establish currency for all set occurrences in which the stored record will participate as an *automatic* member. Depending on the set order, the stored record occurrence is positioned as follows:

- **If the named record is defined as a member of a set that is ordered FIRST or LAST**, the record that is current of set establishes the set occurrence to which the new record will be connected.
- **If the named record is defined as a member of a set that is ordered NEXT or PRIOR**, the record that is current of set establishes the set occurrence into which the new record will be connected *and* determines its position within the set.
- **If the named record is defined as a member of a sorted set**, the record that is current of set establishes the set occurrence into which the new record will be connected. The DBMS compares the sort key of the new record with the sort key of the current record of set to determine if the new record can be inserted into the set by movement in the next direction. If it can, the current of set remains positioned at the record that is current of set and the new record is inserted. If it cannot, the DBMS finds the owner of the current of set (not necessarily the current occurrence of the owner record type) and moves as far forward in the next direction as is necessary to determine the logical insertion point for the new record.

If the record being stored has a location mode of VIA, currency must be established for that VIA set, regardless of whether the record being stored is an automatic or manual member of that set. Current of the VIA set provides the suggested page for the record being stored.

5. Issue the STORE command.
6. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

What STORE does: STORE performs the following functions:

- Acquires space and assigns a database key for a new record occurrence in the database
- Transfers the value of the appropriate elements from program variable storage to the space acquired for the record occurrence in the database
- Connects the new record occurrence to all sets for which it is defined as an automatic member

The program excerpt below shows storing records in the database.

The program establishes the proper DEPARTMENT and OFFICE currencies and stores the new EMPLOYEE record.

```

PROCEDURE DIVISION.
    READ NEW-EMP-FILE-IN.
    AT END MOVE 'Y' TO EOF-SW.
    PERFORM A300-STORE-EMP THRU A300-EXIT
        UNTIL END-OF-FILE.
    FINISH.
    GOBACK.
A300-STORE-EMP.
    MOVE DEPT-ID-IN TO DEPT-ID-0410.
*** ESTABLISH CORRECT DEPARTMENT CURRENCY ***
    FIND CALC DEPARTMENT.
*** CHECK FOR ERROR-STATUS = 0326 ***
    IF DB-REC-NOT-FOUND
        THEN DISPLAY
            'DEPARTMENT ' DEPT-ID-IN ' NOT FOUND'
            'FOR NEW EMPLOYEE ID ' EMP-ID-IN
        GO TO A300-GET-NEXT
    ELSE IF DB-STATUS-OK
        NEXT SENTENCE
    ELSE
        PERFORM IDMS-STATUS.
        MOVE OFFICE-CODE-IN TO OFFICE-CODE-0450.
*** ESTABLISH CORRECT OFFICE CURRENCY ***
        FIND CALC OFFICE.
*** CHECK FOR ERROR-STATUS = 0326 ***
        IF DB-REC-NOT-FOUND
            THEN DISPLAY
                'OFFICE ' OFFICE CODE-IN ' NOT FOUND'
                'FOR NEW EMPLOYEE ID ' NEW-EMP-ID
            GO TO A300-GET-NEXT
        ELSE IF DB-STATUS-OK
            NEXT SENTENCE
        ELSE
            PERFORM IDMS-STATUS.
            PERFORM B300-INITIALIZE-EMPLOYEE.
*** STORE EMPLOYEE RECORD ***
            STORE EMPLOYEE.
            PERFORM IDMS-STATUS.
            PERFORM U500-WRITE-NEW-EMP-REPORT.
A300-GET-NEXT.
    READ NEW-EMP-FILE-IN
    AT END MOVE 'Y' TO EOF-SW.
A300-EXIT.
    EXIT.

```

4.5.2 Modifying records

To change a record occurrence in the database, perform the following steps:

1. Ready all affected areas in one of the update usage modes (for more information, see 2.4.4, “Area usage modes” on page 2-18).

Areas should be readied whether they are affected explicitly or implicitly (for example, as owner of a mandatory automatic set whose members are being modified).

2. Establish the specified record as current of run unit by issuing either a FIND or an OBTAIN statement.
3. Change the variable-storage fields of the record to be modified.

When using FIND, be sure to initialize *all* the appropriate values of the record to be modified. The best practice, however, is to use the OBTAIN statement to ensure that all the elements in the modified record are present in variable storage.

4. Issue the MODIFY command.
5. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

CALC and sort key considerations: The following special considerations apply to the modification of CALC- and sort-keys:

- If modification of a CALC- or sort-key will violate a duplicates-not-allowed option, the record is not modified and an error condition results.
- If a CALC-key is modified, successful execution of the MODIFY statement enables the record to be accessed on the basis of its new CALC-key value. The db-key of the specified record is not changed.
- If a sort-key is to be modified, the sorted set in which the specified record participates must be included in the subschema invoked by the program. A record occurrence that is a member of a set not defined in the subschema can be modified *if the undefined set is not sorted*.
- If any of the modified elements in the specified record are defined as sort keys for any set occurrence in which that record is currently a member, the DBMS tests that set occurrence to ensure that set order is maintained. If necessary, the DBMS disconnects the specified record and reconnects it in the set occurrence to maintain the set order specified in the schema.

Native VSAM considerations: The length of a record in an entry-sequenced data set (ESDS) cannot be changed even in the case of variable-length records.

The prime key for a key-sequenced data set (KSDS) cannot be modified.

Example of modifying records: The program excerpt below modifies records in the database.

The program retrieves the specified EMPLOYEE record and modifies the address and phone number. This program issues a COMMIT statement after every 100 updates. COMMIT releases all implicit exclusive locks and writes a checkpoint to the log file.

```

WORKING-STORAGE SECTION.
01 COMMIT-COUNTER          PIC S9(4) COMP VALUE +0.
PROCEDURE DIVISION.

    READ NEW-EMP-ADDRESS-FILE-IN.
    AT END MOVE 'Y' TO EOF-SW.
    PERFORM A300-CHANGE-ADDRESS THRU A300-EXIT
        UNTIL END-OF-FILE.

    FINISH.
    GOBACK.
A300-CHANGE-ADDRESS.
    MOVE EMP-ID-IN TO EMP-ID-0415.
*** RETRIEVE EMPLOYEE RECORD ***
    OBTAIN CALC EMPLOYEE.
*** CHECK FOR ERROR-STATUS = 0326 ***
    IF DB-REC-NOT-FOUND
        THEN DISPLAY
            'EMPLOYEE ' EMP-ID-IN ' NOT FOUND'
        GO TO A300-GET-NEXT
    ELSE IF DB-STATUS-OK
        NEXT SENTENCE
    ELSE
        PERFORM IDMS-STATUS.
        PERFORM U500-WRITE-OLD-ADDRESS.
*** CHANGE DATA AND ISSUE THE MODIFY STATEMENT ***
    MOVE NEW-ADDRESS-IN TO EMP-ADDRESS-0415.
    MOVE NEW-PHONE-IN TO EMP-PHONE-0415.
    MODIFY EMPLOYEE.
    PERFORM IDMS-STATUS.
    ADD 1 TO COMMIT-COUNTER.
    IF COMMIT-COUNTER > 100 THEN
        COMMIT
        PERFORM IDMS-STATUS
        MOVE 0 TO COMMIT-COUNTER.
    PERFORM U0510-WRITE-NEW-ADDRESS.
A300-GET-NEXT.
    READ NEW-EMP-ADDRESS-FILE-IN
    AT END MOVE 'Y' TO EOF-SW.
A300-EXIT.
EXIT.

```

4.5.3 Erasing records

To delete a record occurrence from the database, perform the following steps:

1. Specify a subschema that includes the following:
 - All sets in which the specified record participates as owner either directly or indirectly (for example, as owner of a set with a member that is owner of another set)
 - All member record types in the sets specified above
2. Ready all affected areas in one of the update usage modes (for more information, see 2.4.4, “Area usage modes” on page 2-18).

Areas should be readied whether they are affected explicitly or implicitly (for example, as owner of a mandatory automatic set whose members are being erased).
3. Establish the specified record as current of run unit.

4. Issue the ERASE command.
5. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

What ERASE does: The ERASE statement performs the following functions:

- Disconnects the specified record from all set occurrences in which it participates as a member and logically or physically deletes the record from the database
- Optionally erases all records that are mandatory members of set occurrences owned by the specified record
- Optionally disconnects or erases all records that are optional members of set occurrences owned by the specified record

ERASE is a two-step procedure that first cancels the existing membership of the named record in specific set occurrences and then releases for reuse the space occupied by the named record and its db-key. Erased records are unavailable for further processing by any DML statement.

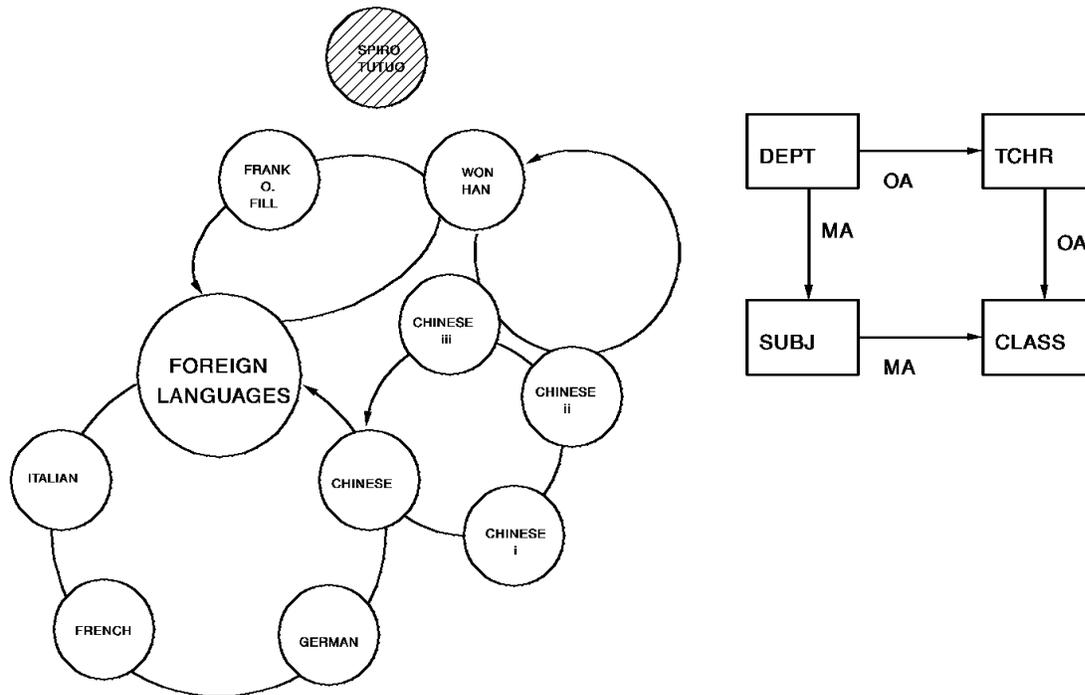
Currencies after an ERASE: Following successful execution of an ERASE statement:

- Currency is *nullified* for all record types involved in the erase, both explicitly and implicitly.
- Currency is *preserved* for run unit and area.
- Next, prior, and owner currencies are *preserved* for sets from which the last record occurrence was erased. These currencies enable you to retrieve the next or prior records within the area or the next, prior, or owner records within the set in which the erased record participated.

ERASE statement with no options: To issue the ERASE statement with no options:

- The record must be current of run unit.
- All sets in which the record participates as owner must be empty (that is, an error condition will result if this version of the ERASE statement is attempted against an owner record that has any member occurrences).

In the illustration below, an ERASE TCHR statement with no options disconnects the shaded occurrence (SPIRO TUTUO) from membership in the DEPT-TCHR set and then erases the record occurrences. This statement executes without error because the TCHR-CLASS set owned by record occurrence SPIRO TUTUO is an empty set (he doesn't have any classes).

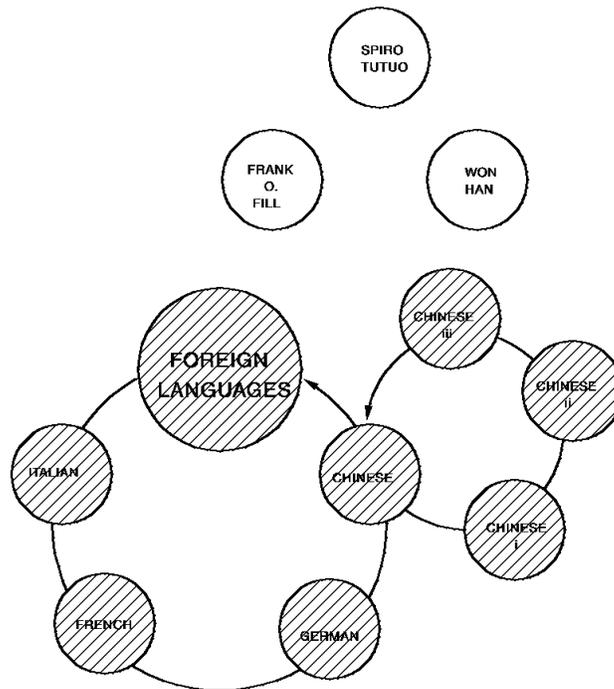


ERASE options: You can qualify the ERASE statement with these options to specify how the ERASE statement affects member occurrences:

- PERMANENT
- SELECTIVE
- ALL

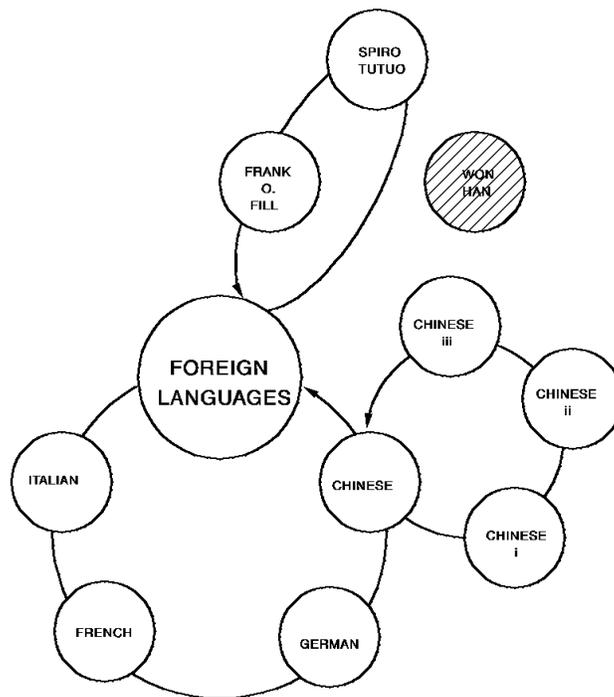
ERASE PERMANENT: ERASE PERMANENT erases the specified record and all mandatory member record occurrences owned by the specified record. Optional member records are disconnected. If any of the erased mandatory members are themselves the owners of any set occurrences, they are erased as if they were directly the object record of an ERASE PERMANENT statement (that is, all mandatory members of such sets are also erased). This process continues until all direct and indirect members have been processed.

In the illustration below, currency has been set on the FOREIGN LANGUAGES occurrence of the DEPT record, and an ERASE DEPT PERMANENT statement has been issued. All subjects are erased because they are mandatory members of the DEPT-SUBJ set. All classes are also erased because they are mandatory members of the SUBJ-CLASS set. However, since membership in DEPT-TCHR is optional, members of the set owned by FOREIGN LANGUAGES are disconnected, not erased.



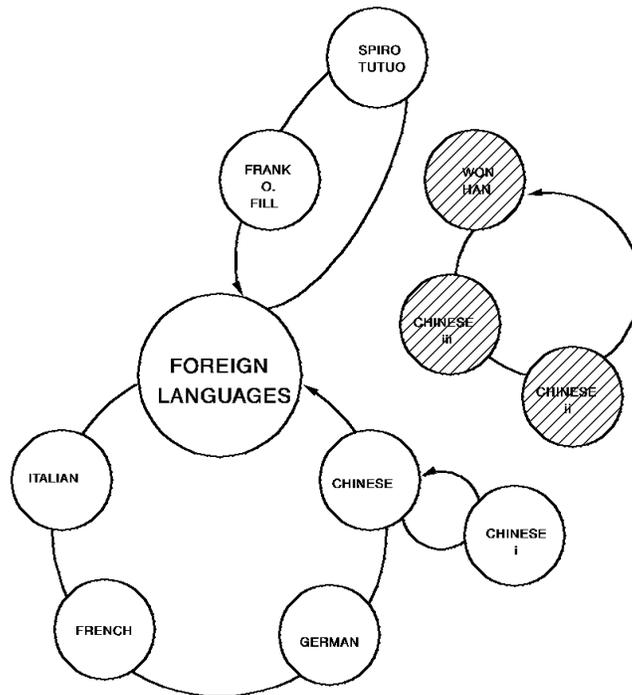
ERASE SELECTIVE: ERASE SELECTIVE erases the specified record and all mandatory member record occurrences owned by the specified record. Optional member records are erased if they do not *currently participate* as members in other set occurrences. All erased member records that are themselves the owners of any set occurrences are treated as if they were the object of an ERASE SELECTIVE statement.

In the illustration below, currency has been set on the WON HAN occurrence of the TCHR record, and an ERASE TCHR SELECTIVE statement has been issued. Since WON HAN was the owner of two occurrences of the TCHR-CLASS set, an ERASE statement without an option would fail. The SELECTIVE option prevents these occurrences from being erased because they currently participate in another set (SUBJ-CLASS). This means, in effect, that the department still offers the classes even though the teacher is gone.



ERASE ALL: ERASE ALL erases the specified record and all mandatory and optional member record occurrences owned by the specified record. All erased member records that are themselves the owners of any set occurrences are treated as if they were the object record of an ERASE ALL statement.

In the illustration below, currency has been set on the WON HAN occurrence of the TCHR record, and an ERASE TCHR ALL statement has been issued. Since WON HAN was the owner of two occurrences of the TCHR-CLASS set, the ERASE ALL statement erases these member occurrences. This means, in effect, that when the teacher leaves the department, his classes are dropped.



4.5.4 Connecting records to a set

To connect a record to a set or to reconnect a record that has been disconnected from a set, perform the following steps:

1. Ready all affected areas in one of the update usage modes (for more information, see 2.4.4, "Area usage modes" on page 2-18).

Areas should be readied whether they are affected explicitly or implicitly (for example, as owner of a set whose members are being connected).

2. Establish the following currencies:
 - The specified record must be current of its record type.
 - The occurrence of the set into which the specified record will be connected must be current of set. If set order is NEXT or PRIOR, current of set also determines the position at which the specified record will be connected within the set.

3. Issue the CONNECT command; CONNECT establishes the specified record occurrence as a member of a set occurrence.
4. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

The specified record must previously have been either stored (manual membership) or disconnected (optional membership).

Native VSAM users: The CONNECT statement is not valid since all sets in native VSAM files must be defined as mandatory automatic.

4.5.5 Disconnecting records from a set

To cancel the membership of a record occurrence in a set occurrence defined with the optional set membership option, perform the following steps:

1. Ready all affected areas in one of the update usage modes (for more information, see 2.4.4, “Area usage modes” on page 2-18).

Areas should be readied whether they are affected explicitly or implicitly (for example, as owner of a set whose members are being disconnected).

2. Establish the following currencies:
 - The specified record must be current of its record type.
 - The specified record must currently participate as a member in an occurrence of the named set.
3. Issue the DISCONNECT statement.
4. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

Accessing a disconnected record: Following successful execution of the DISCONNECT statement, you cannot access the record through the set for which membership was canceled. You can still access the record in the following ways:

- Through an area sweep
- By using its db-key
- Through any other sets in which it still participates
- If it has a location mode of CALC, by using its CALC key

Currencies after a DISCONNECT: Although a successfully executed DISCONNECT statement nullifies currency in the specified set, the DBMS maintains next, prior (if specified), and owner currencies so you can still issue the OBTAIN NEXT, PRIOR, or OWNER WITHIN SET statements.

Native VSAM users: The DISCONNECT statement is not valid because all sets in native VSAM files must be defined as mandatory automatic.

Example of disconnecting and connecting records: The program excerpt below disconnects and subsequently reconnects EMPLOYEE records in the DEPT-EMPLOYEE set.

Employees have been transferred to another department. The program ensures that both the new and the old departments exist before disconnecting the EMPLOYEE record from the old DEPT-EMPLOYEE set and connecting it to the new DEPT-EMPLOYEE set.

```
DATA DIVISION.
FILE SECTION.
FD DEPT-TRANSFER-FILE.
01 TRANS-EMP-REC-IN.
   02 NEW-DEPT-ID-IN          PIC 9(4).
   02 OLD-DEPT-ID-IN         PIC 9(4).
   02 EMP-ID-IN              PIC 9(4).
WORKING-STORAGE SECTION.
01 SWITCHES.
   05 EOF-SW                  PIC X    VALUE 'N'.
   88 END-OF-FILE             VALUE 'Y'.
01 CONNECT-DBKEY             PIC S9(8) COMP SYNC.
PROCEDURE DIVISION.

      .
      READ DEPT-TRANSFER-FILE
      AT END MOVE 'Y' TO EOF-SW.
      PERFORM A300-DISCONNECT-EMP THRU A300-EXIT
      UNTIL END-OF-FILE.

      FINISH.
      GOBACK.
A300-DISCONNECT-EMP.
      MOVE NEW-DEPT-ID-IN TO DEPT-ID-0410.
      FIND CALC DEPARTMENT.
*** IF ERROR-STATUS = 0326, NEW DEPT ID IS INVALID ***
      IF DB-REC-NOT-FOUND
      DISPLAY
      'NEW DEPARTMENT ' NEW-DEPT-ID-IN ' NOT FOUND'
      'FOR EMPLOYEE ID ' EMP-ID-IN
      GO TO A300-GET-NEXT
      ELSE IF DB-STATUS-OK
      NEXT SENTENCE
      ELSE
      PERFORM IDMS-STATUS.
*** SAVE NEW DEPT DB-KEY TO REOBTAIN RECORD LATER ***
      MOVE DBKEY TO CONNECT-DBKEY.
      PERFORM IDMS-STATUS.

      MOVE OLD-DEPT-ID-IN TO DEPT-ID-0410.
      FIND CALC DEPARTMENT.
*** IF ERROR-STATUS = 0326, OLD DEPT ID IS INVALID ***
      IF DB-REC-NOT-FOUND
      DISPLAY
      'OLD DEPARTMENT ' OLD-DEPT-ID-IN ' NOT FOUND'
      'FOR EMPLOYEE ID ' EMP-ID-IN '
      GO TO A300-GET-NEXT
      ELSE IF DB-STATUS-OK
      NEXT SENTENCE
      ELSE
      PERFORM IDMS-STATUS.
      MOVE EMP-ID-IN TO EMP-ID-0415.
      OBTAIN CALC EMPLOYEE.
```

```
*** IF ERROR-STATUS = 0326, EMP ID IS INVALID ***
  IF DB-REC-NOT-FOUND
    DISPLAY
    'EMPLOYEE ' EMP-ID-IN ' NOT FOUND'
    'FOR OLD DEPARTMENT ' OLD-DEPT-ID-IN
    '*** NEW DEPARTMENT ' NEW-DEPT-ID-IN
    GO TO A300-GET-NEXT
  ELSE IF DB-STATUS-OK
    NEXT SENTENCE
  ELSE
    PERFORM IDMS-STATUS.
*** CHECK IF EMPLOYEE IS A MEMBER IN DEPT-EMPLOYEE SET ***
  IF NOT DEPT-EMPLOYEE MEMBER
    DISPLAY
    'EMPLOYEE ' EMP-ID-IN
    'NOT CONNECTED TO DEPARTMENT ' OLD-DEPT-ID-IN
    GO TO A300-GET-NEXT.
DISCONNECT EMPLOYEE FROM DEPT-EMPLOYEE.
PERFORM IDMS-STATUS.
*** REACCESS NEW DEPARTMENT USING ITS DB-KEY ***
  FIND DEPARTMENT DB-KEY IS CONNECT-DBKEY.
  PERFORM IDMS-STATUS.
CONNECT EMPLOYEE TO DEPT-EMPLOYEE.
PERFORM IDMS-STATUS.
```

4.6 Locking records

You can explicitly place a shared or exclusive lock on a record that is current of run unit, record, set, or area. You should place explicit locks on records for the following reasons:

- To ensure later access to a specified record occurrence because implicit locks are released due to additional navigation (that is, to prevent other run units from placing an exclusive lock on the occurrence)
- To ensure exclusive access to a specified record occurrence (that is, to prevent other run units from accessing the occurrence in any way)

Steps in locking records: To place an explicit lock on a record, perform the following steps:

1. Establish the appropriate run unit, record, set, or area currency.
2. Issue the KEEP statement.
3. Perform the IDMS-STATUS routine if the DBMS returns a nonzero value.

How long locks are held: The DBMS maintains record locks until the next COMMIT, FINISH, or ROLLBACK statement.

Alternatively, you can use the KEEP option of the FIND/OBTAIN statement to place locks on records as they are retrieved.

►► For more information on shared and exclusive locks, see 2.4.2, “Record locks” on page 2-13.

Example of using KEEP to lock a record: The program excerpt below shows the use of the KEEP statement in a program that connects and disconnects records.

The program places an explicit shared lock on the new DEPARTMENT record occurrence to prevent other run units from modifying it and to guarantee access later in the program.

```
A300-DISCONNECT-EMP.
  MOVE NEW-DEPT-ID-IN TO DEPT-ID-0410.
  FIND CALC DEPARTMENT.
*** IF ERROR-STATUS = 0326, NEW DEPT ID IS INVALID ***
  IF DB-REC-NOT-FOUND
    DISPLAY
      'NEW DEPARTMENT ' NEW-DEPT-ID-IN ' NOT FOUND'
      'FOR EMPLOYEE ID ' EMP-ID-IN
    GO TO A300-GET-NEXT
  ELSE IF DB-STATUS-OK
    NEXT SENTENCE
  ELSE
    PERFORM IDMS-STATUS.
*** LOCK NEW DEPARTMENT TO ENSURE THAT ***
*** OTHER RUN UNITS DO NOT MODIFY IT ***
*** KEEP CURRENT DEPARTMENT. ***
*** SAVE NEW DEPT DB-KEY TO REOBTAIN RECORD LATER ***
  MOVE DBKEY TO CONNECT-DBKEY.
  PERFORM IDMS-STATUS.

  MOVE OLD-DEPT-ID-IN TO DEPT-ID-0410.
  FIND CALC DEPARTMENT.
*** IF ERROR-STATUS = 0326, OLD DEPT ID IS INVALID ***
  IF DB-REC-NOT-FOUND
    DISPLAY
      'OLD DEPARTMENT ' OLD-DEPT-ID-IN ' NOT FOUND'
      'FOR EMPLOYEE ID ' EMP-ID-IN '
    GO TO A300-GET-NEXT
  ELSE IF DB-STATUS-OK
    NEXT SENTENCE
  ELSE
    PERFORM IDMS-STATUS.
  MOVE EMP-ID-IN TO EMP-ID-0415.
  OBTAIN CALC EMPLOYEE.
*** IF ERROR-STATUS = 0326, EMP ID IS INVALID ***
  IF DB-REC-NOT-FOUND
    DISPLAY
      'EMPLOYEE ' EMP-ID-IN ' NOT FOUND'
      'FOR OLD DEPARTMENT ' OLD-DEPT-ID-IN
      '*** NEW DEPARTMENT ' NEW-DEPT-ID-IN
    GO TO A300-GET-NEXT
  ELSE
    PERFORM IDMS-STATUS.
```

Chapter 5. Advanced DML Programming Topics

- 5.1 About this chapter 5-3
- 5.2 Copying record definitions and their synonyms 5-4
- 5.3 Accessing bill-of-materials structures 5-6
 - 5.3.1 Storing a bill-of-materials structure 5-6
 - 5.3.2 Retrieving a bill-of-materials structure 5-8

5.1 About this chapter

This chapter explains how to direct the precompiler to copy records from the dictionary and how to access a bill-of-materials structure in the database.

5.2 Copying record definitions and their synonyms

Typically, you copy subschema records into variable storage using the primary name given by the DBA either in the schema or through IDD (and then copying the record into a schema). Synonyms are alternative names for existing dictionary entities. A given file, record, or element can contain multiple names through the use of IDD synonyms. This allows all programs that use that record to access the same data description in the dictionary.

Uses of synonyms: Synonyms are typically used for the following reasons:

- **To allow you to copy schema-owned records into a program whose subschema is not associated with that schema.** If a record has been copied into a schema, it can be copied only into a program that uses a subschema associated with that schema; for example:

```
COPY IDMS RECORD EMPLOYEE.
```

However, if your program uses a subschema that is not associated with that schema, you can copy a synonym that has been defined for the record. To copy a synonym, specify a **VERSION** clause:

```
COPY IDMS RECORD EMPLOYEE VERSION 100.
```

- **To allow different programming languages to access the same record definition.** For example, in Assembler the EMPLOYEE record can be defined as EMPLOYEE; in FORTRAN, EMPLOY; and so on.

Note: The precompiler for PL/I automatically converts hyphens to underscores. That is, you define schema and IDD records with hyphens:

```
INCLUDE IDMS (PLITEST-REC);
```

The precompiler converts them to underscores:

```
DMLP      INCLUDE IDMS (PLITEST_REC);
          DECLARE 1 PLITEST_REC,
                2 PLITEST_ELEM CHARACTER (3);
```

Terminology for using synonyms: You should be familiar with the following terms:

- **Schema-owned** refers to any record that is defined in the schema.
- **IDD-defined** refers to any record that is defined using the DDDL compiler that has not been included in a schema.
- **Mode** refers to the operating mode of your program (that is, BATCH, IDMS-DC, DC-BATCH, CICS, and so on). IDD-defined records can be assigned one of these modes, a mode of NON-MODESPECIFIC, or no mode attribute at all.
- **Language attribute** refers to the optional attribute that can be included in IDD-defined records and synonyms. For example, LANGUAGE IS COBOL, LANGUAGE IS PL/I, or LANGUAGE IS DC.

How the precompiler performs COPY IDMS: When the precompiler selects which record or synonym to copy into your program, it first checks to see if you have specified a VERSION clause in the COPY command. If no VERSION clause is given, a two-fold search is undertaken, first for a record associated with the subschema and then, if the first test fails, for an IDD-defined record.

To determine if the specified record is associated with the subschema, the precompiler performs the following steps:

1. **Forms a table of records defined in the subschema and their synonyms.** This table contains all records copied into the subschema and, for every record copied, the names of its synonyms *not* copied into another subschema.
2. **Searches this table to match the name of the record in the COPY statement.** If a match is found, that record is copied in; if no match is found, the search continues as described below.

If you specify a VERSION clause or if the test listed above fails, the precompiler assumes that the record is an IDD-defined record and performs the following steps:

1. **Forms a table of IDD-defined records and their synonyms.** This table contains all IDD-defined records that have a synonym that matches the name listed in the COPY request.
2. **Checks the VERSION clause.** If a VERSION clause is given, the record's version must match or another record is chosen as described in step 1. If no VERSION clause is given, the record with the highest version meeting all the given criteria is chosen.
3. **Checks the builder code.** The record candidate is tested for being either schema-owned (builder code of S) or a subschema view (builder code of V). If either is true, another record is chosen as described in step 1. If neither is true, the candidate is chosen and the record is copied into the program.
4. **Checks the language attribute.** If the record has a language attribute matching that of the compiler being used (for example, PL/I), the record remains a candidate. Also, a record that has no language associated with it remains a candidate.
5. **Checks the mode.** The mode associated with the record is compared with the operating mode specified in the program. If they match, the test continues. If there is no record with a match on mode, then a search is made for a record with a mode of NON-MODESPECIFIC. If there is no record with a match on NON-MODESPECIFIC, a record is searched for that has no mode associated with it. If no record is found, another record is chosen as described in step 1.

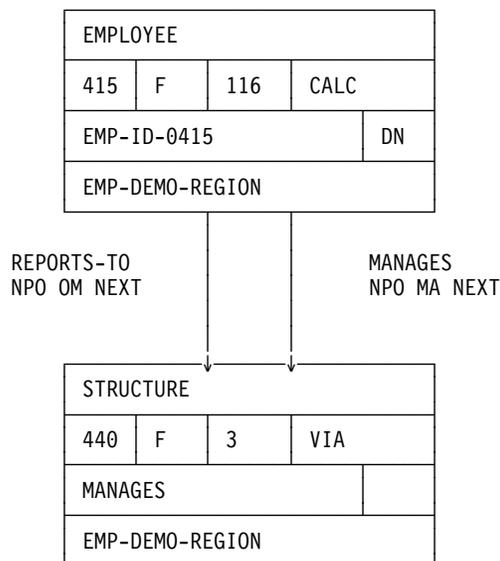
5.3 Accessing bill-of-materials structures

A bill-of-materials structure is a relationship between record occurrences *of the same type*. This structure is derived from the manufacturing environment where it is used to demonstrate relationships between parts: a part can be a component of another part and a part can contain other parts as its components.

This structure is typically represented as a many-to-many relationship (that is, by using two sets and a junction record).

Example of a bill-of-materials structure: In the EMPLOYEE database, a bill-of-materials structure signifies relationships between managers and subordinates: an employee can *manage* other employees through the MANAGES set, and can also be *managed by* other employees through the REPORTS-TO set.

The figure below shows this bill-of-materials structure. The STRUCTURE record serves as the junction record between employees and their managers. Note that one set is defined with the automatic set membership option and the other is defined with the manual set membership option.



5.3.1 Storing a bill-of-materials structure

To store a bill-of-materials structure in the database, perform the following steps:

1. Set run-unit currency at the owner record in the automatic set:

```
MOVE 15 TO EMP-ID-0415.
OBTAIN CALC EMPLOYEE.
```

2. Initialize and store the junction record:

```
PERFORM A100-INITIALIZE-STRUCTURE.
STORE STRUCTURE.
```

The DBMS automatically connects the STRUCTURE record to the automatic set (MANAGES); EMPLOYEE 15 is now defined as the manager in the bill-of-materials structure.

3. Set run-unit currency at the owner record in the manual set:

```
MOVE 467 TO EMP-ID-0415.
OBTAIN CALC EMPLOYEE.
```

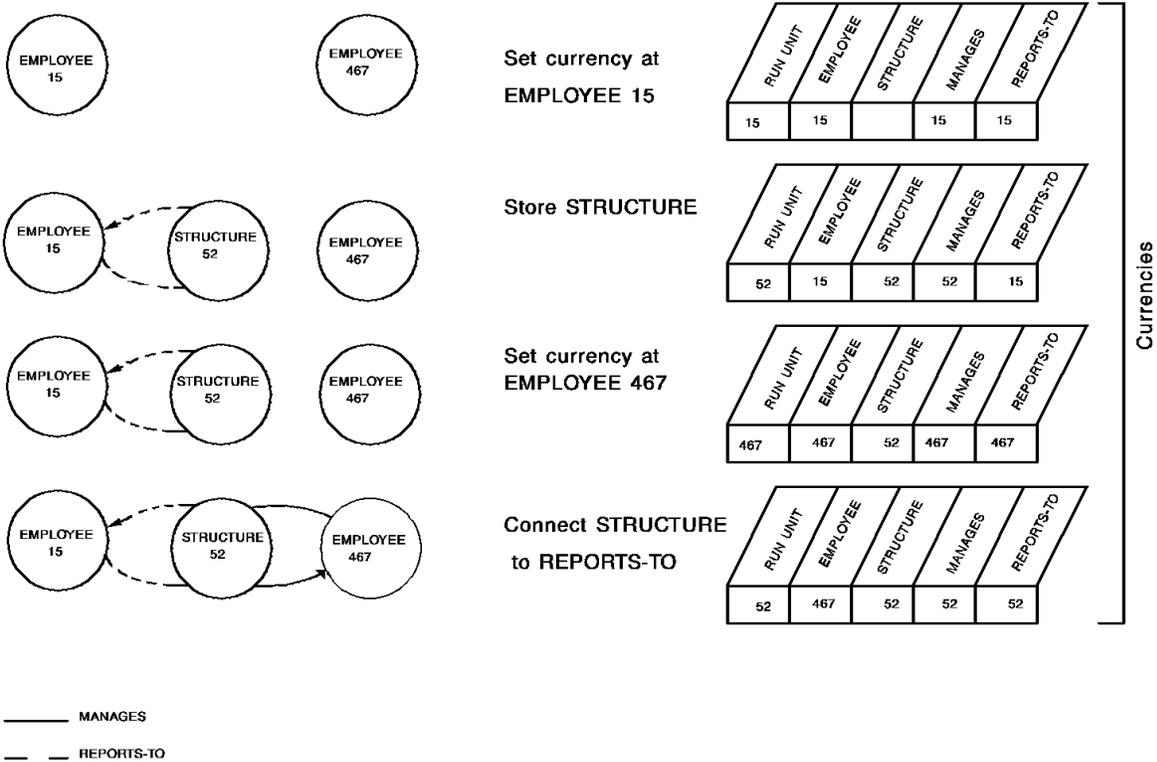
4. Connect the junction record to the manual set:

```
CONNECT STRUCTURE TO REPORTS-TO.
```

The bill-of-materials structure is now complete; EMPLOYEE 467 reports to EMPLOYEE 15.

Example of storing a bill-of-materials structure: The figure below shows the steps and currencies involved in storing an occurrence of a bill-of-materials structure.

To define EMPLOYEE 15 as the manager of EMPLOYEE 467, store and connect a STRUCTURE record as a member of the two EMPLOYEE records: EMPLOYEE 15 in the MANAGES set and EMPLOYEE 467 in the REPORTS-TO set.



5.3.2 Retrieving a bill-of-materials structure

A bill-of-materials structure can contain a variable number of levels. Tracing all records under a given record (for example, finding a manager and all subordinates, and all of their subordinates, and so on) is called an **explosion** of the structure for that record. Tracing all records above a given record (for example, finding an employee and manager, and the manager's manager, and so on) is called an **implosion** of the structure for that record.

To perform a multi-level explosion or implosion, you must maintain a stack of db-keys in order to reestablish the appropriate currencies.

Steps to retrieve one bill-of-materials level: To retrieve a manager and one level of employees, perform the following steps:

1. Retrieve the manager's EMPLOYEE record:

```
MOVE 15 TO EMP-ID-0415.  
OBTAIN CALC EMPLOYEE.
```

2. Retrieve the first STRUCTURE record in the MANAGES set:

```
FIND NEXT STRUCTURE WITHIN MANAGES.
```

3. Because REPORTS-TO is defined as OM, you must test for set membership:

```
IF NOT REPORTS-TO MEMBER  
GO TO A100-EXIT.
```

4. Retrieve the owner EMPLOYEE record in the REPORTS-TO set:

```
OBTAIN OWNER WITHIN REPORTS-TO.
```

5. FIND the current STRUCTURE record to reestablish the original currency within the MANAGES set:

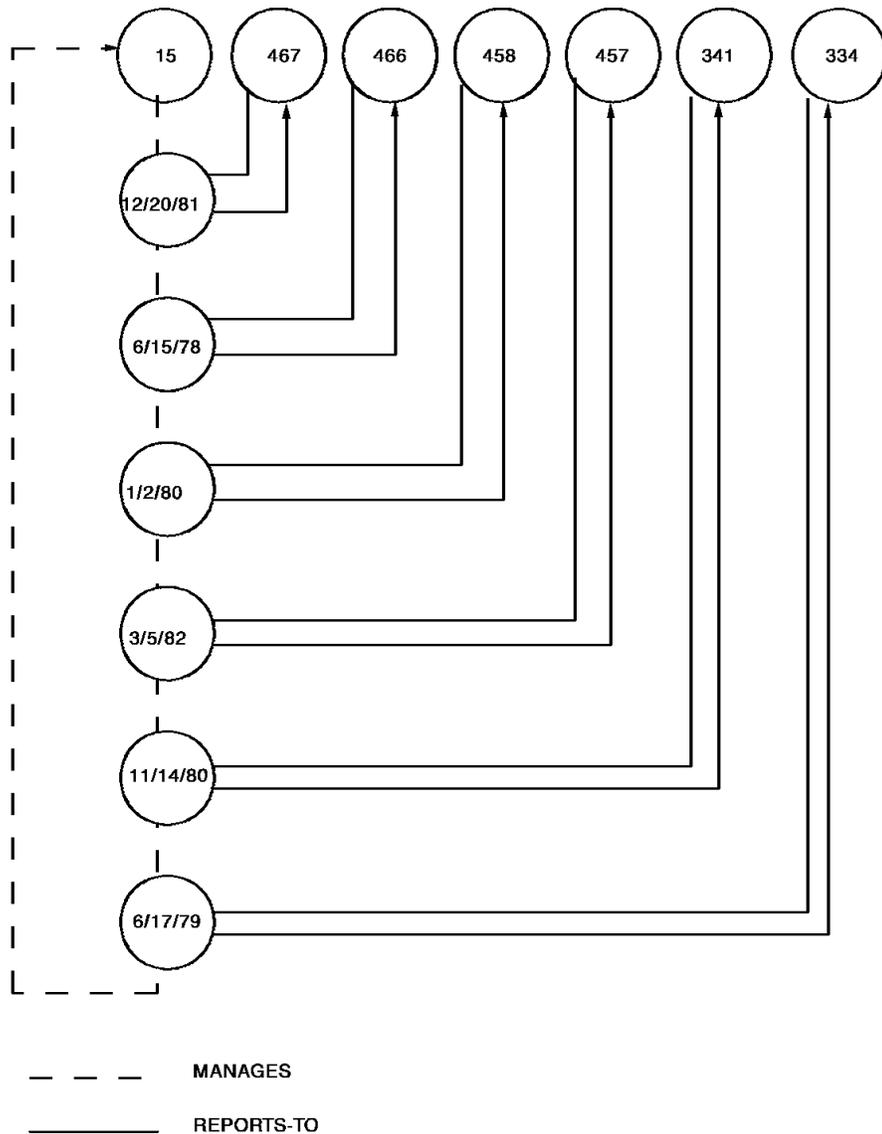
```
FIND CURRENT STRUCTURE.
```

6. Retrieve the next STRUCTURE record in the MANAGES set:

```
FIND NEXT STRUCTURE WITHIN MANAGES.
```

Perform steps 3 through 6 iteratively until step 6 returns a status of 0307 (DB-END-OF-SET).

Example of one bill-of-materials level: The figure below shows the relationship between manager and employees by showing all the employees managed by employee 15.



Steps to retrieve additional levels: To retrieve an EMPLOYEE record, its manager's EMPLOYEE record, its manager's manager, and so on, perform the following steps:

1. Retrieve the specified EMPLOYEE record:

```
MOVE 91 TO EMP-ID-0415.
OBTAIN CALC EMPLOYEE.
```

2. Retrieve the STRUCTURE record in the REPORTS-TO set:

```
FIND NEXT STRUCTURE WITHIN REPORTS-TO.
IF ERROR-STATUS = '0307'
GO TO A100-EXIT.
```

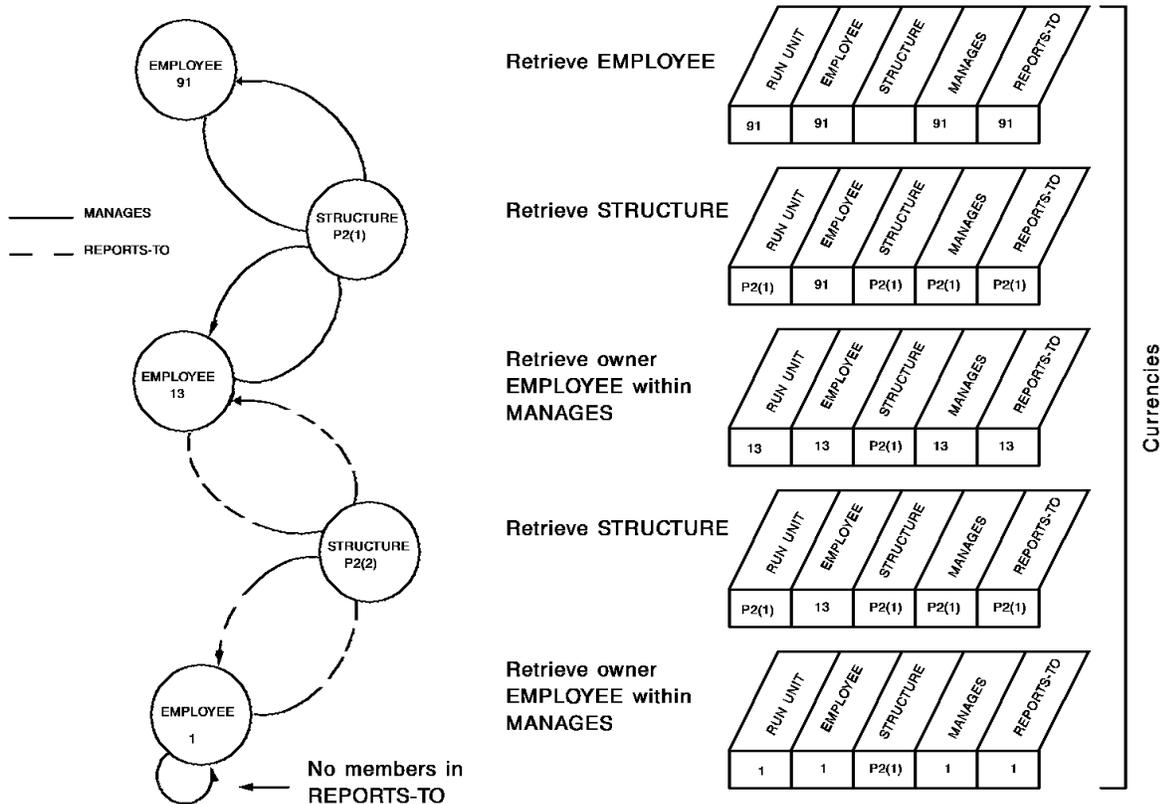
3. Optionally, test the junction record for predetermined criteria (for example, if you only want managers for a specific project). Testing for selection criteria in the junction record can prevent looping if there are any circular structures defined.

4. Retrieve the owner EMPLOYEE record in the MANAGES set:

OBTAIN OWNER EMPLOYEE WITHIN MANAGES.

Perform steps 2 through 4 iteratively until step 2 returns a status of 0307, indicating that the REPORTS-TO set is empty.

Example of retrieving additional levels: The figure below shows the relationship between an employee and all managers on the P2 project by showing the hierarchy of managers above EMPLOYEE 91 on the project.



Chapter 6. Introduction to Online Programming

6.1 About this chapter	6-3
6.2 DC as an operating system	6-4
6.3 Transaction and task processing	6-5
6.4 Pseudoconversational programming	6-6
6.5 Performance considerations	6-9
6.6 Error handling	6-10
6.7 Using the IDMS communications block	6-11

6.1 About this chapter

CA-IDMS provides the teleprocessing monitor DC, which is fully integrated with CA-IDMS database services (DB) and the CA-IDMS dictionary (IDD). DC enables your online application to request data communications and database functions through DML statements.

Online applications typically involve interactive processing between a user and an application. Because online programs are interactive, they require certain programming strategies that differ from the strategies employed in batch programming.

This chapter reviews the following online programming considerations:

- **DC as an operating system** — DC functions as an operating system within an operating system.
- **Transaction and task processing** — Transactions and tasks are the logical units of work in an online application.
- **Programming style** — Pseudoconversational programming enables the DC system to run more efficiently, thereby enabling optimal program response time.
- **Performance considerations** — You should use CA-IDMS resources efficiently in order to provide each user with an efficient work environment.
- **Error handling** — Error handling is approached by displaying messages on the terminal and allowing the user to resubmit the data.
- **Communication with DC** — The IDMS communications block is the interface between your program and DC.

6.2 DC as an operating system

DC functions as an operating system running under the main operating system (OS/390, VSE/ESA, or BS2000/OSD). That is, the operating system perceives DC as a long-running batch job. DC controls the concurrent execution of multiple application programs and performs many other functions typically associated with the operating system.

How DC services programs: Your program can also request DC to perform functions that are typically handled automatically by the operating system. Such functions include:

- **System-control functions** — A program can request DC to perform many system-control functions typically associated with the operating system, including:
 - Loading programs and tables
 - Transferring control between programs
 - Managing dynamic variable storage
 - Synchronizing program execution by using events and resources
- **I/O functions** — A program can request DC to transfer data between variable storage and a terminal device or between variable storage and the database.
- **Scratch and queue functions** — A program can request DC to store data in local or system-wide areas for later access.
- **Utility functions** — A program can request DC to perform various utility functions such as obtaining task or system information, obtaining the current time, writing printed reports, and sending messages.

6.3 Transaction and task processing

In online programming, your application is logically divided into transactions and tasks.

Transaction: A transaction is a logical unit of work performed by the user. Updating an employee record and adding a skill are two typical transactions.

Each transaction consists of one or more tasks, organized and executed to form a logical unit of work.

Task: A task is a logical unit of work performed by the DC system. Processing within a task is transparent to the user. Displaying a screen that solicits an employee ID number is one typical task; reading that ID and retrieving the specified employee information is another.

A task, which can consist of one or more programs, is invoked by a task code. The user usually begins a transaction by typing the initial task code and pressing [Enter]. When a task finishes, it specifies the next task to be invoked on that terminal.

6.4 Pseudoconversational programming

The most efficient way of coding tasks and transactions is through pseudoconversational programming. This technique uses a series of terminal tasks to conduct a transaction; that is, each task terminates after soliciting information from the user and specifying the next task to be invoked. The next task, which is initiated only when the user presses an attention identifier (AID) key, accepts the operator response and performs the specified processing.

Note: The AID keys are:

[Enter]
[Clear]
[PF1] through [PF24]
[PA1] through [PA3]

For details, see 7.2, “Mapping mode” on page 7-4.

While the user is entering data, all system resources (such as program or storage pool space that the task might have been using) are available to other tasks.

Conversational programs: Conversational programs, unlike pseudoconversational programs, solicit and accept information from the user and perform all other processing in one task. While the user is entering data, (which may take seconds, minutes, or hours) the task is executing; all resources held by the task are not available to other tasks. The conversational approach to transaction processing is *not* recommended.

Pseudoconversational programming considerations: The following considerations apply to pseudoconversational programming:

- You must identify the task to be started when the user presses an AID key. Your program must include a DC RETURN request that specifies the NEXT TASK CODE. For example, the following DC RETURN requests would be associated with the pseudoconversational transaction illustrated in this program excerpt:

```
EMPID program
.
.
.
    DC RETURN NEXT TASK CODE 'EINF'.

EMPINFO program
.
.
.
    DC RETURN NEXT TASK CODE 'EMOD'.

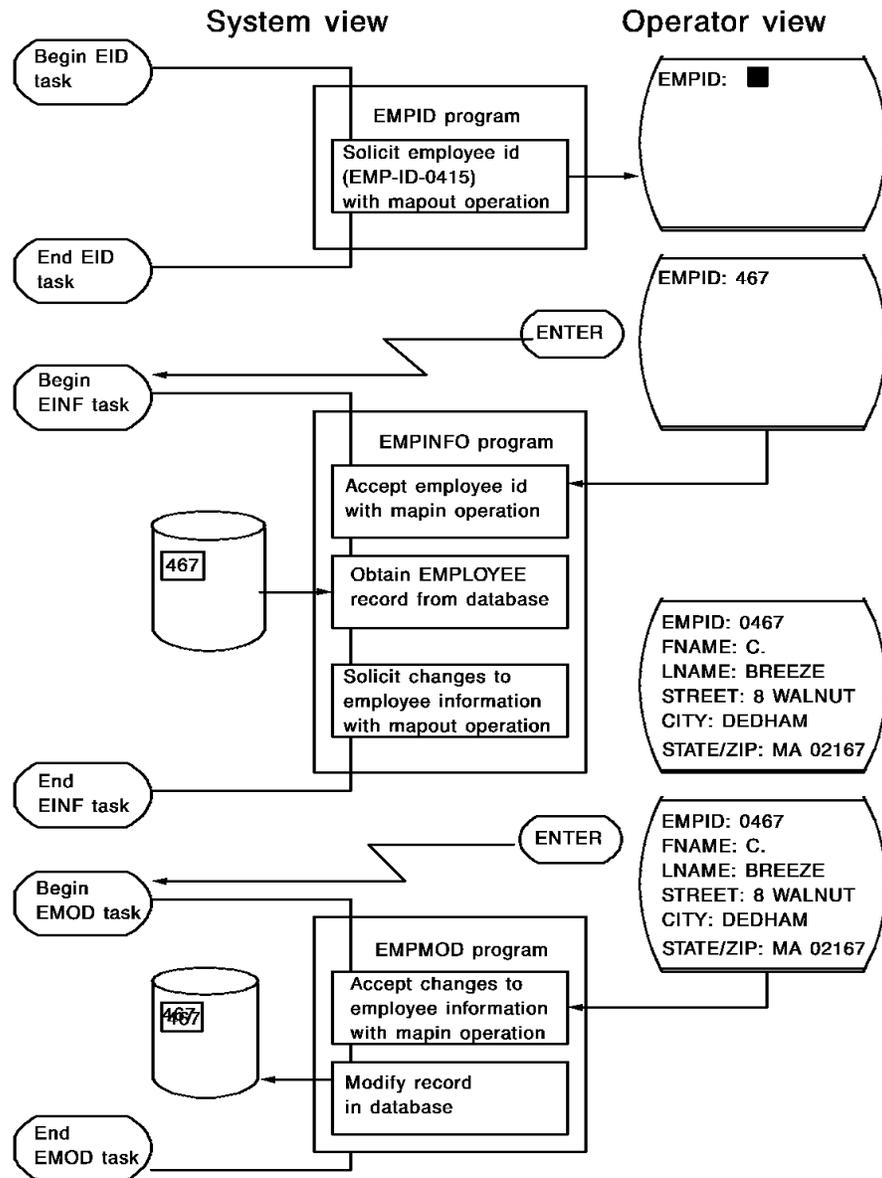
EMPMOD program
.
.
.
    DC RETURN.
```

►► For more information on DC RETURN, see 9.2.1, “Returning to a higher level program” on page 9-5.

- Within pseudoconversational transactions, you may need to pass data from program to program or from task to task. DC provides data management facilities to maintain such data. However, the use of these facilities can involve trade-offs among storage pool overhead, disk access speed, allocation of system resources, and recovery procedures. For more information on DC data management facilities, refer to Chapter 8, “Storage, Scratch, and Queue Management” on page 8-1.

Pseudoconversational program example: The figure below shows the pseudoconversational approach to transaction processing.

Processing associated with modifying passenger information is performed by three tasks: EID, EINF, and EMOD. Until the user presses [Enter] (or any other AID key), no task is executing and all system resources are available to other tasks in the DC system.



6.5 Performance considerations

Because of the interactive and simultaneous nature of online applications, your programs must do everything possible to ensure good system performance. Thus, good response time will be available to all online applications running under the DC system. A user should not have to wait more than a few seconds for each task to execute.

Efficiency factors: A well-written DC program provides fast response time to the user and efficient resource usage to the DC system. The following considerations are important factors in an efficient online application:

- **Save I/O** by accessing the database in an efficient manner; use database keys, CALC keys, and indexes whenever possible.
- **Minimize the data stream** transmitted on terminal input and output operations. Your program should send only the data necessary to perform a specified task. For example, you should transmit literals to the terminal screen only on the initial output operation and not on subsequent output operations.
- **Acquire storage dynamically** whenever possible by using the GET STORAGE command; release it as soon as possible by using the FREE STORAGE command. This provides efficient use of the storage available in the DC system.
- **Minimize internal queuing and resource contention** by acquiring the minimum amount of storage necessary to execute your task.

6.6 Error handling

Because online processing is interactive, your program can inform the user of erroneous data and require that a request be resubmitted. To avoid extra I/O, your program should perform extensive error checking. This ensures that the data input by the user is correct before attempting any database access. For example, you could check that monetary fields are numeric and are within the range 0.00 to 100,000.00.

Automatic editing: If a map has enabled automatic editing, the runtime mapping system can perform certain editing and error-handling functions.

►► For more information on automatic editing and error handling, refer to *CA-IDMS Mapping Facility*.

6.7 Using the IDMS communications block

The IDMS communications block is the interface between your program and DC. Whenever your program issues a call to DC, DC returns information about the outcome of the requested service to the IDMS communications block.

The data description (identified as SUBSCHEMA-CTRL) of the IDMS communications block is copied from the dictionary into program variable storage. When you submit the program to the precompiler, the IDMS-DC communications block is copied automatically unless you specify that records are to be copied manually. In that case, your program explicitly copies in the data description from the dictionary.

Assembler programmers: The IDMS communications block is not available in Assembler. Assembler programs should check the value returned to register 15 to determine the result of a DC call. For more information, see Appendix B, “Assembler Considerations” on page B-1.

IDMS communications block fields: You should take note of the following IDMS communications block fields:

- **PROGRAM-NAME** contains the name of the current program. It is a good programming practice to initialize this field at the beginning of every program.
- **ERROR-STATUS** contains a value that indicates the status of the last DML call.
- **DBKEY** contains the db-key of the last database record accessed by the program.
- **SSC-ERRSTAT-SAVE** contains a value that indicates the status of the DML call that caused the program to be terminated.
- **SSC-DMLSEQ-SAVE** contains the sequence number of the DML statement that caused the program to be terminated (only if the precompiler option DEBUG is in effect).
- **DML-SEQUENCE** contains the sequence number of the last DML statement executed by the program (only if the precompiler option DEBUG is in effect).

Checking the ERROR-STATUS field: Your program should examine the ERROR-STATUS field in the IDMS-DC communications block after *every* CA-IDMS DML call. COBOL and PL/I programs can check the ERROR-STATUS field by using the IDMS-STATUS routine, which can be copied in from the dictionary.

Performing IDMS-STATUS: Your program should perform the IDMS-STATUS routine after first checking for zeros and for any anticipated nonzero ERROR-STATUS values. Under DC, this routine checks the ERROR-STATUS field for zeros and, on finding a nonzero value, writes a memory dump of the IDMS communications block and terminates the program.

COBOL programmers: CA-IDMS COBOL includes the AUTOSTATUS protocol. AUTOSTATUS directs the precompiler to generate a PERFORM IDMS-STATUS statement after every DML statement. Since all examples are

in COBOL, PERFORM IDMS-STATUS is not coded in the sample programs, nor is it listed in the programming steps.

Under AUTOSTATUS, you can check for a nonzero status by including an **ON clause** at the end of a DML statement. If the specified status is returned, the imperative statement included in the ON clause is executed; otherwise, IDMS-STATUS is performed.

Because IDMS-STATUS is a COBOL SECTION, you should copy it into your program using at least one of the following methods in order to avoid runtime errors:

- Always PERFORM IDMS-STATUS THRU ISABEX.
- Start a new SECTION after IDMS-STATUS.
- Place IDMS-STATUS at the end of the program.

Example of performing IDMS-STATUS: The program excerpt below shows the IDMS-STATUS routine used in CA-IDMS COBOL programs.

This routine checks the ERROR-STATUS field in the IDMS-DC communications block for a value of zero (DB-STATUS-OK) to see whether the previously issued DML statement was executed successfully. If a nonzero value is returned, the routine snaps the subschema control block and abends the program.

```
*****
IDMS-STATUS                                SECTION.
***** IDMS-STATUS FOR CA-IDMS/DC *****
      IF DB-STATUS-OK GO TO ISABEX.
      PERFORM IDMS-ABORT.
      MOVE ERROR-STATUS TO SSC-ERRSTAT-SAVE.
      MOVE DML-SEQUENCE TO SSC-DMLSEQ-SAVE.
      SNAP FROM SUBSCHEMA-CTRL TO SUBSCHEMA-CTRL-END
          ON ANY-STATUS
          NEXT SENTENCE.
      ABEND CODE SSC-ERRSTAT-SAVE
          ON ANY-STATUS
          NEXT SENTENCE.
ISABEX. EXIT.
DMLC-DC-GEN-GOBACK SECTION.
GOBACK.
```

Chapter 7. Terminal Management

7.1 About this chapter	7-3
7.2 Mapping mode	7-4
7.2.1 Housekeeping	7-5
7.2.2 Displaying screen output	7-5
7.2.3 Reading screen input	7-8
7.2.4 Modifying map options	7-11
7.2.5 Writing and reading in one step	7-12
7.2.6 Suppressing map error messages	7-13
7.2.7 Testing for identical data	7-14
7.3 Using pageable maps	7-15
7.3.1 Pageable map format	7-15
7.3.2 Conducting a map paging session	7-17
7.3.3 How to code a browse application	7-20
7.3.4 How to code an update application	7-23
7.3.5 Overriding automatic mapout for pageable maps	7-27
7.4 Line mode	7-30
7.4.1 Beginning a line mode session	7-30
7.4.2 Writing a line of data	7-30
7.4.3 Reading a line of data	7-32
7.4.4 Ending a line mode session	7-33
7.4.5 3270-type considerations	7-33

7.1 About this chapter

DC terminal management functions enable your program to transfer data to and from the terminal. You can use one of the following modes to transfer data:

- **Mapping mode** transfers an entire screen of data on a field-by-field basis. Mapping mode can be used only with 3270-type devices and glass TTYs that have established device-independence tables.
- **Line mode** transfers data one line at a time.
- **Basic mode** transfers a variable amount of data, as specified in the program.

The table below compares the three types of terminal management.

Mode	Data transfer	Device-control characters	Line-control characters	Terminal devices
Mapping	Field-by-field	DC-built	DC-built	3270-type and glass TTYs
Line	One line at a time	DC-built	DC-built	Device independent
Basic	Data length specified in the program	Program	DC-built	Device dependent

►► For more information about basic mode, see 10.3, “Basic mode” on page 10-6.

7.2 Mapping mode

In mapping mode, your program communicates with 3270-type terminal devices. DC uses maps to associate screen positions on the terminal with fields in program variable storage.

Example of map data fields: The EMPDISPM map below associates row 4, column 24, with the EMP-ID-0415 field in variable storage; the map associates row 5, column 24, with the EMP-LAST-NAME-0415 field, and so on.

```
*** EMPLOYEE INFORMATION SCREEN ***

EMPLOYEE ID : _____
LAST NAME  : _____
FIRST NAME : _____
ADDRESS   : _____
           : _____
           : _____
DEPARTMENT : _____

ENTER AN EMPLOYEE ID AND PRESS ENTER *** PRESS CLEAR TO EXIT
```

Creating a map: To transfer data in mapping mode, you must first create a map by using either the online or batch compiler of the CA-IDMS mapping facility. You associate map variable fields with either database records or IDD-defined work records.

Maps are available as load modules to the DC runtime system. DC views map load modules as programs.

Mapping mode terminal management: Using mapping mode terminal management, you can perform the following functions:

- Write data to a terminal screen
- Read data input from a terminal screen and query the status of conditions related to the input operation
- Modify previously established map and map field options
- Write unlimited detail occurrences that can be displayed one page at a time by using a pageable map

Mapping terminology: You should understand the following terms related to maps:

- **Attribute byte** — The nondisplayable byte that begins each map field at runtime. The contents of the attribute byte determine the characteristics of the field (such as protection and intensity). Attributes bytes are a 3270 feature.
- **Automatic editing and error handling** — An optional map feature that can be used to perform editing and error-handling functions at runtime. These functions can compare input and output data with internal and external pictures, validate data against edit tables, and encode or decode data through code tables.
- **Modified data tag (MDT)** — The internal switch for a map data field that indicates whether the value in that field has been changed by the user. Modified data tags are a 3270 feature.
- **Write control character (WCC)** — The internal character that holds various specifications for the display of the map such as resetting the keyboard to allow user input. Write control characters are a 3270 feature.

►► For a complete description of maps and map attributes, refer to *CA-IDMS Mapping Facility*.

7.2.1 Housekeeping

To define the map to the precompiler at compile time, and to establish addressability to DC at runtime, you must perform certain mapping mode housekeeping functions:

- **Identify the map you want to use** by including a MAP SECTION (COBOL), a DECLARE MAP statement (PL/I), or the MAP parameter in the @INVOKE statement (Assembler).
- **Copy the map request block (MRB) and the map records** by including compiler-directive statements in program variable storage.
- **Establish addressability between DC and the MRB** by issuing a BIND MAP statement.
- **Establish addressability to map records** by issuing a BIND MAP RECORD statement for each record defined for the map.

►► For more information on mapping mode housekeeping statements, refer to the language-specific CA-IDMS DML reference manual.

7.2.2 Displaying screen output

To display a map on the terminal screen, perform the following steps:

1. Issue mapping mode housekeeping statements as described above.
2. Initialize variable-storage data fields as needed.
3. Transfer data from variable-storage data fields to map fields on the screen by issuing a MAP OUT statement.

You can also use the MAP OUT statement to transfer data between two variable-storage data fields; this is referred to as a *native mode data transfer*.

►► For more information about native mode data transfers, refer to the language-specific CA-IDMS DML reference manual.

►► Pageable maps have different output considerations. For more information, see 7.3, “Using pageable maps” on page 7-15 later in this chapter.

Mapping considerations: You need to know about the following considerations when writing a program that displays maps:

- Sending informational messages to the user
- Keeping the data stream short
- Choosing asynchronous or synchronous processing

Sending informational messages: You can send a variety of messages to the user's terminal, depending on the situation. For example, if the application is being accessed for the first time, you might transmit the following message:

```
ENTER AN EMPLOYEE ID AND PRESS ENTER **** PRESS CLEAR TO EXIT
```

You might send a different message with the same map at another time to indicate the completion status of a task:

```
**** SPECIFIED EMPLOYEE CANNOT BE FOUND ****
```

COBOL and PL/I programmers: To avoid unpredictable results at runtime, specify messages that are 100 bytes or less in length.

Keeping the data stream short: Because you want to promote the fastest possible response time, an important programming consideration is the length of the data stream transmitted to or from the terminal. You should ensure that your program always transmits the smallest amount of data necessary to successfully complete a mapping operation.

Ways to minimize the data stream include:

- **Avoid rewriting literals.** If you are rewriting to the same map, there usually is no need to retransmit literal fields. Specify the NEWPAGE and the LITERALS options only on an initial map output.
- **Transmit only the attribute bytes.** If your program determines that the user has entered invalid data, you need not retransmit the invalid values; these values are still listed on the terminal screen. Instead, you can specify OUTPUT DATA IS ATTRIBUTE to transmit only the attribute bytes for map fields.

The ATTRIBUTE specification is useful when sending error messages to the terminal because DC still transmits the data in the message field. For example, you could minimize the data stream transmitted by coding the following MAP OUT statement:

```
MAP OUT USING DEPTMAP
  OUTPUT DATA IS ATTRIBUTE
  MESSAGE IS ID-EDIT-ERROR-MESS TO ID-EDIT-ERROR-MESS-END.
```

If automatic editing and error handling are enabled and you use the **ERROR** option of the **MODIFY MAP** statement, the **ATTRIBUTE** specification is automatically invoked.

Synchronous and asynchronous processing: Mapping mode supports synchronous and asynchronous map output operations:

- During a **synchronous** map output request, DC places your task in an inactive state until processing is complete.

To issue a synchronous map output request, specify the **WAIT** option of the **MAP OUT** statement. This option allows you to ensure that the output request was completed successfully before continuing program processing.

- During an **asynchronous** map output request, DC returns control to your task before the output processing is complete. Before issuing subsequent map output requests, you must ensure that the first request is finished by issuing a **CHECK TERMINAL** request. **CHECK TERMINAL** is a basic mode DML statement that is described in 10.3, “Basic mode” on page 10-6.

To issue an asynchronous map output request, specify the **NOWAIT** option of the **MAP OUT** statement.

You may want to specify **NOWAIT** if your program issues a **MAP OUT** just before task termination. This causes DC to release a task's resources sooner. In this case, however, you cannot issue the **CHECK TERMINAL** statement; you won't be able to determine the completion status of the **MAP OUT** operation.

Example of an initial application screen: The program excerpt below displays an application's initial screen. It initializes the **EMP-ID-0415** field and displays the screen, soliciting user input.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 TSK02                                PIC X(8) VALUE 'TSK02'.
01 MESSAGES.
   05 INITIAL-MESSAGE                   PIC X(54) VALUE
   'ENTER AN EMPLOYEE ID AND PRESS ENTER *** CLEAR TO EXIT'.
   05 INITIAL-MESSAGE-END               PIC X.
PROCEDURE DIVISION.
*** ESTABLISH ADDRESSABILITY TO MAP ***
  BIND MAP SOLICIT.
*** ESTABLISH ADDRESSABILITY TO MAP RECORDS ***
  BIND MAP SOLICIT RECORD EMPLOYEE.
  BIND MAP SOLICIT RECORD DATE-WORK-REC.
  MOVE ZERO TO EMP-ID-0415.
*** DISPLAY THE MAP ***
  MAP OUT USING SOLICIT
  WAIT NEWPAGE
  MESSAGE IS INITIAL-MESSAGE TO INITIAL-MESSAGE-END.
*** RETURN CONTROL TO CA-IDMS/DC NEXT TASK TSK02 ***
  DC RETURN
  NEXT TASK CODE TSK02.
```

7.2.3 Reading screen input

When the user finishes inputting data and presses an AID key, DC invokes the specified input task. The task reads data from the screen and tests for certain input conditions.

To transfer data from map fields on the terminal screen to the corresponding variable storage data fields, perform the following steps:

1. Issue mapping mode housekeeping statements, as explained in 7.2.1, “Housekeeping” on page 7-5 earlier in this chapter.
2. Transfer data from map fields on the terminal screen to variable-storage data fields by issuing a MAP IN statement.

You can use the MAP IN statement to transfer data between two variable-storage data fields; this is referred to as a *native mode data transfer*.

►► For more information about native mode data transfers, refer to the language-specific CA-IDMS DML reference manual.

►► Pageable maps have different input considerations. For more information, see 7.3, “Using pageable maps” on page 7-15 later in this chapter.

Example of reading input: The program excerpt below reads data from the screen.

It transfers data from the terminal screen to map data fields in program variable storage by issuing a MAP IN statement.

```
PROCEDURE DIVISION.  
*** ESTABLISH ADDRESSABILITY TO THE MAP ***  
    BIND MAP SOLICIT.  
*** ESTABLISH ADDRESSABILITY TO THE MAP RECORDS ***  
    BIND MAP SOLICIT RECORD EMPLOYEE.  
    BIND MAP SOLICIT RECORD EMP-DATE-WORK-REC.  
*** TRANSFER DATA FROM MAP DATA FIELDS TO VARIABLE STORAGE ***  
    MAP IN USING SOLICIT.  
*** FURTHER PROCESSING OF ENTERED DATA ***
```

Testing for input conditions: After a MAP IN request, your program can inquire about conditions related to the input operation. For example, you may need to perform processing based on the AID key pressed by the user or determine if the user entered data in a particular map data field.

To test for conditions related to a map input operation, issue an INQUIRE MAP statement. By using this statement, you can obtain the following information:

- The control key pressed.
- The current cursor position.
- Information on conditions regarding a map data field or group of map data fields:

- Is data present?
- Has data been modified?
- Has data been truncated?
- What is the entered length of a specific map input field?
- Whether specified map fields are in error (the error flag has been set on for those fields) or are correct (the error flag has been set off). This option applies only to those maps and map fields for which automatic editing is enabled.
- Whether the screen was formatted before the input operation was performed.

Frequent uses of the INQUIRE MAP statement are listed below:

- **To determine what control key was pressed.** Typically, an application offers various processing options to the user. Each option can be associated with a control key. Your program should check the AID byte after every MAP IN statement to determine the option chosen. The table below lists the AID characters associated with each 3270-type control key.

Key	AID character
[Enter]	" ' " (single quote)
[Clear]	'_' (underscore)
[PF1]	'1'
[PF2]	'2'
[PF3]	'3'
[PF4]	'4'
[PF5]	'5'
[PF6]	'6'
[PF7]	'7'
[PF8]	'8'
[PF9]	'9'
[PF10]	':'
[PF11]	'#'
[PF12]	'@'
[PF13]	'A'
[PF14]	'B'
[PF15]	'C'
[PF16]	'D'
[PF17]	'E'
[PF18]	'F'
[PF19]	'G'
[PF20]	'H'
[PF21]	'I'
[PF22]	'¢'
[PF23]	':'
[PF24]	'<'
[PA1]	'%'
[PA2]	'>'
[PA3]	','

- **To ensure that necessary data has been entered.** You should make sure that the user has entered data in all fields necessary for successful processing.

- **To determine if automatic editing and error-handling have detected any errors.** If input errors are detected, DC automatically transmits only the attribute bytes for the next map output operation.

You can use the TASK CODE parameter of the ACCEPT statement to retrieve the calling task code.

►► For more information about the ACCEPT statement, see 9.3, “Retrieving task-related information” on page 9-9.

The program excerpt below performs processing based on conditions related to the last map input operation. It uses the INQUIRE MAP statement to determine what control key was pressed and to ensure that the DEPT-ID-0410 field contains data.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 DC-AID-CONDITION-NAMES.
   03 DC-AID-IND-V          PIC X.
                               88 ENTER-HIT VALUE QUOTE.
                               88 CLEAR-HIT VALUE '_'.

PROCEDURE DIVISION.
*** ESTABLISH ADDRESSABILITY TO THE MAP AND MAP RECORDS ***
   BIND MAP SOLICIT.
   BIND MAP SOLICIT RECORD EMPLOYEE.
   BIND MAP SOLICIT RECORD DEPARTMENT.
   BIND MAP SOLICIT RECORD EMP-DATE-WORK-REC.
*** TRANSFER DATA FROM THE MAP TO VARIABLE STORAGE ***
   MAP IN USING SOLICIT.
*** DETERMINE THE AID KEY PRESSED BY THE TERMINAL OPERATOR ***
   INQUIRE MAP SOLICIT
     MOVE AID TO DC-AID-IND-V.
*** IF OPERATOR PRESSED CLEAR THEN DC RETURN ***
   IF CLEAR-HIT DC RETURN.
*** DETERMINE IF THE TERMINAL OPERATOR ***
*** ENTERED DATA IN THE DEPT-ID-0410 FIELD ***
   INQUIRE MAP SOLICIT
     IF DFLD DEPT-ID-0410
       DATA IS NO
       GO TO A100-NO-DATA.
.
*** FURTHER PROCESSING OF ENTERED DATA ***

```

7.2.4 Modifying map options

Before issuing an input or output request, you may need to modify a map's WCC options or specify attributes for one or more map data fields. You can make modifications either for the length of the session or for the next mapping operation. For example, you may need to:

- Position the cursor on the next MAP OUT operation
- Require that the user enter data in a specified map data field
- Prevent the user from entering data in specified map data fields (this is especially useful on the initial MAP OUT of a session)

- Require that data from a specified map data field be transmitted regardless of whether it was modified by the user
- Modify the WCC and attribute options for an entire session

Steps to modify a map: To modify a map's WCC options or to specify attributes for one or more map data fields, perform the following steps:

1. Issue mapping mode housekeeping statements
 - ▶▶ For more information about housekeeping statements, see 7.2.1, “Housekeeping” on page 7-5 earlier in this chapter.
2. Issue the MODIFY MAP command
3. Issue either a MAP IN or MAP OUT statement

Example of modifying a map: The program excerpt below uses the MODIFY MAP statement to protect map data fields from operator input. The program is used in an application's initial MAP OUT to help ensure that the user will enter data in the correct field (EMP-ID-0415) by positioning the cursor and preventing input to all other map data fields.

```
PROCEDURE DIVISION.  
  BIND MAP SOLICIT.  
  BIND MAP SOLICIT RECORD EMPLOYEE.  
  BIND MAP SOLICIT RECORD EMP-DATE-WORK-REC.  
*** SET CURSOR AND PREVENT INPUT INTO ALL BUT EMP-ID-0415 ***  
  MODIFY MAP SOLICIT TEMPORARY  
    CURSOR AT DFLD EMP-ID-0415  
    FOR ALL EXCEPT DFLD EMP-ID-0415  
    ATTRIBUTES PROTECTED.  
*  
  MOVE ZERO TO EMP-ID-0415.  
  MAP OUT USING SOLICIT  
    YES NEWPAGE  
    MESSAGE IS INITIAL-MESSAGE TO INITIAL-MESSAGE-END.  
*  
  DC RETURN  
  NEXT TASK CODE TSK02.
```

7.2.5 Writing and reading in one step

To write data to the terminal and read data input from the terminal in one synchronous operation, issue a MAP OUTIN statement. **CAUTION:** **MAP OUTIN forces your program to be conversational; it is not recommended.**

If your application needs to write and read in one step, perform the following steps:

1. Issue mapping mode housekeeping statements
 - ▶▶ For more information about housekeeping statements, see 7.2.1, “Housekeeping” on page 7-5 earlier in this chapter.
2. Modify map or map data fields,
 - ▶▶ For more information about modifying map data fields, see 7.2.4, “Modifying map options” on page 7-11 earlier in this chapter.

3. Initialize variable-storage data fields as needed
4. Transfer data from variable-storage data fields to map fields on the terminal screen and back again by issuing the MAP OUTIN statement

7.2.6 Suppressing map error messages

You can suppress the display of error messages for map fields. For example, you can code a data validation test so that it suppresses a map field's default error message and displays a different message when the field is in error.

What to do: Include the ERROR MESSAGE IS ACTIVE/SUPPRESS parameter on your MODIFY MAP statement. ERROR MESSAGE immediately follows the REQUIRED/OPTIONAL parameter.

Example of suppressing error messages: This COBOL example issues a MODIFY MAP statement that suppresses the display of default error messages for the ORDER-AMOUNT field on the current map.

In this application, the data validation routine compares the ORDER-AMOUNT field with the number of widgets on hand. If the current stock restricts the size of ORDER-AMOUNT, an alternative message is displayed.

1. Define an alternative message in working storage. For example:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 MESSAGES.
   05 INITIAL-MESSAGE          PIC X(80) VALUE
      'ENTER A NUMERIC ORDER-AMOUNT AND PRESS ENTER'.
   05 EDIT-ERROR-MESSAGE      PIC X(80) VALUE
      'ORDER-AMOUNT EITHER NOT ENTERED OR NOT NUMERIC'.
   05 INVENTORY-MESSAGE       PIC X(80) VALUE
      'NOT ENOUGH WIDGETS IN STOCK TO DELIVER THAT AMOUNT'.
   05 DISPLAY-MESSAGE         PIC X(80) VALUE
      'CLEAR TO EXIT ** ENTER ORDER-AMOUNT AND ENTER TO CONTINUE'
```

2. Modify the map to display alternative messages when a specific error is found:

```
MODIFY MAP MAP01 TEMPORARY
FOR DFLD ORDER-AMOUNT
ERROR MESSAGE IS SUPPRESS.
```

3. Perform your data validation routine. For example, you can compare the number of widgets in stock to ORDER-AMOUNT. If ORDER-AMOUNT is greater than the number in stock, issue an alternative message indicating that the order cannot be filled.

If the data validation routine indicates that there are not enough widgets in stock, display the map with the alternative message.

TEMPORARY and PERMANENT options: The use of the SUPPRESS option is affected by the TEMPORARY/PERMANENT option:

- If TEMPORARY is specified, error messages are suppressed for the next mapout only.

- If `PERMANENT` is specified, error messages are suppressed until the program terminates or until the error message specifications are overridden by a subsequent `MODIFY MAP` statement.

7.2.7 Testing for identical data

You can compare the contents of a mapped-in field with the map data that is currently in your program's record buffer.

This means that you can test whether a map field contains the same data that was previously mapped out. By comparing the fields, your program updates the database only when the user enters different data, reducing the number of database I/O operations.

How this relates to MDT settings: The input test condition does not test a field's modified data tag (MDT). For example, the statement `INQUIRE MAP MAP01 DATA IS IDENTICAL` is *true* in either of the following cases:

- The field's MDT is off. On mapin, the MDT is usually off if the user did not type any characters in the field.
- The field's MDT is on, but each character that the user typed in is identical (including capitalization) to the data in variable storage.

What to do: Include the `IDENTICAL/DIFFERENT` parameter in your `INQUIRE MAP` statement.

Example of testing for identical data: This COBOL example uses an `INQUIRE MAP` statement to test whether the user has entered an employee ID number:

- If the `IDENTICAL` condition is *true* (the user doesn't specify a different ID number), the program displays the menu screen
- If the `IDENTICAL` condition is *false* (the user specifies a different ID number), the program obtains the corresponding employee record from the database

The sample `INQUIRE MAP` statement is shown below:

```
INQUIRE MAP MAP01
  IF DFLD EMP-ID-0415 DATA IS IDENTICAL THEN
    PERFORM EMP-PROMPT-20
  ELSE
    PERFORM EMP-OBTAIN-20.
```

Example of testing for changed data: This COBOL example uses an `INQUIRE MAP` statement to test whether the user has entered a new department ID or department name. If the user has changed either value (`DIFFERENT` is *true*), the program branches to `DEPTUP-30`.

```
INQUIRE MAP MAP02
  IF ANY DFLD DEPT-ID-0410
    DFLD DEPT-NAME-0410 DATA IS DIFFERENT
  THEN PERFORM DEPTUP-30.
```

7.3 Using pageable maps

A pageable map can contain more occurrences of a set of map fields than can fit on the screen at one time; therefore, it can contain unlimited occurrences of the set of map fields. Each occurrence of the multiply-occurring set is called a **detail occurrence**. The MAP OUT and MAP IN statements can create, retrieve, and modify detail occurrences of a pageable map.

About pageable maps: You should know about the following aspects of pageable maps:

- The format of a pageable map
- How to conduct a map paging session
- How to code an application that allows the user to browse through a pageable map but not update it
- How to code an update application that allows the user to perform database updates by using a pageable map

7.3.1 Pageable map format

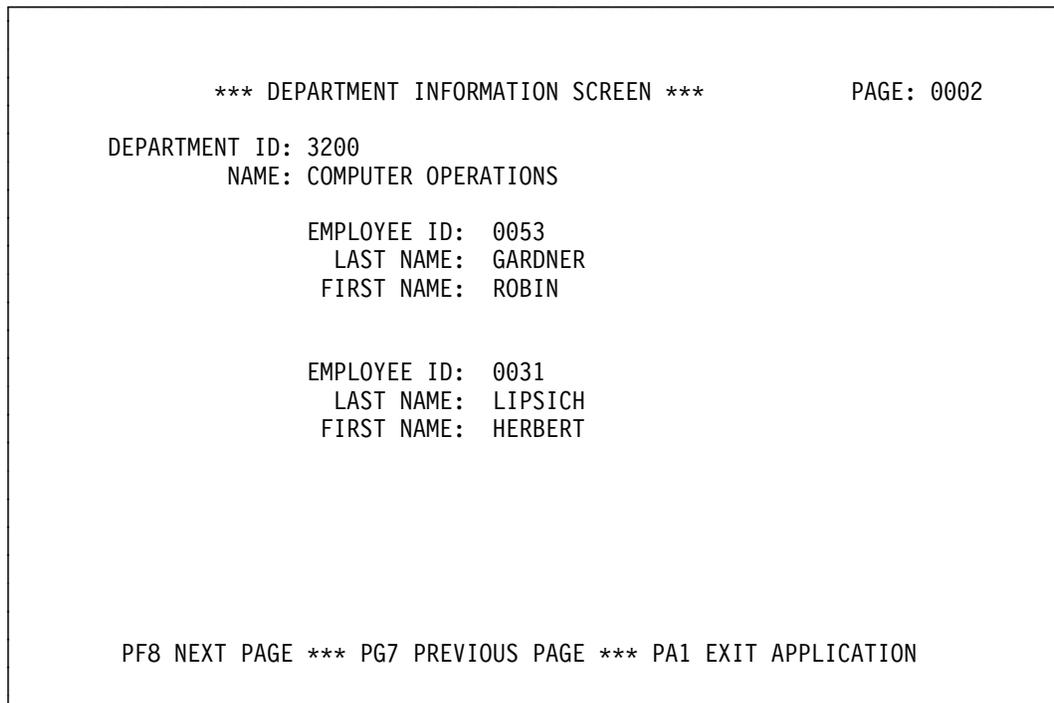
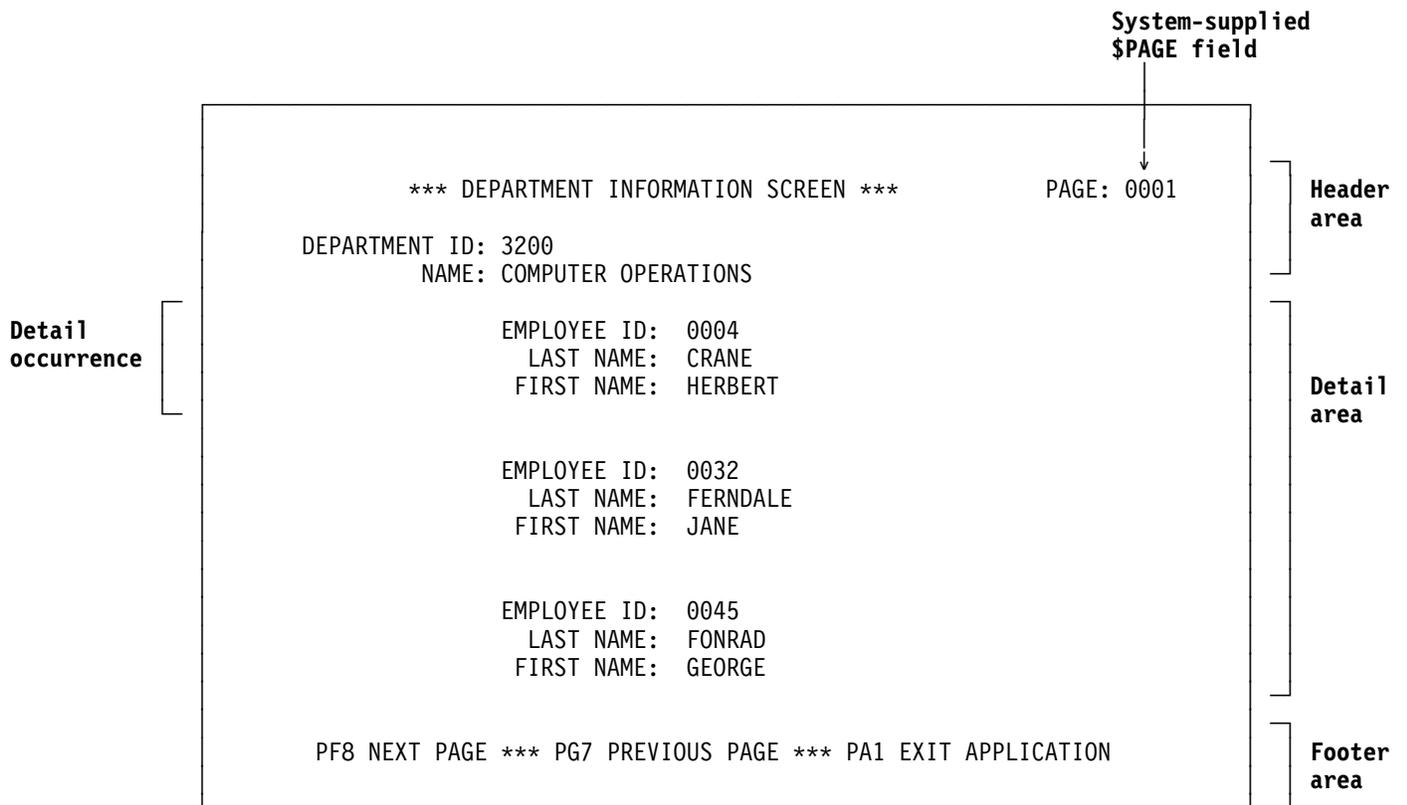
A pageable map is divided into the **header area**, the **detail area**, and the **footer area**. The header and footer areas consist of general information such as the map title, the page number, or the useable PF-keys. The detail area consists of detail occurrences.

►► For information on defining a pageable map, refer to *CA-IDMS Mapping Facility*.

CAUTION:

To prevent excessive database record locking, you should not define database records as map records in a pageable map; use IDD-defined work records instead.

Example of a pageable map: The figures below illustrate two pages of a map screen. Note that the display of information in the header and footer areas is unchanged except for the \$PAGE field.



7.3.2 Conducting a map paging session

A map paging session involves interaction among the user, the runtime mapping system, and your map paging application program. You should understand this interaction and the sequence of events that occurs during a map paging session before planning the logic of your application program.

Typical map paging sequence: This sequence of events typically occurs during a map paging session:

1. Your program begins the session and defines map paging parameters
2. The program creates detail occurrences
3. A map page is displayed on the terminal
4. The user pages forward and backward through the pageable map
5. The user optionally modifies map data fields
6. The program receives control and updates the database
7. The user ends the map paging session

The following discussion describes each step in detail.

Beginning the paging session: A map paging session begins when your program issues a `STARTPAGE` statement. Options included in this statement specify the following:

- **The runtime flow of control.** The paging type (`NOWAIT/WAIT/RETURN`) determines whether the runtime mapping system or your program receives control when the user presses a control key, as detailed in the table below.

The paging type affects the frequency with which your program will receive control and the processing logic you must provide. For example, in `NOWAIT`, runtime mapping performs all paging operations for you; in `WAIT` and `RETURN`, you must provide coding logic that performs the paging operations specified by the user.

`NOWAIT` is best for applications in which the user can display but not update; `WAIT` and `RETURN` are best for update applications.

- **Whether the user can display a previous map page.** If backpaging is allowed, the runtime system must maintain the resources that describe the detail occurrences of previous pages. If backpaging is not allowed, the runtime system deletes all previous pages of detail occurrences when a new map page is displayed.

Note: Always allow backpaging for pageable map applications that perform database updates.

- **Whether the user can update map data fields.** A paging mode of `UPDATE` specifies that the user can modify map data fields, subject to restrictions specified

in the map and by previous MODIFY MAP statements. BROWSE specifies that the user can modify only the system-supplied \$PAGE field (if present).

The tables below summarize flow of control in a map paging session.

■ Paging request*:

Paging type	No data fields modified	Data fields modified**
NOWAIT	Runtime mapping displays the requested map page	Runtime mapping displays the requested map page
WAIT	Runtime mapping displays the requested map page	Control passes to the program
RETURN	Control passes to the program	Control passes to the program

* If the user presses [Clear], [PA1], [PA2], or [PA3], and that key is not associated with backward or forward paging, refer instead to "Nonpaging request" below.

** If the user presses [Clear], [PA1], [PA2], or [PA3], refer to the "No data fields modified" column.

■ Nonpaging request:

Paging type	No data fields modified	Data fields modified**
NOWAIT	Control passes to the program	Runtime mapping redisplay the same map page
WAIT	Control passes to the program	Control passes to the program
RETURN	Control passes to the program	Control passes to the program

** If the user presses [Clear], [PA1], [PA2], or [PA3], refer to the "No data fields modified" column.

Creating detail occurrences Your program retrieves data, moves it to map data fields, and creates detail occurrences by issuing MAP OUT DETAIL commands.

Displaying the first page: The first page is displayed on the terminal screen in one of the following ways:

- **Runtime mapping** automatically displays the first map page when the first detail occurrence of the second page of occurrences is created. The program continues to execute and create additional detail occurrences.

When the first page is displayed by runtime mapping, DC returns a status of 4676 (DC-FIRST-PAGE-SENT). Your program must check for this status after every MAP OUT DETAIL statement.

- **Your program displays the first map page.** When all detail occurrences are created, your program should check to determine if the first page was written to the terminal. You do this by setting a switch when DC returns a status of 4676 (DC-FIRST-PAGE-SENT). If 4676 was never returned, your program explicitly displays the first map page by issuing a MAP OUT RESUME statement.

Paging forward and backward: To specify the next map page to be displayed, the user does one of the following:

- Presses the control key associated with paging forward one page
- Presses the control key associated with paging backward one page
- Changes the \$PAGE map field, if defined on the map, and presses a control key other than [Clear], [PA1], [PA2], or [PA3].

Modifying map fields: The user can change map data fields, including header and footer data fields, subject to restrictions specified by the STARTPAGE command (UPDATE/BROWSE) or by a previously specified MODIFY MAP command.

Updating the database: If the user has modified any map data fields or if the paging type is RETURN, the program reads modified detail occurrences and updates the database.

A modified detail occurrence contains one or more map fields whose modified data tags (MDTs) are set on.

To retrieve a modified detail occurrence, issue a MAP IN DETAIL statement. MAP IN DETAIL can retrieve a modified detail occurrence sequentially, by the order of detail occurrences, or randomly by a key value that can be associated with an occurrence. If sequential or random retrieval cannot retrieve a modified detail occurrence, DC returns a status of 4668 (DC-NO-MORE-UPD-DETAILS).

If you need to modify the current detail occurrence (for example, to send an error message), issue a MAP OUT DETAIL CURRENT statement. This statement modifies the detail occurrence most recently referenced by a MAP IN DETAIL or MAP OUT DETAIL statement.

After processing all modified detail occurrences, write the map to the terminal screen by issuing a MAP OUT RESUME statement. If WAIT or RESUME has been specified, your program is responsible for displaying the next page specified by the user.

If you need to create additional detail occurrences, you can do so at any time by issuing further MAP OUT DETAIL statements. The new occurrences are stored at the end of the set of detail occurrences.

Ending the paging session: When a map paging session ends, the system deletes all the detail occurrences created during the session. To end a session, issue an ENDPAGE SESSION command.

7.3.3 How to code a browse application

To write a pageable map application that allows the user to display data but not update it, perform the following steps:

1. Establish a switch in variable storage. This switch should be set on if runtime mapping has transmitted the first page.
2. Issue mapping mode housekeeping statements, as explained in 7.2.1, “Housekeeping” on page 7-5 earlier in this chapter.
3. Initiate the map paging session by issuing a STARTPAGE statement that specifies NOWAIT and BROWSE.
4. Initialize header data fields.
5. Perform the following steps iteratively until all data is retrieved:
 - a. Perform database retrieval and move data to map data fields in variable storage.
 - b. Issue a MAP OUT DETAIL NEW statement, checking for a status of 4676 (DC-FIRST-PAGE-SENT).
 - c. Set the first-page switch if 4676 is returned; perform the IDMS-STATUS routine if 4676 is not returned.

Ending the browse session: If, after all detail occurrences have been created, the first-page switch is not set, you should transmit the map page to the terminal screen by issuing a MAP OUT RESUME statement.

The next task specified in the DC RETURN NEXT TASK CODE statement should include logic that tests to see if the user has indicated the end of the map paging session. If so, issue an ENDPAGE SESSION statement.

Example of a browse application: The program excerpt below shows a pageable map application in which runtime mapping handles all paging requests (paging type of NOWAIT) and the operator cannot make updates (paging mode of BROWSE).

After acquiring the data passed from a previous task and establishing that database records are present, this program issues MAP OUT DETAIL statements iteratively until all detail occurrences are written. DEPTEND, which is specified as the next task, ends the paging session with the ENDPAGE command and performs processing based on the control key pressed.

```

DATA DIVISION
01 FIRST-PAGE-SW          PIC X VALUE 'N'.
   88 LESS-THAN-A-PAGE    VALUE 'N'.
01 MAP-WORK-REC.
   05 WORK-FIRST          PIC X(10).
   05 WORK-LAST           PIC X(15).
   05 WORK-EMP-ID         PIC X(4).
LINKAGE SECTION.
01 PASS-DEPT-INFO.
   05 PASS-DEPT-ID        PIC 9(4).
   05 PASS-DEPT-INFO-END  PIC X.

PROCEDURE DIVISION.
  BIND MAP DCTEST01.
  BIND MAP DCTEST01 RECORD MAP-WORK-REC.
*** ACQUIRE DEPT-ID FROM ERROR CHECKING PROGRAM ***
  GET STORAGE FOR PASS-DEPT-INFO TO
    PASS-DEPT-INFO-END
  WAIT SHORT USER
  STGID 'RKNS'.

*
  MOVE PASS-DEPT-ID TO DEPT-ID-0410.
  FREE STORAGE STGID 'RKNS'.

*
  COPY IDMS SUBSCHEMA-BINDS.
  READY USAGE-MODE RETRIEVAL.

*
  OBTAIN CALC DEPARTMENT
  ON DB-REC-NOT-FOUND GO TO NO-DEPT-ERR.
  IF DEPT-EMPLOYEE IS EMPTY
  GO TO NO-EMP-ERR.
*** BEGIN MAP PAGING SESSION ***
  STARTPAGE SESSION DCTEST01 NOWAIT BACKPAGE BROWSE.
  PERFORM A100-GET-EMPLOYEES THRU A100-EXIT
  UNTIL DB-END-OF-SET.
  FINISH.
*** IF FIRST PAGE NOT YET SENT, MAP OUT RESUME ***
  IF LESS-THAN-A-PAGE
  MAP OUT USING DCTEST01 RESUME.
*** NEXT TASK ENDS PAGING SESSION ***
  DC RETURN NEXT TASK CODE 'DEPTEND'.

A100-GET-EMPLOYEES.
  OBTAIN NEXT WITHIN DEPT-EMPLOYEE
  ON DB-END-OF-SET GO TO A100-EXIT.
  MOVE EMP-FIRST-NAME-0415 TO WORK-FIRST.
  MOVE EMP-LAST-NAME-0415 TO WORK-LAST.
  MOVE EMP-ID-0415 TO WORK-EMP-ID.
*** MAP OUT CURRENT DETAIL, CHECK FOR ERROR-STATUS OF 4676 ***
  MAP OUT USING DCTEST01
  DETAIL NEW
  ON DC-FIRST-PAGE-SENT
  MOVE 'Y' TO FIRST-PAGE-SW.
A100-EXIT.
  EXIT.
.
.
.
*** FURTHER PROCESSING, INCLUDING ERROR ROUTINES ***

```

7.3.4 How to code an update application

To write a pageable map application that allows the user to update map data fields, establish a retrieval program and an update program.

Retrieval program: The retrieval program initiates the pageable map update session and retrieves and displays the data. This program can be similar to the one displayed in 7.3.3, “How to code a browse application” on page 7-20 earlier in this chapter. You should make the following changes to the retrieval program:

- Specify one of the following options of the STARTPAGE statement:
 - **WAIT** causes your program to acquire control after every *update* paging request and after every nonpaging request.
 - **RETURN** causes your program to acquire control after *every* paging request (update or nonupdate) and every nonpaging request.

Note: Because of editing and error-handling considerations, updating pageable maps by using a paging type of NOWAIT is not recommended.

- Use the KEY IS parameter of the MAP OUT DETAIL statement to pass the db-key of each retrieved record:

```
MAP OUT USING DCTEST01
  DETAIL NEW
  KEY IS DBKEY.
```

Including the db-key in this manner allows for DB-KEY retrieval in subsequent tasks.

- Code a DC RETURN statement that indicates the pageable map update program to be invoked when the user presses a control key.

Update program: The update program retrieves modified detail occurrences and updates the database. In this program, perform the following steps:

1. Establish a switch in variable storage. This switch should be set on if your program encounters any invalid data in modified detail occurrences.
2. Issue mapping mode housekeeping statements, as explained in 7.2.1, “Housekeeping” on page 7-5 earlier in this chapter.
3. Issue a MAP IN HEADER statement that includes the PAGE option. You can use the PAGE value later in your program when determining the next page to map out.
4. Issue an INQUIRE MAP statement to determine what control key was pressed. The control key pressed by the user can specify:
 - **The flow of control.** You can associate certain control keys with specific functions (for example, [Clear] might always exit the application).
 - **The next page to be displayed.** The user can indicate the next page to be displayed by pressing a site-standard paging control key.

- **A user error.** If the user presses an invalid control key, you should redisplay the current page.
5. Perform the following steps iteratively until all modified detail occurrences have been mapped in:
 - a. Issue a MAP IN DETAIL statement that includes the RETURNKEY parameter.
 - b. Check for a status of 4668 (DC-NO-MORE-UPD-DETAILS). If 4668 is returned, all updated details have been returned and you should display the pageable map, as specified by the user. If 4668 is not returned, perform the IDMS-STATUS routine.
 - c. Perform error and range checking to ensure that the user entered valid data. If invalid data is found, set the error switch and issue a MAP OUT DETAIL CURRENT statement that includes a message that indicates the error.
 - d. Perform database retrieval to access the database record to be modified. Retrieve the record by using its db-key (acquired from the RETURNKEY parameter). If data cannot be retrieved, set the error switch and issue a MAP OUT DETAIL CURRENT statement that includes a message that indicates the error.
 - e. Move data from the work record to the database record.
 - f. Issue database modification statements.
 - g. After all modified detail occurrences have been successfully processed, issue a MAP OUT RESUME statement that specifies the page requested by the user. If errors were encountered in the MAP IN DETAIL processing, you should redisplay the current page so the operator can correct the invalid data.

Ending the update session: The next task specified in the DC RETURN NEXT TASK CODE statement should include logic that tests to see if the user has indicated the end of the map paging session. If so, issue an ENDPAGE SESSION statement.

Example of an update application: The program excerpt below shows a pageable map update application. The program contains paging logic that works with a paging type of either WAIT or RETURN.

After determining user specifications, the program issues MAP IN DETAIL statements iteratively, modifying the database as specified, until all modified detail occurrences are processed.

```

DATA DIVISION
WORKING-STORAGE SECTION.
01 RETURN-DBKEY          PIC S9(8) COMP.
01 DEPTMOD               PIC X(8) VALUE 'DEPTMOD'.
01 FIRST-PAGE-SW        PIC X   VALUE 'N'.
   88 LESS-THAN-A-PAGE  VALUE 'N'.
01 MAP-IN-ERR-SW        PIC X   VALUE 'N'.
   88 MAP-IN-ERR        VALUE 'Y'.
01 PAGE-INDICATOR.
   05 SPEC-PAGE          PIC S9(8) COMP.
01 MESSAGES.
   05 EDIT-ERR-MESS      PIC X(21)
     VALUE 'CORRECT INVALID INPUT'.
   05 EDIT-ERR-MESS-END  PIC X.
*
   05 EMP-NOT-FOUND-MESS PIC X(18)
     VALUE 'EMPLOYEE NOT FOUND'.
   05 EMP-NOT-FOUND-MESS-END PIC X.
01 DC-AID-CONDITION-NAMES.
   03 DC-AID-IND-V      PIC X.
   88 ENTER-HIT VALUE QUOTE.
   88 CLEAR-HIT VALUE ' '.
   88 PF01-HIT VALUE '1'.
   88 PF02-HIT VALUE '2'.
   88 PF03-HIT VALUE '3'.
   88 PF04-HIT VALUE '4'.
   88 PF05-HIT VALUE '5'.
   88 PF06-HIT VALUE '6'.
   88 PF07-HIT VALUE '7'.
   88 PF08-HIT VALUE '8'.
   88 PF09-HIT VALUE '9'.
   88 PF10-HIT VALUE ':'.
   88 PF11-HIT VALUE '#'.
   88 PF12-HIT VALUE '@'.
   88 PF13-HIT VALUE 'A'.
   88 PF14-HIT VALUE 'B'.
   88 PF15-HIT VALUE 'C'.
   88 PF16-HIT VALUE 'D'.
   88 PF17-HIT VALUE 'E'.
   88 PF18-HIT VALUE 'F'.
   88 PF19-HIT VALUE 'G'.
   88 PF20-HIT VALUE 'H'.
   88 PF21-HIT VALUE 'I'.
   88 PF22-HIT VALUE '•'.
   88 PF23-HIT VALUE '.'.
   88 PF24-HIT VALUE '>'.
   88 PA01-HIT VALUE '%'.
   88 PA02-HIT VALUE '<'.
   88 PA03-HIT VALUE ','.
   88 PEN-ATTN-SPACE-NULL VALUE '='.
   88 PEN-ATTN VALUE QUOTE.
01 MAP-WORK-REC.
   05 WORK-EMP-ID        PIC X(4).
   05 WORK-FIRST        PIC X(10).
   05 WORK-LAST         PIC X(15).

```

```
PROCEDURE DIVISION.  
  BIND MAP DCTEST01.  
  BIND MAP DCTEST01 RECORD MAP-WORK-REC.  
  MOVE 'N' TO MAP-IN-ERR-SW.  
  *** MAP IN HEADER AND PAGE FIELD ***  
  MAP IN USING DCTEST01  
  HEADER  
  PAGE IS SPEC-PAGE  
  ON DC-DETAIL-NOT-FOUND  
  NEXT SENTENCE.  
  *** DETERMINE THE PF-KEY PRESSED ***  
  INQUIRE MAP DCTEST01 MOVE AID TO DC-AID-IND-V.  
  IF PA01-HIT  
  ENDPAGE  
  DC RETURN.  
  *** CHECK FOR HEADER ERRORS, MAP OUT IF ANY ARE FOUND ***  
  INQUIRE MAP DCTEST01  
  IF ANY EDIT IS ERROR  
  THEN  
    MODIFY MAP DCTEST01 TEMPORARY  
    FOR ALL ERROR FIELDS  
    ATTRIBUTES BRIGHT  
    MAP OUT USING DCTEST01 RESUME  
    DC RETURN NEXT TASK CODE DEPTMOD.  
  *  
  COPY IDMS SUBSCHEMA-BINDS.  
  READY ORG-DEMO-REGION USAGE-MODE IS UPDATE.  
  READY EMP-DEMO-REGION USAGE-MODE IS UPDATE.  
  *  
  PERFORM A100-MAP-IN-DETAILS THRU A100-EXIT  
  UNTIL DC-NO-MORE-UPD-DETAILS.  
  FINISH.  
  *** PAGING ROUTINES FOLLOW ***  
  *** IF ERROR SWITCH IS SET, REDISPLAY CURRENT PAGE ***  
  IF MAP-IN-ERR  
  THEN  
    MAP OUT USING DCTEST01  
    RESUME PAGE IS CURRENT  
    DC RETURN NEXT TASK CODE DEPTMOD.  
  *** IF PF07, DISPLAY PRIOR PAGE ***  
  IF PF07-HIT  
  THEN  
    MAP OUT USING DCTEST01  
    RESUME PAGE IS PRIOR  
    DC RETURN NEXT TASK CODE DEPTMOD.
```

```

*** IF PF08, DISPLAY NEXT PAGE ***
  IF PF08-HIT
  THEN
    MAP OUT USING DCTEST01
    RESUME PAGE IS NEXT
    DC RETURN NEXT TASK CODE DEPTMOD.
*** ELSE, USE PAGE VALUE FROM MAP IN HEADER ***
  MAP OUT USING DCTEST01
  RESUME PAGE IS SPEC-PAGE.
  DC RETURN NEXT TASK CODE DEPTMOD.
A100-MAP-IN-DETAILS.
*** MAP IN EACH MODIFIED DETAIL. EXIT ***
*** WHEN NO MORE MODIFIED DETAILS REMAIN ***
  MAP IN USING DCTEST01
  DETAIL
  RETURNKEY IS RETURN-DBKEY
  ON DC-NO-MORE-UPD-DETAILS GO TO A100-EXIT.
*** IF ERROR, MAP OUT DETAIL WITH MESSAGE, SET SWITCH ***
  INQUIRE MAP DCTEST01
  IF ANY EDIT IS ERROR
  THEN
    MODIFY MAP DCTEST01 TEMPORARY
    FOR ALL ERROR FIELDS
    ATTRIBUTES BRIGHT
    MAP OUT USING DCTEST01
    MESSAGE IS EDIT-ERR-MESS
    TO EDIT-ERR-MESS-END
    DETAIL CURRENT
    KEY IS RETURN-DBKEY
    MOVE 'Y' TO MAP-IN-ERR-SW
    GO TO A100-EXIT.
*** RETRIEVE EMPLOYEE, USING DBKEY FROM RETURNKEY ***
  OBTAIN EMPLOYEE DB-KEY IS RETURN-DBKEY
  ON ANY-STATUS NEXT SENTENCE.
*** IF ERROR, MAP OUT DETAIL WITH MESSAGE, SET SWITCH ***
  IF DB-REC-NOT-FOUND
  MAP OUT USING DCTEST01
  MESSAGE IS EMP-NOT-FOUND-MESS
  TO EMP-NOT-FOUND-MESS-END
  DETAIL CURRENT
  KEY IS RETURN-DBKEY
  MOVE 'Y' TO MAP-IN-ERR-SW
  GO TO A100-EXIT
  ELSE
    PERFORM IDMS-STATUS.
*
  MOVE WORK-FIRST TO EMP-FIRST-NAME-0415.
  MOVE WORK-LAST TO EMP-LAST-NAME-0415.
  MOVE WORK-EMP-ID TO EMP-ID-0415.
  MODIFY EMPLOYEE.
A100-EXIT.
EXIT.

```

7.3.5 Overriding automatic mapout for pageable maps

You can override the automatic mapout of a pageable map's first page.

By default, the first page of a pageable map is displayed as soon as the first detail occurrence of the second map page is written to scratch.

You can override this automatic mapout by specifying `NOAUTODISPLAY` in your `STARTPAGE` statement. By overriding the automatic display of the map's first page, you can add messages or modify the map before the page is displayed.

Return code for map page built: A map paging return code tells you before mapout whether a map page has been built.

The table below lists the map paging return code for map page built in COBOL, PL/I, and Assembler.

Language	Return code	Description
COBOL and PL/I	4680	<ul style="list-style-type: none"> ■ Returned in: IDMS communications block status code field ■ Returned after: <code>MAP OUT DETAIL</code> statement for a pageable map ■ Represented by the COBOL 88-level status code <code>DC-PAGE-READY</code>.
Assembler	X'50'	<ul style="list-style-type: none"> ■ Returned in: DC/UCF runtime register 15 ■ Returned after: <code>#MREQ OUT DETAIL=YES</code> statement for a pageable map

How to code a noautosave application: To code a pageable map application that does not automatically mapout when the first map page is built, perform the following steps:

1. Issue mapping mode housekeeping statements, as explained in 7.2.1, "Housekeeping" on page 7-5.
2. Initiate a map paging session by issuing a `STARTPAGE` statement that specifies `NOAUTODISPLAY`.


```
STARTPAGE SESSION MAP01 NOAUTODISPLAY
```
3. Initialize header data fields.

Map out detail occurrences: Perform the following steps iteratively until all data is retrieved:

1. Perform database retrieval and move data to map data fields in variable storage.
2. Issue a `MAP OUT DETAIL` statement. After each pageable map statement that writes a detail occurrence, test for `DC-PAGE-READY` to determine whether a map page has been built.


```
MAP OUT USING MAP01 OUTPUT DATA IS YES
  DETAIL NEW
    ON DC-PAGE-READY PERFORM FIRST-PAGE THRU FIRST-PAGE-XIT.
  .
  .
```
3. If you do find `DC-PAGE-READY`, you can optionally:
 - Modify the map.
 - Define messages to display on mapout.

If you do not find DC-PAGE-READY, perform the IDMS-STATUS routine.

4. Manually map out the first page:

```
FIRST-PAGE.
  MAP OUT USING MAP01 OUTPUT DATA IS YES
  RESUME PAGE FIRST
```

If you never find DC-PAGE-READY: If, after all detail occurrences have been created, you have not received a DC-PAGE-READY status code, you should transmit the map page to the terminal screen by issuing a MAP OUT RESUME statement.

Ending the paging session: The next task specified in the DC RETURN NEXT TASK CODE statement should include logic to test whether the user has indicated the end of the map paging session. If so, issue an ENDPAGE SESSION statement.

Example of suppressing automatic mapout: The following application does not automatically display the first page after it has been built.

```
OBTAIN CALC DEPARTMENT
  ON DB-REC-NOT-FOUND GO TO NO-DEPT.
  IF DEPT-EMPLOYEE IS EMPTY
    GO TO NO-EMP.
  MOVE DEPT-ID-0410 TO WORK-DEPT-ID.
  STARTPAGE SESSION DCTEST01
  NOWAIT
  BACKPAGE
  BROWSE
  NOAUTODISPLAY.
  PERFORM A100-GET-EMPLOYEES THRU A100-EXIT
    UNTIL DB-END-OF-SET.
  FINISH.
  IF LESS-THAN-A-PAGE
    MAP OUT USING DCTEST01 RESUME.
  DC RETURN NEXT TASK CODE 'DEPTEND'.
A100-GET-EMPLOYEES.
  OBTAIN NEXT WITHIN DEPT-EMPLOYEE
    ON DB-END-OF-SET GO TO A100-EXIT.
  MOVE EMP-ID-0415 TO WORK-EMP-ID.
  MOVE EMP-LAST-NAME-0415 TO WORK-LAST.
  MOVE EMP-FIRST-NAME-0415 TO WORK-FIRST.
  MAP OUT USING DCTEST01 OUTPUT DATA IS YES
    DETAIL NEW
  ON ANY-STATUS
    NEXT SENTENCE.
  IF DC-PAGE-READY
    PERFORM A100-FIRST-PAGE THRU
      A100-FIRST-PAGE-EXIT
    ELSE PERFORM IDMS-STATUS.
A100-EXIT.
  EXIT.
A100-FIRST-PAGE.
  MOVE 'Y' TO FIRST-PAGE-SW.
  IF ALREADY-MAPPED-OUT
    GO TO A100-FIRST-PAGE-EXIT
  ELSE
    MOVE EMP-MESSAGE-01 TO MESSAGE-01
    MAP OUT USING DCTEST01 OUTPUT DATA IS YES
    RESUME PAGE FIRST
A100-FIRST-PAGE-EXIT.
  EXIT.
```

7.4 Line mode

Line mode supports line-by-line transfers of data to and from a terminal buffer. Line mode transfers are recommended for programs requiring a simple transfer of unformatted data, independent of terminal type. Line mode supports synchronous read and write operations and asynchronous write operations.

Note: While a line mode I/O session is in progress, only line-mode requests can be issued; basic mode and mapping mode requests can cause unpredictable results.

By using line mode terminal management statements, you can:

- Initiate a line mode I/O session
- Write a line of data
- Read a line of data from the terminal screen
- End a line mode session

7.4.1 Beginning a line mode session

You initiate a line mode I/O session by issuing either of the following line mode DML statements:

- WRITE LINE TO TERMINAL
- READ LINE FROM TERMINAL

7.4.2 Writing a line of data

To transfer data from program variable storage to the screen, issue a WRITE LINE TO TERMINAL statement. DC automatically inserts the appropriate device control characters.

Transmission of data stream: WRITE LINE TO TERMINAL transmits a data stream to the terminal, as follows:

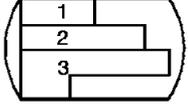
- For **line-by-line devices**, DC writes each line to the terminal immediately after the program issues the WRITE LINE TO TERMINAL request. New lines are added to lines already on the screen until the screen becomes full or the program requests DC to begin a new page.
- For **3270-type devices**, DC collects the number of output lines in buffers (or pages) that correspond to the terminal model in use. Data is written to the screen when:
 - The buffer becomes full
 - A READ LINE FROM TERMINAL request is issued
 - A WRITE LINE FROM TERMINAL request that specifies the NEWPAGE option is issued

- The issuing task terminates

Formatting the line: With either device type, data passed with each WRITE LINE TO TERMINAL request begins in the first character position of the next available line on the screen. If the length of the data exceeds the width of the screen, DC automatically reformats data into lines of the appropriate width.

Example of WRITE LINE TO TERMINAL: The figure below shows the processing associated with WRITE LINE TO TERMINAL requests for 3270-type terminals.

When the program issues the WRITE LINE TO TERMINAL NEWPAGE request, DC writes all buffered lines to the terminal. Because the data in line 3 exceeds the width of the screen, it is displayed as two lines.

Sequence of WRITE LINE request	Request	Buffer contents	3270-type terminal screen						
1	WRITE LINE TO TERMINAL.	<table border="1"> <tr> <td>1</td> <td></td> </tr> <tr> <td colspan="2"> </td> </tr> </table>	1						
1									
2	WRITE LINE TO TERMINAL.	<table border="1"> <tr> <td>1</td> <td>2</td> </tr> <tr> <td colspan="2"> </td> </tr> </table>	1	2					
1	2								
3	WRITE LINE TO TERMINAL.	<table border="1"> <tr> <td>1</td> <td>2</td> </tr> <tr> <td colspan="2">3</td> </tr> <tr> <td colspan="2"> </td> </tr> </table>	1	2	3				
1	2								
3									
4	WRITE LINE TO TERMINAL NEWPAGE LENGTH 0.	<table border="1"> <tr> <td colspan="2"> </td> </tr> </table>							

Displaying header lines: If you want to display header lines that will appear on the terminal, include the HEADER option of the WRITE LINE TO TERMINAL statement. This header will be displayed until a subsequent WRITE LINE TO TERMINAL request modifies or deletes it.

You can display a maximum of three header lines; each line can be a maximum of two physical terminal lines in length. Headers are cleared at the end of each line I/O session.

7.4.3 Reading a line of data

To transfer data from the terminal buffer to program variable storage, issue a `READ LINE FROM TERMINAL` statement. `READ LINE FROM TERMINAL` transfers data to your program as follows:

- For **line-by-line devices**, DC treats the entire screen contents as a single data field; a `READ LINE FROM TERMINAL` request returns all data to the program at once.
- For **3270-type devices**, a `READ LINE FROM TERMINAL` request returns the first data field on the screen marked for input. DC queues remaining data fields marked for input and passes them back to the program one at a time.

Uses of `READ LINE FROM TERMINAL`: Typical uses of the `READ LINE FROM TERMINAL` function are:

- **To retrieve any information entered in addition to a task code.** That is, for tasks assigned the `INPUT` attribute, the user can enter data following the task code. For example, if the user enters:

```
GETEMP HENDON
```

DC replaces the task code (`GETEMP`) with leading blanks and returns the data (`HENDON`) to the program.

- **To read the one-line response to a `WRITE LINE TO TERMINAL` request that has prompted the terminal operator for information.** On non-3270 devices, when a `READ LINE FROM TERMINAL` request is issued after one or more `WRITE LINE TO TERMINAL` requests, DC writes a question mark (?) to the terminal to indicate that a response is required.
- **To enable a program to read formatted 3270-type data fields sequentially.** The first `READ LINE FROM TERMINAL` request returns the first field on the screen that is marked for input. Subsequent `READ LINE FROM TERMINAL` requests return the remaining fields to the program, one at a time, as illustrated in the example below.

Example of `READ LINE FROM TERMINAL`: In the figure below, the first `READ LINE FROM TERMINAL` request returns the value in the first data field; subsequent `READ LINE FROM TERMINAL` requests return the values in the remaining data fields in the order in which they appear on the screen.

Sequence of <code>READ LINE</code> request	Returned data	Screen contents
1	KING	
2	DORIS	
3	0106	
4	04	
5	040886	

7.4.4 Ending a line mode session

A line mode I/O session ends when one of the following events occurs:

- The task terminates without issuing a DC RETURN request. Programs that specify the NEXT TASK CODE parameter in a DC RETURN request can extend the line I/O session to include data transfers initiated by the next task.
- The user presses one of the following keys:
 - [Clear] — 3270 terminals
 - [Attn] — 2741 terminals
 - [Break] — Teletype terminals
- The program issues an END LINE TERMINAL SESSION request.

Following an END LINE TERMINAL request, DC does not automatically display lines that remain in a partially filled buffer; typically, this data is of no use to the user. However, to display the contents of a partially filled buffer before ending the line I/O session, your program can issue a WRITE LINE TO TERMINAL request that specifies the NEWPAGE option and a dummy data line (that is, one with a length of zero).

7.4.5 3270-type considerations

The following special considerations apply to 3270-type devices:

- DC assigns each page of data in the line I/O session a sequential number starting with 1; page numbers are displayed at the bottom of the screen.
- DC keeps all pages associated with a line I/O session in a scratch area unless otherwise requested. When the I/O session terminates, these pages remain in the scratch area where they can be viewed by the user and subsequently deleted. At any point, the user can display any page either by its position relative to the currently displayed page or by page number:
 - **Next page** — Press [PA1]
 - **Previous page** — Press [PA2] or the CANC key.
 - **Specific page** — Enter the desired page number following the words NEXT PAGE at the bottom of the screen and press [Enter].

Unless the NOBACKPAGE option has been specified in a READ LINE FROM TERMINAL or WRITE LINE TO TERMINAL request, all pages processed during the I/O session remain available until the user signals completion of their use by pressing [Enter] with no request to see another page. If the page displayed is the last page of the session, DC deletes all pages associated with the current session, clears page header lines, and resets the current page number to one (1).

►► For further details regarding line mode DML statements, refer to the language-specific CA-IDMS DML reference manual.

Chapter 8. Storage, Scratch, and Queue Management

- 8.1 About this chapter 8-3
- 8.2 Using storage pools 8-4
 - 8.2.1 User storage 8-5
 - 8.2.2 User kept storage 8-6
 - 8.2.3 Shared storage 8-9
 - 8.2.4 Shared kept storage 8-10
 - 8.2.5 Storage pool summary 8-11
- 8.3 Using scratch records 8-15
- 8.4 Using queue records 8-22
- 8.5 Using the terminal screen to transmit data 8-27

8.1 About this chapter

Pseudoconversational programming demands techniques that efficiently pass data from one task to another. You should choose the method or combination of methods best suited to the needs of your application. You can choose different methods based on the following considerations:

- Length of time that the data is needed
- Availability of the data to other users
- Data recoverability
- System resources used
- Network resources used
- Number of variables

CA-IDMS provides these services for managing online variable storage:

- **Storage pools** manage short-term variable-storage resources and pass data from one task to another
- **Scratch records** pass temporary data between tasks running on the same logical terminal
- **Queue records** pass more permanent data from one task to another
- The **terminal buffer** passes very small amounts of data between tasks running on the same logical terminal.

8.2 Using storage pools

To facilitate online programming and intertask communication, CA-IDMS provides storage management functions that allow you to acquire space explicitly in storage pools.

These functions control allocation of variable storage in a CA-IDMS storage pool or work area. Shared by system and user programs, the storage pool also contains space for buffers and initial storage areas (ISAs) used by Assembler and PL/I programs.

Note: All variable-storage entries (except COBOL LINKAGE SECTION and PL/I BASED storage entries) defined by your program are acquired automatically from the CA-IDMS storage pool when the program starts and released automatically when the program ends.

Using CA-IDMS storage management functions, you can:

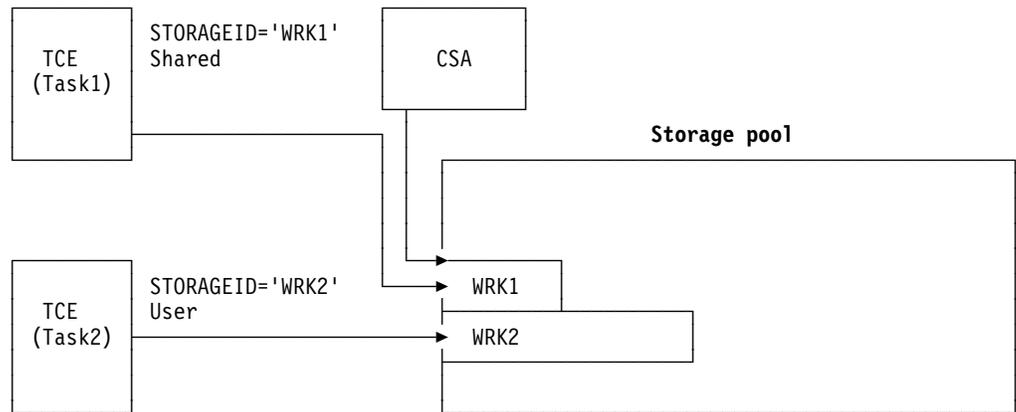
- Acquire variable storage from a storage pool
- Establish addressability to previously acquired variable storage
- Release all or part of previously acquired variable storage

Types of acquired storage: You must specify whether the acquired storage is available to other users:

- **User** storage is available only to the issuing task; no other tasks can access it. CA-IDMS maintains user storage through the issuing task's task control element (TCE).
- **Shared** storage is available to all tasks running under the CA-IDMS system. CA-IDMS links shared storage to the common system area (CSA) as well as to the TCE, as illustrated in the figure below. CA-IDMS uses the CSA to locate the address of a shared area to satisfy requests from other tasks for shared storage.

Note: Shared storage is available to all tasks within the CA-IDMS system; however, each task must explicitly establish addressability to access such storage.

TCE and CSA ownership: Shared storage is linked to both the TCE and CSA; user storage is linked only to the TCE, as the figure below shows.



Kept storage: If you require that storage remain allocated after a task ends, it should be assigned the **KEEP** attribute when it is initially allocated. Kept storage is associated with the logical terminal on which the task is executing and with the task itself; such storage can be released only through a program request.

Releasing storage: When storage is explicitly released or a task terminates, CA-IDMS releases linkage to the TCE.

►► For a quick reference of storage release procedures and conditions, see 8.2.5, “Storage pool summary” on page 8-11 later in this chapter.

User storage only: You can explicitly release all or a part of user storage. For a partial release, TCE linkage and the KEEP attribute remain unaffected.

8.2.1 User storage

User storage is associated exclusively with the issuing task through the TCE; when the task terminates, user storage is released. By dynamically acquiring only the amount of storage needed, you can make more effective use of storage resources.

Steps to acquire user storage: To dynamically acquire and use variable storage from the storage pool within a single task, perform the following steps:

1. Acquire variable storage from the storage pool by issuing a GET STORAGE statement that specifies the USER parameter.
2. Check for an ERROR-STATUS of 3210 (DC-NEW-STORAGE).
3. Perform the IDMS-STATUS routine if 3210 is not returned.
4. Perform processing, using the acquired storage as needed.
5. Release the acquired storage by issuing a FREE STORAGE statement that specifies the appropriate storage ID.

Example of acquiring user storage: The program excerpt below shows the acquisition and release of user storage.

The program acquires the minimum amount of storage needed to complete the processing specified by the user.

```

DATA DIVISION.
LINKAGE SECTION.
01 COPY IDMS RECORD EMPLOYEE.
   05 EMPLOYEE-END      PIC X.
01 COPY IDMS RECORD DEPARTMENT.
   05 DEPARTMENT-END    PIC X.
01 ERROR-DATA.
   05 ERROR-DEPT-ID     PIC 9(4).
   05 ERROR-MESSAGE-CODE PIC X(4).
   05 ERROR-DATA-END    PIC X.
PROCEDURE DIVISION.
MAIN-LINE.
*** THIS PROGRAM ACQUIRES STORAGE FOR EITHER THE ***
*** DEPARTMENT RECORD OR THE EMPLOYEE RECORD ***
*** DEPENDING ON THE CONTROL KEY PRESSED BY THE ***
*** TERMINAL OPERATOR. ***
   BIND MAP SOLICIT.
   BIND MAP SOLICIT RECORD SOLICIT-REC.
   MAP IN USING SOLICIT.
   INQUIRE MAP SOLICIT MOVE AID TO DC-AID-IND-V.
   IF CLEAR-HIT DC RETURN
   ELSE
     IF PA01-HIT GO TO A100-GET-EMPLOYEE
   ELSE
     IF PA02-HIT GO TO A100-GET-DEPARTMENT
   ELSE
     GO TO U100-ERROR-PROC.
*
A100-GET EMPLOYEE.
   IF SOLICIT-EMP-ID NOT NUMERIC
     GO TO U200-ERROR-EMP-ID.
*** ACQUIRE USER STORAGE FOR THE EMPLOYEE RECORD ***
   GET STORAGE FOR EMPLOYEE TO
     EMPLOYEE-END
     NOWAIT SHORT USER
     STGID 'EMPL' VALUE IS LOW-VALUE
   ON DC-NEW-STORAGE
     NEXT SENTENCE.
   MOVE SOLICIT-EMP-ID TO EMP-ID-0415.
   OBTAIN CALC EMPLOYEE
   ON DB-REC-NOT-FOUND
     GO TO U200-ERROR-NO-EMP.

```

8.2.2 User kept storage

User kept storage is available to all tasks running on a logical terminal until a task associated with that terminal releases the storage. User kept storage is ideal for passing small amounts of information between tasks. CA-IDMS maintains TCE linkage for user kept storage across tasks by using the logical terminal element (LTE). When a new task is initiated from the same terminal, CA-IDMS transfers this linkage from the LTE to the TCE of the new task.

Steps to acquire user kept storage: To dynamically acquire and use variable storage from the storage pool and make the storage available to multiple tasks running on the same logical terminal:

1. Acquire variable storage from the storage pool by issuing a GET STORAGE statement that specifies both the USER and the KEEP parameters.

Note: You can indicate that storage is eligible for allocation above the 16Mb line by specifying LOCATION IS ANY on the GET STORAGE statement.

2. Check for an ERROR-STATUS of 3210 (DC-NEW-STORAGE).
3. Perform the IDMS-STATUS routine if 3210 is not returned.
4. Perform processing, using the acquired storage as needed.
5. Issue a DC RETURN statement, optionally specifying the next task to be invoked.

Accessing user kept storage: In subsequent tasks invoked on the same logical terminal:

1. Establish addressability to the previously acquired storage by issuing a GET STORAGE request that names the storage ID specified for the storage area when it was first allocated.
2. Perform processing, using the acquired data.

You should release the acquired storage as soon possible by issuing a FREE STORAGE statement that specifies the appropriate storage ID.

Example of acquiring user kept storage: The program excerpt below shows the initial assignment of user kept storage.

The program performs preliminary error checking before transferring control to a database retrieval program.

```
DATA DIVISION.
01 TRANSPROG          PIC X(8)  VALUE 'DEPTGET'.
01 SOLICIT-REC.
   05 SOLICIT-DEPT-ID  PIC X(4).
LINKAGE SECTION.
01 PASS-DEPT-INFO.
   05 PASS-DEPT-ID     PIC 9(4).
   05 PASS-DEPT-INFO-END PIC X.

PROCEDURE DIVISION.
  BIND MAP SOLICIT.
  BIND MAP SOLICIT RECORD SOLICIT-REC.
*
  MAP IN USING SOLICIT.
  INQUIRE MAP SOLICIT MOVE AID TO DC-AID-IND-V.
  IF CLEAR-HIT DC RETURN.
*
  IF SOLICIT-DEPT-ID NOT NUMERIC
    GO TO ERROR-DEPT-ID.
*** ACQUIRE USER KEPT STORAGE ***
  GET STORAGE FOR PASS-DEPT-INFO
    TO PASS-DEPT-INFO-END
    NOWAIT KEEP LONG USER
    STGID 'DEPT' VALUE IS LOW-VALUE
  ON DC-NEW-STORAGE
  NEXT SENTENCE.
*** MOVE MAP DATA TO FIELDS IN ACQUIRED STORAGE ***
  MOVE SOLICIT-DEPT-ID TO PASS-DEPT-ID.
*** TRANSFER CONTROL TO DATABASE ACCESS PROGRAM ***
  TRANSFER CONTROL TO TRANSPROG
  NORETURN.
```

Reestablishing addressability to user kept storage: The program excerpt below establishes addressability to the previously acquired storage and releases it. The program uses data from the previously acquired storage to perform database access.

```

DATA DIVISION.
01 NTCODES.
   05 NEXT-TASK          PIC X(8)  VALUE 'DEPTMOD'.
01 MESSAGES.
   05 DEPT-DISPLAY-MESS  PIC X(20)
      VALUE 'DEPARTMENT DISPLAYED'
   05 DEPT-DISPLAY-MESS-END. PIC X.
01 SOLICIT-REC.
   05 SOLICIT-DEPT-ID    PIC X(4).
LINKAGE SECTION.
01 PASS-DEPT-INFO.
   05 PASS-DEPT-ID       PIC 9(4).
   05 PASS-DEPT-INFO-END PIC X.

PROCEDURE DIVISION.
*** ESTABLISH ADDRESSABILITY TO PREVIOUSLY ACQUIRED STORAGE ***
    GET STORAGE FOR PASS-DEPT-INFO
        TO PASS-DEPT-INFO-END
        NOWAIT KEEP LONG USER
        STGID 'DEPT'.
    BIND MAP SOLICIT.
    BIND MAP SOLICIT RECORD SOLICIT-REC.
*** MOVE DATA TO DATABASE CALC-KEY AND MAP DATA FIELD ***
    MOVE PASS-DEPT-ID TO DEPT-ID-0410.
    MOVE PASS-DEPT-ID TO SOLICIT-DEPT-ID.
*** RELEASE STORAGE ***
    FREE STORAGE STGID 'DEPT'.
.
*** DATABASE ACCESS ***
.
    MAP OUT USING SOLICIT
        MESSAGE IS DEPT-DISPLAY-MESS TO DEPT-DISPLAY-MESS-END.
    DC RETURN NEXT TASK CODE NEXT-TASK.

```

8.2.3 Shared storage

Shared storage is available to all tasks running concurrently under the CA-IDMS system.

Shared storage is usually accessed by a concurrent nonterminal task. For example, such a nonterminal task might support the main task by performing print functions.

►► For more information on nonterminal tasks, see 9.13, “Initiating nonterminal tasks” on page 9-35.

When shared storage is released: CA-IDMS maintains an in-use counter for each area of shared storage. Each time a task establishes addressability to an area of shared storage, CA-IDMS adds 1 to the in-use counter. When a task terminates or releases the storage, CA-IDMS subtracts 1 from the in-use counter. CA-IDMS releases shared storage when the in-use counter is set to zero.

Steps to acquire shared storage: To dynamically acquire and use variable storage from the storage pool and make the storage available to other tasks running under the same CA-IDMS system, perform the following steps:

1. Acquire variable storage from the storage pool by issuing a GET STORAGE statement that specifies the SHARED parameter.

2. Check for an ERROR-STATUS of 3210 (DC-NEW-STORAGE).
3. Perform the IDMS-STATUS routine if 3210 is not returned.
4. Perform processing, using the acquired storage as needed.
5. Optionally, release the shared storage by issuing a FREE STORAGE statement that specifies the appropriate storage ID.

Steps to access shared storage: To access the data from another task executing concurrently under the same CA-IDMS system, perform the following steps:

1. Establish addressability to the previously acquired storage by issuing a GET STORAGE request that names the storage ID specified for the storage area when it was first allocated.
2. Perform processing using the acquired data.
3. Optionally, release the shared storage by issuing a FREE STORAGE statement that specifies the appropriate storage ID.

8.2.4 Shared kept storage

Shared kept storage is available to all tasks running under the CA-IDMS system. Once a storage area with the SHARED KEEP attribute is established, any task running under the CA-IDMS system can access that area.

When shared kept storage is released: CA-IDMS maintains an in-use counter and a keep flag for each area of shared kept storage. Shared kept storage is released only when *both* of the following conditions are true:

1. The in-use counter is set to zero, indicating that there are no current users of the area.
2. The keep flag is turned off (the FREE STORAGE statement turns the keep flag off)

If either condition is false, the storage area remains allocated. With this feature, shared kept storage areas remain allocated even when they are not being used, provided the keep flag remains on.

Each time a task establishes addressability to an area of shared kept storage, CA-IDMS adds 1 to the in-use counter. When the task terminates, CA-IDMS subtracts 1 from the in-use counter. When a program issues a FREE STORAGE request, CA-IDMS subtracts 1 from the in-use counter *and* turns off the keep flag. Once a FREE STORAGE request is issued, if the in-use counter is zero, CA-IDMS releases the storage. Once turned off, the keep flag cannot be reset.

Difference from user kept storage: Unlike user kept storage, shared kept storage is not linked to the LTE across tasks executing on the same terminal.

Startup shared kept storage: One shared storage area having the keep attribute is allocated at system startup for use by all tasks; this storage area is never freed. This area is called the common work area (CWA) and can contain application-defined information, if so requested during system generation.

►► For more information about the CWA, refer to *CA-IDMS System Generation*.

Example of shared kept storage: The program excerpt below shows programmatic access to data previously placed in the CWA.

This program accesses the CWA in order to obtain the current date in Gregorian format:

```

DATA DIVISION.
01 NTCODES.
   05 NEXT-TASK          PIC X(8)  VALUE 'DEPTGET'.
01 CWA                   PIC X(4)  VALUE 'CWA'.
01 SOLICIT-REC.
   05 SOLICIT-DEPT-ID   PIC X(4).
   05 SOLICIT-GREG-DATE PIC X(8).
LINKAGE SECTION.
01 CWA-DATA.
   05 CWA-DATE          PIC X(8).
   05 CWA-DATA-END     PIC X.

PROCEDURE DIVISION.
  BIND MAP SOLICIT.
  BIND MAP SOLICIT RECORD SOLICIT-REC.
*** GET THE DATE IN GREGORIAN FORMAT FROM THE CWA ***
  GET STORAGE FOR CWA-DATA TO CWA-DATA-END
  NOWAIT KEEP SHORT SHARED
  STGID CWA.
  MOVE ZEROS TO SOLICIT-DEPT-ID.
  MOVE CWA-DATE TO SOLICIT-GREG-DATE.
  MAP OUT USING SOLICIT.
  DC RETURN NEXT TASK CODE NEXT-TASK.

```

8.2.5 Storage pool summary

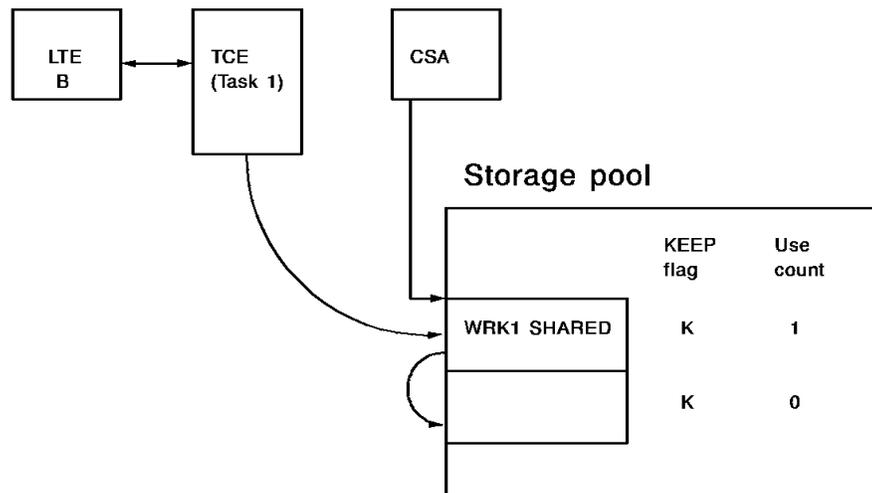
Acquired storage is associated with the TCE, the CSA, or both. Additionally, user storage with the keep attribute is linked to the LTE.

The table below shows the procedures and conditions under which CA-IDMS maintains linkage when storage is released. This table assumes that the FREE STORAGE request releases the entire storage area.

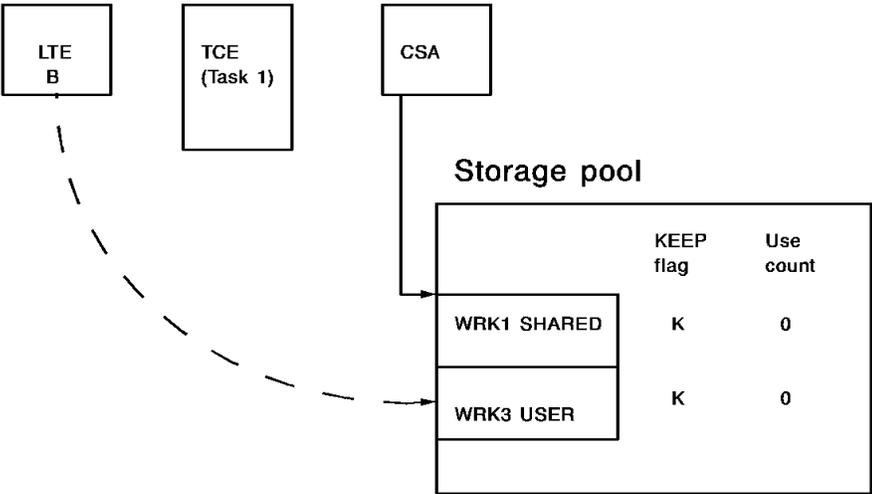
Storage attribute	After FREE STORAGE request	After task termination
USER	Storage is released.	Storage is released.
USER KEEP	Storage is released.	Storage remains allocated; TCE linkage is transferred to the LTE.
SHARED	Storage is released only if the in-use counter is set to zero.	Storage is released only if the in-use counter is set to zero.
SHARED KEEP	Storage is released only if the in-use counter is set to zero.	Storage remains allocated.

How storage is allocated and released: The following diagrams illustrate how CA-IDMS allocates and releases storage.

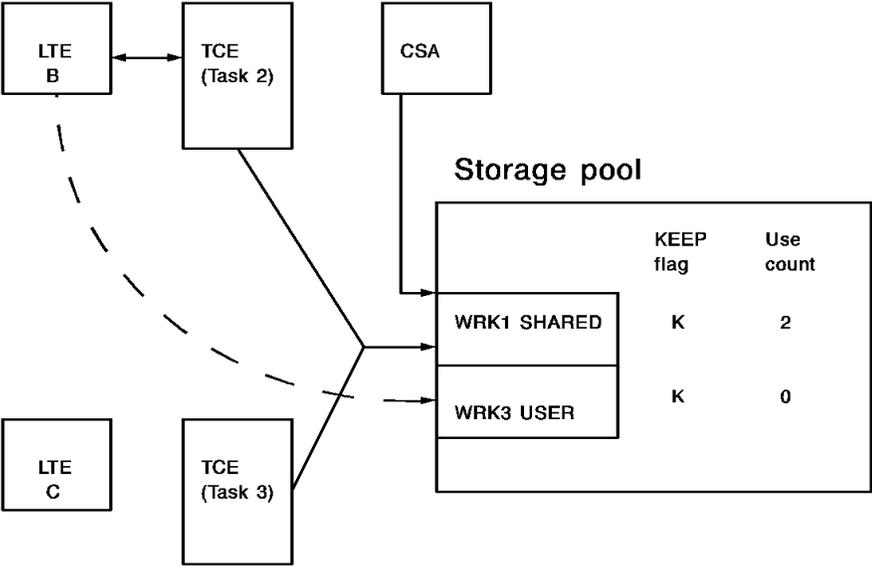
- Task 1, running on terminal B, establishes addressability to two variable areas of kept storage. WRK1 is designated shared keep; WRK3 is designated user keep. Because task 1 is the only task using WRK1, the in-use counter associated with WRK1 is set to 1.



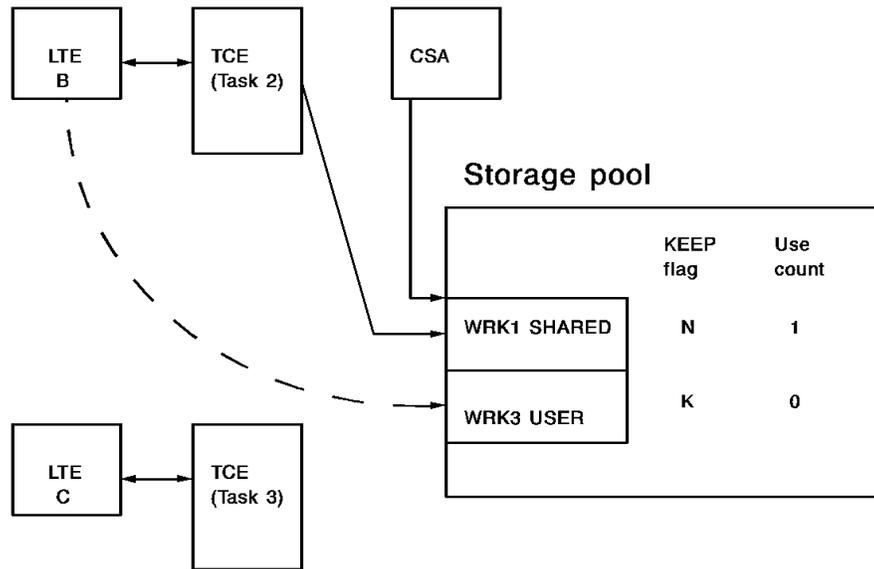
- Task 1 terminates without issuing a FREE STORAGE request for either WRK1 or WRK3. CA-IDMS automatically decrements the in-use counter and transfers linkage for WRK3 to the LTE for terminal B. Because WRK1 is shared, CA-IDMS does not maintain linkage to the LTE. Although WRK1 has no users, it remains allocated because an explicit FREE STORAGE was not issued.



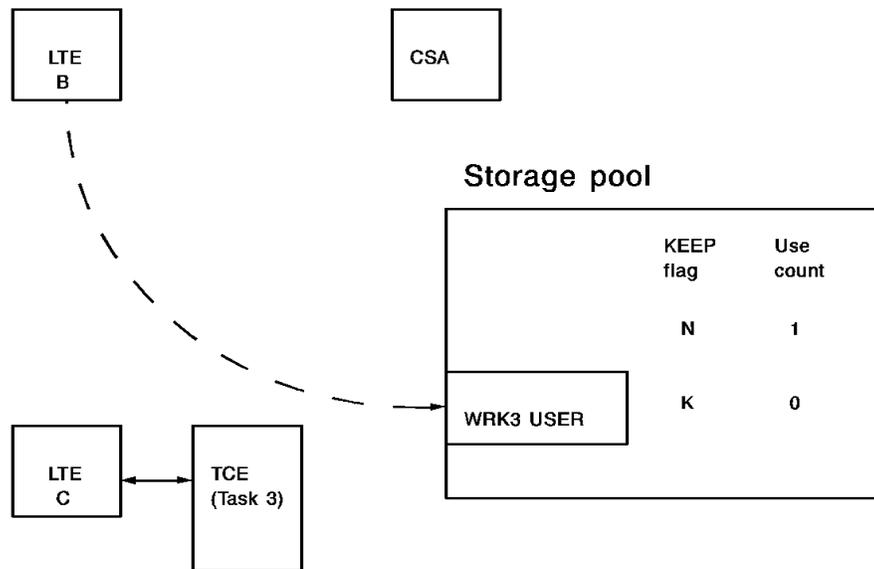
3. Task 2 is initiated on terminal B and issues a GET STORAGE request for WRK1. Task 3 is initiated on terminal C and issues a GET STORAGE request for WRK1. The in-use counter for WRK1 indicates two users.



4. Task 3 issues a FREE STORAGE request for WRK1; CA-IDMS turns the keep flag off and decrements the in-use counter by 1.



- Task 2 terminates without issuing a FREE STORAGE request for WRK1; CA-IDMS decrements the in-use counter by 1. Because the keep flag is off and the in-use counter is set to zero, CA-IDMS releases the storage associated with WRK1.



8.3 Using scratch records

CA-IDMS scratch management functions allow you to allocate, retrieve, and delete scratch records. Scratch records, which are stored in the DDLDCSCR area of the dictionary, are used to pass data from one task to subsequent tasks running on the same terminal. These records are not accessible to tasks executing on other terminals.

Fast access: Scratch records provide fast access because:

- **Scratch records are indexed.** They are stored in an indexed set in the DDLDCSCR area of the dictionary.
- **The DDLDCSCR area provides efficient access.** It is initialized at system startup; any previously existing records are deleted.
- **Scratch records are unavailable to other users.** You do not have to wait for record locks to be freed.

Best use of scratch records: Scratch records are not recoverable across a shutdown/startup or a system crash. All scratch records are deleted at system startup. Because they are not saved across a system shutdown, scratch records are best used for temporary storage of data.

Availability to a subsequent task: When a task terminates, CA-IDMS temporarily associates that task's scratch areas with the logical terminal from which the task was invoked. This is done using the logical terminal element (LTE). When a new task is initiated on the same terminal, CA-IDMS transfers the scratch areas to the task control element (TCE) for the new task. All scratch records and currencies associated with the old task are available to the new task.

What you can do with scratch records: You can use CA-IDMS scratch management functions to do the following:

- Store or replace a scratch record in the dictionary
- Retrieve a scratch record from the dictionary and place it in a variable-storage area associated with the issuing task
- Delete a scratch record from the dictionary

Steps to allocate or replace a scratch record: To allocate or replace a scratch record, perform the following steps:

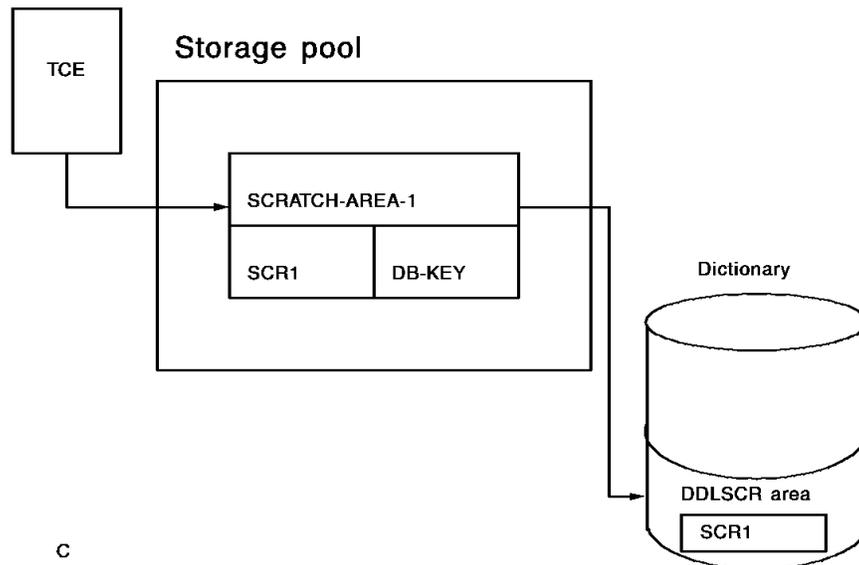
1. Initialize the appropriate fields in program variable storage.
2. Issue a PUT SCRATCH command that specifies the variable-storage location of the data to be stored; to replace a record, include the REPLACE parameter.
3. If you specify the REPLACE parameter, check for a status of 4317 (DC-REC-REPLACED).
4. Perform the IDMS-STATUS routine. (If you specify REPLACE, perform this step only if 4317 is not returned.)

Scratch area: In response to your PUT SCRATCH request, CA-IDMS places the scratch record in the DDLDCSCR area of the dictionary. An index pointer to the record is placed in a storage pool scratch area. Each scratch area is identified by its area ID; scratch records in each area are indexed in ascending order by scratch record ID (SRID).

Typically, your program assigns the SRID. If not, CA-IDMS assigns the SRID, places the record last within the scratch area, and returns the SRID to your program.

Any number of scratch areas can be associated with a task and any number of scratch records can be associated with a scratch area.

Example of scratch record allocation: The figure below shows scratch record allocation. When a PUT SCRATCH request is issued, CA-IDMS creates a scratch record in the dictionary and places a pointer to that record in a scratch area associated with the issuing task.



Steps to retrieve a scratch record: To retrieve a scratch record, perform the following steps:

1. Issue a GET SCRATCH command that specifies the appropriate scratch area ID and indicates the variable-storage location in which the scratch record is to be placed. You can retrieve scratch records by position within the area, by relationship to the current record of the scratch area, or by SRID.
2. If you are issuing the GET SCRATCH command iteratively and specifying the DELETE parameter, check for a status of 4303 (DC-AREA-ID-UNK); this indicates the end of the scratch area. If you specify KEEP, check for a status of 4305 (DC-REC-NOT-FOUND); this indicates the end of the scratch area.

If there is any chance that the length of the retrieved record exceeds the length of its allocated variable storage, you should do the following:

- Include the KEEP parameter of the GET SCRATCH statement to ensure that data is not deleted when it is retrieved.
- Check for a status of 4319 (DC-TRUNCATED-DATA).

3. Perform the IDMS-STATUS routine if neither 4303, 4305, nor 4319 is returned.

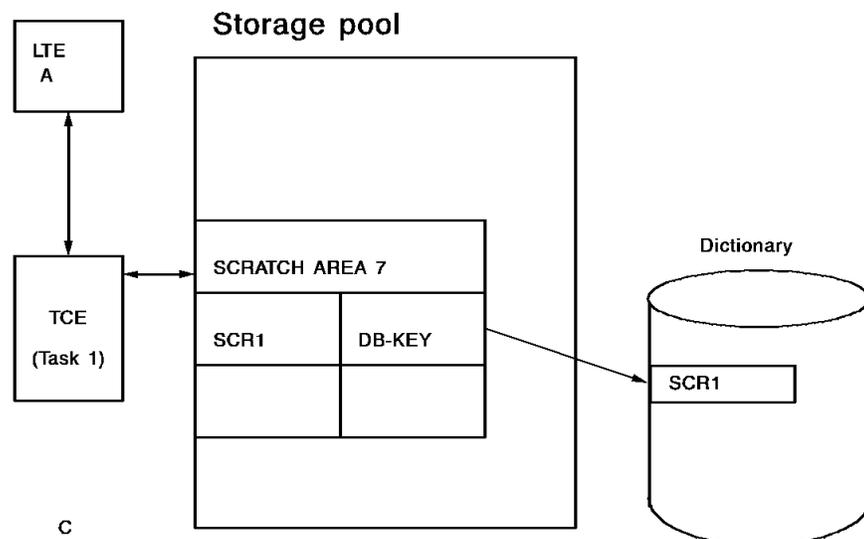
Scratch record currency: CA-IDMS maintains currency for the records in each scratch area. Because CA-IDMS maintains currency across tasks, you should be aware that the NEXT option does not default to FIRST, and PRIOR does not default to LAST.

Steps to delete a scratch record: To delete a scratch record, issue either of the following commands:

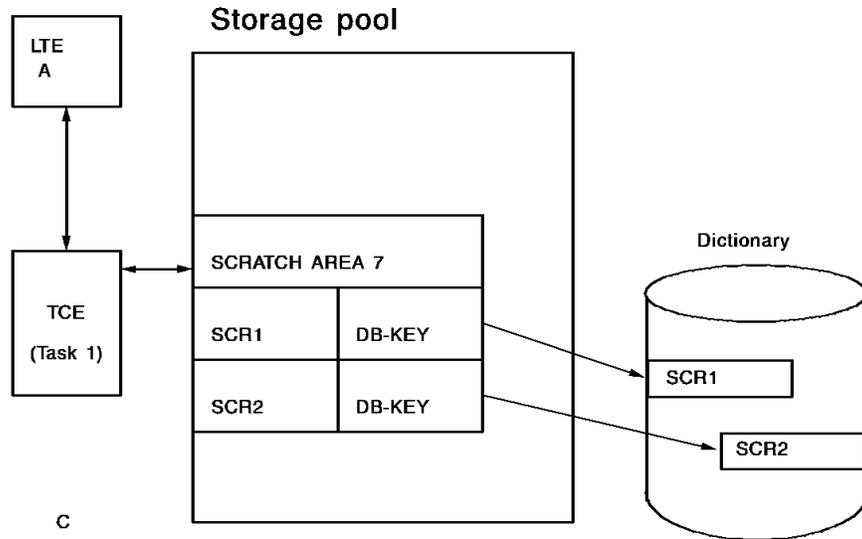
- A GET SCRATCH command that specifies the DELETE parameter. CA-IDMS copies the scratch record to the appropriate variable-storage area and deletes the record. When all scratch records associated with a given scratch area have been deleted, CA-IDMS deletes the scratch area. CA-IDMS returns a status of 4303 (DC-AREA-ID-UNK) to later GET SCRATCH requests that specify that area ID.
- A DELETE SCRATCH command that specifies one of the following:
 - That a particular occurrence of the scratch record is to be erased
 - That the entire scratch area should be erased

Allocating scratch records across tasks: The following diagrams illustrate how CA-IDMS dynamically allocates scratch records across tasks:

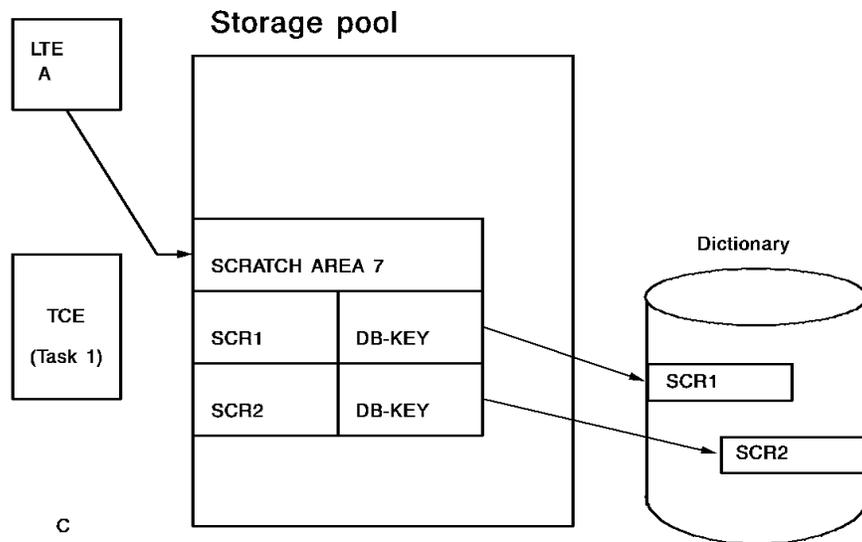
1. Task 1 stores scratch record SCR1 in scratch area 7. Because no scratch area with that identifier exists for task 1, CA-IDMS dynamically allocates the area within the variable-storage pool. A scratch record is placed in the dictionary and is associated with task 1's TCE.



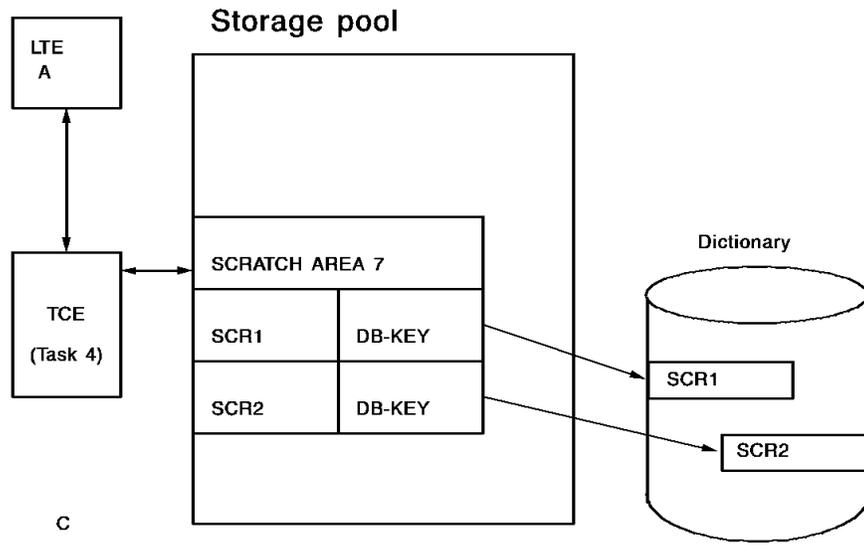
- Task 1 stores SCR2 in scratch area 7. CA-IDMS creates a second entry in scratch area 7 and places the new record in the dictionary.



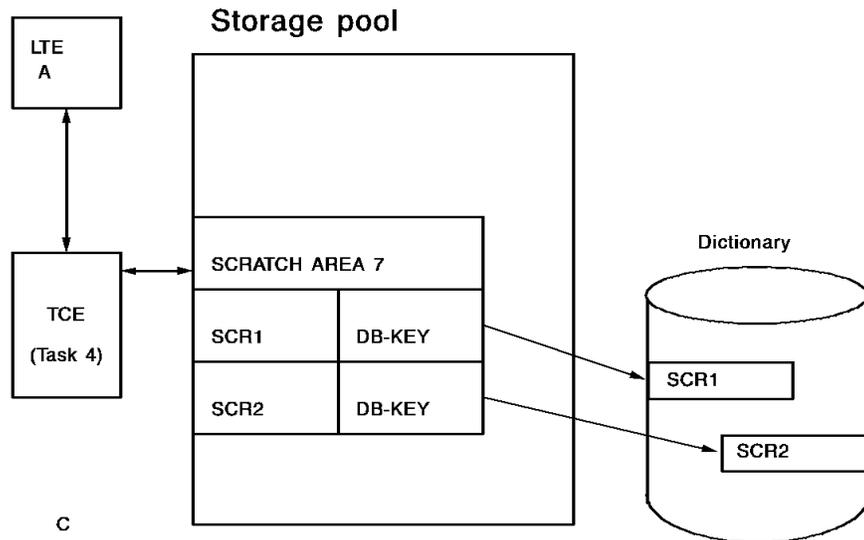
- Task 1 terminates. CA-IDMS associates scratch area 7 with the LTE for terminal A. Scratch area 7 is no longer associated with task 1.



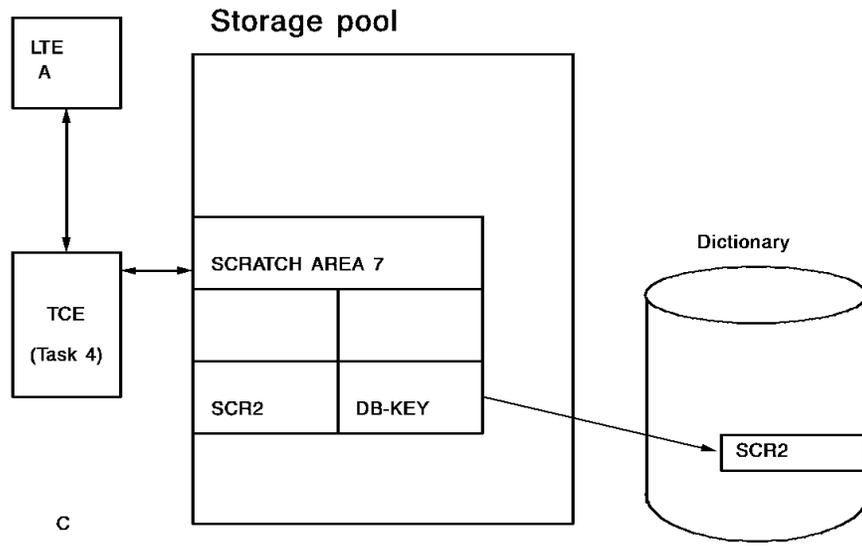
- Task 4 is initiated on terminal A. CA-IDMS associates scratch area 7 with task 4's TCE.



5. Task 4 issues a GET SCRATCH to obtain SCR2. Data associated with scratch record SCR2 now resides in variable storage for task 4, as well as in the dictionary.



6. Task 4 deletes SCR1. CA-IDMS deletes the scratch area entry for that record and removes the record from the dictionary.



Example of retrieving scratch records: The program excerpt below retrieves scratch records from the TEST-SCRATCH scratch area. The program uses a pageable map in order to display an unlimited number of scratch records.

The program retrieves all occurrences in the TEST-SCRATCH scratch area. Each occurrence contains the employee's ID, last name, and first name.

```

WORKING-STORAGE SECTION.
01 TC                                PIC X(8).
   88 GETOUT                          VALUE 'GETSCR2'.
01 SWITCHES.
   05 FIRST-PAGE-SW                    PIC X VALUE 'N'.
   88 LESS-THAN-A-PAGE                 VALUE 'N'.
01 GETSCR2                            PIC X(8) VALUE 'GETSCR2'.
01 TESTSCR                            PIC X(8) VALUE 'TESTSCR'.
01 TEST-SCRATCH.
   05 SCR-ID                           PIC 9(4).
   05 SCR-LNAME                        PIC X(15).
   05 SCR-FNAME                        PIC X(10).
   05 TEST-SCRATCH-END                 PIC X.
01 SCRMAP-REC.
   02 ID                               PIC 9(4).
   02 LNAME                           PIC X(15).
   02 FNAME                           PIC X(10).
PROCEDURE DIVISION.
MAIN-LINE.
  ACCEPT TASK CODE INTO TC.
  IF GETOUT ENDPAGE
    DC RETURN.
  BIND MAP SCRMAP01.
  BIND MAP SCRMAP01 RECORD SCRMAP-REC.
  STARTPAGE SESSION SCRMAP01 NOWAIT BACKPAGE BROWSE
  ON DC-SECOND-STARTPAGE NEXT SENTENCE.
*
  GET SCRATCH AREA ID TESTSCR FIRST KEEP
  INTO TEST-SCRATCH TO
  TEST-SCRATCH-END
  ON DC-AREA-ID-UNK
  GO TO ERR-NO-SCR.
  MOVE SCR-ID TO ID.
  MOVE SCR-LNAME TO LNAME.
  MOVE SCR-FNAME TO FNAME.
  MAP OUT USING SCRMAP01
  DETAIL NEW.
  PERFORM A100-GET-SCRATCH THRU A100-EXIT
  UNTIL DC-REC-NOT-FOUND.
  IF LESS-THAN-A-PAGE
  MAP OUT USING SCRMAP01
  NEWPAGE RESUME.
  DC RETURN NEXT TASK CODE GETSCR2.

A100-GET-SCRATCH.
  GET SCRATCH AREA ID TESTSCR NEXT KEEP
  INTO TEST-SCRATCH TO
  TEST-SCRATCH-END
  ON DC-REC-NOT-FOUND GO TO A100-EXIT.
  MOVE SCR-ID TO ID.
  MOVE SCR-LNAME TO LNAME.
  MOVE SCR-FNAME TO FNAME.
  MAP OUT USING SCRMAP01
  DETAIL NEW
  ON DC-FIRST-PAGE-SENT
  MOVE 'Y' TO FIRST-PAGE-SW.
A100-EXIT.
  EXIT.

```

8.4 Using queue records

CA-IDMS queue management functions allow you to store, retrieve, and delete queue records. Queue records, which are stored in the dictionary, are available to all tasks running under CA-IDMS and to batch programs with an operating mode of DC-BATCH.

Queue records are saved across a system shutdown/startup and recovered across a system crash; however, currencies are lost when the system crashes or is shut down.

In a data sharing environment, queues can be shared between members of a data sharing group.

Queue record storage: CA-IDMS stores queue records in the DDLDCRUN area of the dictionary. Each queue record is a member record in a set owned by a queue header record. All records associated with one queue header are referred to collectively as a **queue**. You can direct records to queues defined at system generation, to queues defined through the DDDL compiler, to program-defined queues, or to null queues.

Sharing queues between CA-IDMS systems: In a data sharing environment, queues can be shared between CA-IDMS systems that are members of a sharing group. The benefit of a shared queue is that it can be read and updated by programs executing on any member of the group. Whether or not a specific queue is shared, is determined by specifications made by the CA-IDMS system administrator. Programs accessing queues are not sensitive to whether or not a queue is shared, since the DML syntax is the same in either case.

How you can use queue management: You can use CA-IDMS queue management functions to do the following:

- Store a queue record and assign an ID to uniquely identify the record
- Retrieve a queue record and place it in a variable-storage area associated with the issuing task
- Delete a record from a specified queue
- Delete an entire queue

Steps to store a queue record: To store a queue record, perform the following steps:

1. Initialize the appropriate fields in program variable storage.
2. Issue a PUT QUEUE command that specifies the variable-storage location of the data to be stored.

Steps to retrieve a queue record: To retrieve a queue record, perform the following steps:

1. Issue a GET QUEUE command that specifies the appropriate queue ID and indicates the variable-storage location in which the queue record is to be placed.

If there is any chance that the length of the retrieved record exceeds the length of its allocated variable storage, you should do the following:

- Include the KEEP parameter of the GET QUEUE statement to ensure that the record is not deleted when it is retrieved.
 - Check for a status of 4419 (DC-TRUNCATED-DATA).
2. Check for a status of 4405 (DC-REC-NOT-FOUND), which indicates that you have retrieved all queue records for the specified queue ID.
 3. Perform the IDMS-STATUS routine if neither 4405 nor 4419 is returned.

Queue record currency: CA-IDMS maintains currency for each queue by task. If several tasks are accessing a queue concurrently, CA-IDMS maintains currency separately for each task. Access to a queue record can be by queue ID, by position within the queue, or by relationship of the specified record to the current record of the queue.

Steps to delete a queue record: To delete a queue record, issue either of the following commands:

- A GET QUEUE command that specifies the DELETE parameter. CA-IDMS copies the record's data to the appropriate variable-storage area and deletes the record.
- A DELETE QUEUE command that specifies one of the following:
 - That the current occurrence of the queue record is to be erased
 - That the entire queue should be deleted

Implicit deletion of queue records: CA-IDMS saves the next and prior currencies following a DELETE QUEUE function so that you can still access the next and prior records in the queue. When all records associated with a given queue have been deleted, CA-IDMS deletes the header record as well. Queue records are also deleted implicitly if the associated queue header record is deleted.

Deleting queues: Queues can also be deleted at system startup or at runtime:

- **At system startup** -- Each queue is assigned a retention period; the retention period specifies the number of days that CA-IDMS will retain the queue. At system startup, CA-IDMS deletes queues that have exceeded their retention periods.
- **At runtime** -- The DCMT VARY QUEUE command can be used to delete unwanted queues at runtime.

►► For more information on DCMT commands, refer to *CA-IDMS System Operations*.

Queue record locks: Because queues are shared among tasks, CA-IDMS must ensure that two tasks do not update a queue record concurrently, causing unexpected alteration of data. Additionally, if a task terminates abnormally, CA-IDMS must ensure that the queue can be restored to its state before the failure. To accomplish this, CA-IDMS handles queues in the following manner:

- When a task stores or retrieves a queue record, CA-IDMS places an implicit exclusive lock on that record, thereby preventing it from being retrieved or updated by other tasks.
- All records locked by CA-IDMS remain locked until the task terminates or until your program issues a COMMIT TASK statement. COMMIT TASK causes some or all of the locks to be released, as specified.
- Queue currencies and locks are not passed from one task to the next on a terminal. Each task is responsible for reestablishing any required currencies.

Avoiding task waits for queue access: Only one task can access a queue record at a time; other tasks attempting access must wait until the current task is complete. Therefore, you should ensure that queue access is short lived. There should be no long waits, such as pseudoconverses, embedded within queue access code.

Retrieving queue records: The program excerpt below retrieves and displays queue records. This program uses a pageable map in order to display an unlimited number of queue records.

The program retrieves all occurrences in the DISPQ queue. This queue lists the employee's ID and last name, and the date and time that each queue record was established.

```

WORKING-STORAGE SECTION.
01 TC                                PIC X(8).
   88 GETOUT                          VALUE 'QOUT'.
01 SWITCHES.
   05 FIRST-PAGE-SW                    PIC X VALUE 'N'.
   88 LESS-THAN-A-PAGE                 VALUE 'N'.
01 GETQUE2                            PIC X(8) VALUE 'QOUT'.
01 CURR-TIME                          PIC X(11).
01 CURR-DATE                          PIC S9(7) COMP-3.
01 MESSAGES.
   05 DIS-QUE-MESS                     PIC X(20) VALUE
   'QUEUE TESTQ DISPLAYED'.
   05 DIS-QUE-MESS-END                 PIC X.
01 TESTQ                              PIC X(6) VALUE 'TESTQ'.
01 TEST-QUEUE.
   05 Q-ID                             PIC 9(4).
   05 Q-LNAME                          PIC X(15).
   05 Q-TIME                           PIC X(11).
   05 Q-DATE                           PIC 9(5).
   05 TEST-QUEUE-END                   PIC X.
01 QUEMAP-REC.
   05 ID                               PIC 9(4).
   05 LNAME                            PIC X(15).
   05 QTIME                            PIC X(11).
   05 QDATE                            PIC 9(5).
   05 MAP-DATE                         PIC 9(5).
   05 MAP-TIME                         PIC X(11).
PROCEDURE DIVISION.
MAIN-LINE.
  BIND MAP QUEMAP01.
  BIND MAP QUEMAP01 RECORD QUEMAP-REC.
  ACCEPT TASK CODE INTO TC.
  IF GETOUT ENDPAGE
    DC RETURN.
  GET TIME INTO CURR-TIME EDIT
    DATE INTO CURR-DATE.
  MOVE CURR-TIME TO MAP-TIME.
  MOVE CURR-DATE TO MAP-DATE.
  STARTPAGE SESSION QUEMAP01 NOWAIT BACKPAGE BROWSE
    ON DC-SECOND-STARTPAGE NEXT SENTENCE.
*
  PERFORM A100-GET-QUEUE-REC THRU A100-EXIT
    UNTIL DC-REC-NOT-FOUND.
  IF LESS-THAN-A-PAGE
    MAP OUT USING QUEMAP01
    NEWPAGE RESUME
    MESSAGE IS DIS-QUE-MESS TO DIS-QUE-MESS-END.
*
  DC RETURN NEXT TASK CODE GETQUE2.
*
```

```
A100-GET-QUEUE-REC.  
  GET QUEUE ID TESTQ NEXT KEEP  
  INTO TEST-QUEUE TO  
  TEST-QUEUE-END  
  ON DC-REC-NOT-FOUND GO TO A100-EXIT.  
  MOVE Q-ID      TO ID.  
  MOVE Q-LNAME   TO LNAME.  
  MOVE Q-TIME    TO QTIME.  
  MOVE Q-DATE    TO QDATE.  
  MAP OUT USING QUEMAP01  
  DETAIL NEW  
  ON DC-FIRST-PAGE-SENT  
  MOVE 'Y' TO FIRST-PAGE-SW.  
A100-EXIT.  
EXIT.
```

8.5 Using the terminal screen to transmit data

You can transfer small amounts of alphanumeric data between tasks by using map data fields defined with the following attributes:

- Dark
- Protected
- MDT set on (nonpageable maps only)

For example, you can convert a record's db-key to display format and transmit the reformatted db-key in the map data stream to allow for DB-KEY retrieval on subsequent database access. You can also transmit the next task code to be invoked by a program.

The terminal screen is ideal for transmitting small amounts of data; more than a small amount of data can affect transmission time.

Example of transmitting screen data: The program excerpt below uses the terminal screen to transmit the db-key of a database record to be modified. This allows for more efficient database access.

The program uses the record's db-key, which was transmitted in the map data stream, to retrieve the EMPLOYEE record.

8.5 Using the terminal screen to transmit data

```
01 MAP-WORK-REC.
   05 WORK-DEPT-ID      PIC 9(4).
   05 WORK-EMP-ID      PIC 9(4).
   05 WORK-FIRST       PIC X(10).
   05 WORK-LAST        PIC X(15).
   05 WORK-ADDRESS     PIC X(42).
   05 WORK-DEPT-NAME   PIC X(45).
   05 DARK-DBKEY       PIC X(12).
   05 RETRIEVE-DBKEY   PIC S9(8) COMP.
PROCEDURE DIVISION.
  BIND MAP EMPMAP.
  BIND MAP EMPMAP RECORD MAP-WORK-REC.
  MAP IN USING EMPMAP.
  IF WORK-EMP-ID NOT NUMERIC
    GO TO U100-INVALID-EMP-ID.
*
  COPY IDMS SUBSCHEMA-BINDS.
  READY.
*** CHANGE DARK-DBKEY FROM DISPLAY TO COMP ***
  MOVE DARK-DBKEY TO RETRIEVE-DBKEY.
  OBTAIN EMPLOYEE DB-KEY IS RETRIEVE-DBKEY
  ON DB-REC-NOT-FOUND
  GO TO U100-INVALID-DBKEY.
*
  MOVE WORK-FIRST TO EMP-FIRST-NAME-0415.
  MOVE WORK-LAST TO EMP-LAST-NAME-0415.
  MOVE WORK-ADDRESS TO EMP-ADDRESS-0415.
  MODIFY EMPLOYEE.
  FINISH.
*** MAP OUT PROCESSING AND ERROR ROUTINES ***
.
.
.
```

Chapter 9. DC Programming Techniques

9.1 About this chapter	9-3
9.2 Passing program control	9-4
9.2.1 Returning to a higher level program	9-5
9.2.2 Passing control laterally	9-6
9.2.3 Passing control, expecting to return	9-7
9.3 Retrieving task-related information	9-9
9.4 Maintaining data integrity in the online environment	9-11
9.4.1 Setting longterm explicit locks	9-11
9.4.2 Monitoring concurrent database access	9-14
9.5 Managing tables	9-18
9.6 Retrieving the current time and date	9-21
9.7 Writing to the journal file	9-23
9.8 Collecting DC statistics	9-25
9.9 Sending messages	9-28
9.9.1 Sending a message to the current user	9-28
9.9.2 Sending a message to other users	9-29
9.10 Writing to a printer	9-31
9.11 Writing JCL to a JES2 internal reader	9-33
9.12 Modifying a task's priority	9-34
9.13 Initiating nonterminal tasks	9-35
9.13.1 Attaching a task	9-35
9.13.2 Time-delayed tasks	9-36
9.13.3 External requests	9-36
9.13.4 Queue threshold tasks	9-36
9.14 Controlling abend processing	9-37
9.14.1 Terminating a task	9-37
9.14.2 Handling db-key deadlocks	9-37
9.14.3 Performing abend routines	9-39
9.15 Establishing and posting events	9-41

9.1 About this chapter

This chapter discusses programming techniques used to request DC services. Functionally similar DC DML statements are presented together; sample code that demonstrates typical usage of each statement is included. The DC DML functions are divided into these categories:

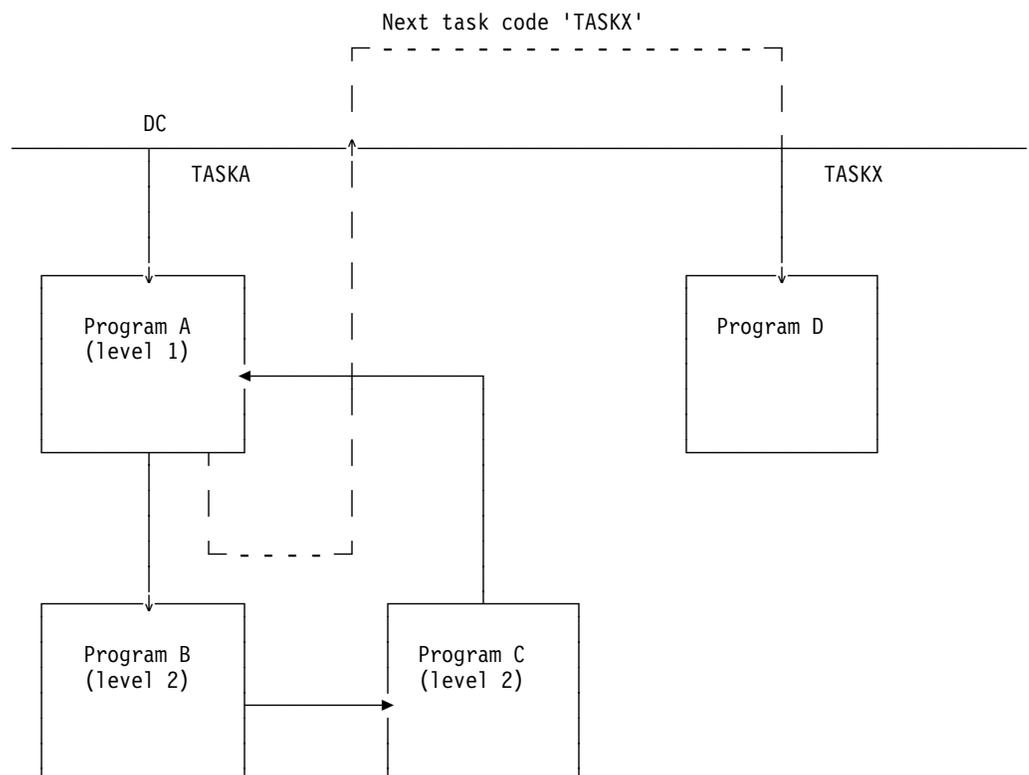
- Controlling the flow of processing in the different levels of your task
- Retrieving task-related information — Accessing system, terminal, and user information related to the current task
- Maintaining online data integrity — Monitoring concurrent database access locking database records across tasks
- Managing tables — Adding and deleting tables from the program pool
- Retrieving the current time and date — Accessing the time and date from the DC system
- Writing to the journal file — Writing task-defined records to the journal file
- Collecting DC statistics — Accessing runtime transaction statistics
- Sending messages — Transmitting messages to other terminals, the user, and the log file
- Writing to a printer — Directing data to printer devices
- Writing JCL to a JES2 internal reader — Sending a JCL stream from the application program to a JES2 internal reader
- Modifying a task's priority — Changing the dispatching priority of a task
- Initiating nonterminal tasks — Using nonterminal tasks
- Controlling abend processing — Specifying the flow of control in the event of an abend
- Establishing and posting events — Establishing and posting event control blocks

9.2 Passing program control

DC provides program management facilities that allow you to pass control either between programs in a single task thread or from task to task. Using these program management functions, you can:

- Return control to the next-higher level within a task, optionally specifying the next task to be invoked on the same terminal
- Initiate execution of a program on the same level within a task; control cannot return to the calling program
- Initiate execution of a subordinate-level program within the same task, with the expectation that control will return to the instruction immediately following the request

Levels of program control: The figure below shows levels of programs in a task. TASKA invokes Program A, which calls Program B expecting return of control. Program B passes control laterally to Program C, which then returns control to Program A. When Program A is finished, it returns control to DC specifying that TASKX should be the next task invoked on that logical terminal.



9.2.1 Returning to a higher level program

You can return control to a higher level within a task or to DC. If you return control to DC and specify the next task code to be invoked, the task ends and a pseudoconverse begins.

DC RETURN statement: To return control to the next-higher level in a task, issue a DC RETURN statement, optionally specifying the next task code to be invoked on the terminal.

If the next-higher-level program specifies a next task code, it overrides any task code specified by the subordinate program. If the issuing program is the highest-level program, DC regains control.

Note: You can bypass intervening link levels and return control to DC by issuing a DC RETURN IMMEDIATE statement.

When the next task is invoked: DC invokes the next task differently depending on how it is defined to the DC system:

- If the next task is defined with the **INPUT** attribute, it is executed when the user next presses an AID key.
- If the next task is defined with the **NOINPUT** attribute, it is executed immediately.

Example of return specifying next task: The program excerpt below returns control to DC and specifies the next task code to be invoked on that terminal.

The first DC RETURN statement returns control to DC. The second DC RETURN statement also specifies that DEPTDISM is the next task invoked on that terminal.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01  DEPTDISM                PIC X(8)  VALUE 'DEPTDISM'.
01  SOLICIT-REC.
    05  SOLICIT-DEPT-ID     PIC X(4).
PROCEDURE DIVISION.
*** ESTABLISH ADDRESSABILITY TO THE MAP AND MAP RECORD ***
    BIND MAP SOLICIT.
    BIND MAP SOLICIT RECORD SOLICIT-REC.
*** CHECK THE AID BYTE ***
    INQUIRE MAP SOLICIT MOVE AID TO DC-AID-IND-V.
*** RETURN CONTROL TO CA-IDMS/DC IF OPERATOR HAS PRESSED CLEAR ***
    IF CLEAR-HIT
        DC RETURN.
        MOVE ZERO TO SOLICIT-DEPT-ID.
*** TRANSMIT THE MAP TO THE TERMINAL SCREEN ***
    MAP OUT USING SOLICIT
        NEWPAGE
        MESSAGE IS INITIAL-MESSAGE LENGTH 80.
*** RETURN CONTROL TO CA-IDMS/DC AND SPECIFY THE NEXT TASK ***
    DC RETURN
        NEXT TASK CODE DEPTDISM.

```

9.2.2 Passing control laterally

After DC gives control to the program specified by an initial task code, that program can transfer control to other DC programs on the same level. That is, the issuing program does not expect return of control.

Steps to transfer control: To transfer control laterally, perform the following steps:

1. Invoke the main program specified by the task code.
2. Perform processing, as required.
3. Acquire storage for any parameters to be passed.
4. Transfer control to the second program by issuing a TRANSFER CONTROL XCTL statement, optionally specifying a parameter list.

Because control is transferred, there is no need to perform the IDMS-STATUS routine.

COBOL programmers: If you specify a parameter list, the specified data items must be defined in the LINKAGE SECTION of both the calling and the receiving programs.

PL/I programmers: If you specify a parameter list, the specified data items must be defined as based storage in both the calling and the receiving programs. For further considerations related to this subject, see Appendix A, “PL/I Considerations” on page A-1.

Example of transferring control laterally: The program excerpt below shows a TRANSFER CONTROL request that includes a parameter list containing database retrieval information.

The ERRCHEK program performs error checking and passes control to the GETPROG program, which performs the database access.

```

PROGRAM-ID.                ERRCHK.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 GETPROG                  PIC X(8) VALUE 'GETPROG'.
LINKAGE SECTION.
01 PASS-DEPT-INFO.
   05 PASS-DEPT-ID          PIC 9(4).
   05 PASS-DEPT-INFO-END   PIC X.
PROCEDURE DIVISION.
  BIND MAP SOLICIT.
  BIND MAP SOLICIT RECORD SOLICIT-REC.
  MAP IN USING SOLICIT.
*** PERFORM ERROR CHECKING ***
  IF SOLICIT-DEPT-ID NOT NUMERIC
    THEN GO TO SOLICIT-ERROR.
*** ACQUIRE STORAGE FOR DEPT-ID TO BE PASSED ***
  GET STORAGE FOR PASS-DEPT-INFO TO PASS-DEPT-INFO-END
  WAIT LONG USER KEEP STGID 'PDIN'
  ON DC-NEW-STORAGE NEXT SENTENCE.
  MOVE SOLICIT-DEPT-ID TO PASS-DEPT-ID.
*** TRANSFER CONTROL TO DATABASE ACCESS PROGRAM ***
  TRANSFER CONTROL TO GETPROG XCTL
    USING PASS-DEPT-INFO.

```

```

PROGRAM-ID.                GETPROG.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 GETPROG                  PIC X(8) VALUE 'GETPROG'.
LINKAGE SECTION.
01 P-DEPT-INFO.
   05 P-DEPT-ID            PIC 9(4).
   05 P-DEPT-INFO-END     PIC X.
PROCEDURE DIVISION USING P-DEPT-INFO.
COPY IDMS SUBSCHEMA-BINDS.
  READY.
  MOVE P-DEPT-ID TO DEPT-ID-0410.
*** OBTAIN DEPARTMENT USING PASSED DEPT-ID ***
  OBTAIN DEPARTMENT CALC
  ON DB-REC-NOT-FOUND
  PERFORM ERR-NO-DEPT.
.
*** FURTHER DATABASE PROCESSING ***

```

9.2.3 Passing control, expecting to return

To transfer program control to a subordinate level, expecting return of control to the instruction immediately following the request, perform the following steps:

1. Invoke the main program specified by the task code.
2. Perform processing, as required.
3. Transfer control to the second program by issuing a TRANSFER CONTROL LINK statement, optionally specifying a parameter list.
4. Perform processing in the subordinate-level program, as required. DC returns control to the next-higher-level program when the subordinate program issues a DC RETURN statement.

Example of passing control to a lower level: The program excerpt below transfers control to DEPTCHEK, a subroutine that performs error-checking.

The GETPROG program performs processing based on the status returned by the DEPTCHEK program.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DEPTCHEK                PIC X(8) VALUE 'DEPTCHEK'.
01 ERRCHEK-INFO.
   05 CHEK-DEPT-ID        PIC 9(4).
   05 CHEK-ERRSTAT       PIC X(4).
PROCEDURE DIVISION.
  BIND MAP SOLICIT.
  BIND MAP SOLICIT RECORD SOLICIT-REC.
  MAP IN USING SOLICIT.
  MOVE SOLICIT-DEPT-ID TO CHEK-DEPT-ID.
  MOVE 'OK' TO CHEK-ERRSTAT.
*** TRANSFER CONTROL TO ERROR CHECKING PROGRAM ***
TRANSFER CONTROL TO DEPTCHEK LINK USING
      CHEK-DEPT-ID
      CHEK-ERRSTAT.
  IF CHEK-ERRSTAT NOT = 'OK'
    GO TO ERR-DEPT-ID.
  COPY IDMS SUBSCHEMA-BINDS.
  READY.
  MOVE SOLICIT-DEPT-ID TO DEPT-ID-0410.
  OBTAIN DEPARTMENT CALC
    ON DB-REC-NOT-FOUND
    PERFORM ERR-NO-DEPT.
*** FURTHER DATABASE PROCESSING ***
```

```
PROGRAM-ID.                DEPTCHEK.
DATA DIVISION.
LINKAGE SECTION.
01 CH-DEPT-INFO.
   05 CH-ID                PIC 9(4).
   05 CH-ERRSTAT          PIC X(4).
PROCEDURE DIVISION USING CH-DEPT-INFO.
*** PERFORM ERROR AND RANGE CHECKING ***
  IF CH-ID NOT NUMERIC
    THEN MOVE 'NNUM' TO CH-ERRSTAT
  ELSE
    IF CH-ID > 8000 OR < 1000
      MOVE 'RANG' TO CH-ERRSTAT.
*** RETURN CONTROL TO CALLING PROGRAM ***
  DC RETURN.
```

9.3 Retrieving task-related information

DC provides task- and system-related information that you can use in your program. Although you can use this information for any number of purposes, it is most often used for the following:

- **Program flexibility** — You can perform various chapters of code based on the calling task code.
- **Operator information** — You can display the logical terminal ID, the physical terminal ID, and the current DC system number on the terminal screen. The program excerpt below shows this technique.
- **Journaling information** — You can write information such as the user ID, logical terminal ID, the physical terminal ID, and the current DC system number to the journal file. For more information, see 9.7, “Writing to the journal file” on page 9-23 later in this chapter.
- **System security** — You can restrict program access based on site-specific factors. For example, you can permit only certain tasks or certain terminals to access a specified program.

Using the ACCEPT statement: To retrieve task- and system-related information, issue an ACCEPT statement that indicates the information needed and the variable-storage location to which it is to be returned.

Example of retrieving task information: The program excerpt below uses ACCEPT statements to retrieve the task code, the logical terminal ID, the physical terminal ID, and the user ID.

9.3 Retrieving task-related information

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DEPTDISM                PIC X(8)  VALUE 'DEPTDISM'.
01 SOLICIT-REC.
   05 SOLICIT-DEPT-ID      PIC X(4).
   05 TASK-INFO.
      07 TC                PIC X(8).
      88 GETOUT            VALUE 'DEPTBYE'.
      07 LTERMINAL         PIC X(8).
      07 PTERMINAL        PIC X(8).
      07 CURR-USER         PIC X(32).
PROCEDURE DIVISION.
*** RETRIEVE THE TASK CODE ***
  ACCEPT TASK CODE INTO TC.
*** IF TASK CODE = DEPTBYE, RETURN TO CA-IDMS/DC ***
  IF GETOUT DC RETURN.
  BIND MAP SOLICIT.
  BIND MAP SOLICIT RECORD SOLICIT-REC.
*** RETRIEVE LTERM, PTERM, AMD USER ID ***
  ACCEPT LTERM ID INTO LTERMINAL.
  ACCEPT PTERM ID INTO PTERMINAL.
  ACCEPT USER ID INTO CURR-USER.
  MOVE ZERO TO SOLICIT-DEPT-ID.
  MAP OUT USING SOLICIT
  NEWPAGE
  MESSAGE IS INITIAL-MESSAGE LENGTH 80.
*
DC RETURN
NEXT TASK CODE DEPTDISM.
```

The mapout performed by the program excerpt results in this screen display:

```
LTERM:  LT12014                                PTERM:  PV12014
                                                USER:  RKN

*** DEPARTMENT SOLICITOR SCREEN ***

DEPARTMENT ID:  0000

ENTER AN DEPT ID AND PRESS ENTER ** CLEAR TO EXIT
```

9.4 Maintaining data integrity in the online environment

To maintain database integrity in the online environment, DC allows you to perform the following functions:

- **Place an explicit lock on a database record** — You can restrict other run units' access to a specified database record occurrence.
- **Monitor concurrent database access across a pseudoconverse** — You can determine if other run units have accessed a certain database record during a pseudoconverse.

9.4.1 Setting longterm explicit locks

In pseudoconversational programming, you may be required to lock records across run units for the duration of a transaction. For example, a high-priority update application may lock record occurrences as they are retrieved in order to prevent other run units from accessing data that is about to be modified.

Steps to set longterm locks: To lock a database record explicitly across a pseudoconverse, perform the following steps:

1. Retrieve the database record.
2. Issue a `KEEP LONGTERM` statement that specifies either the `SHARE CURRENT` or the `EXCLUSIVE CURRENT` parameter:
 - **SHARE CURRENT** places a shared lock on the specified record occurrence; other run units can access the record but not update it.
 - **EXCLUSIVE CURRENT** places an exclusive lock on the specified record occurrence; other run units cannot access the record in any way.
3. Perform pseudoconversational processing, as required.
4. As soon as possible, release the explicit lock by issuing a `KEEP LONGTERM` statement with the `RELEASE` parameter.

Important: Release longterm locks as soon as possible to provide availability to other run units.

Interaction of longterm locks:

Locks in effect	Locks allowed for other run units	Locks disallowed for other run units
Shared	Shared and longterm shared	Exclusive and longterm exclusive
Exclusive	None	Shared, exclusive, longterm shared, and longterm exclusive
Longterm shared	For all run units: shared and longterm shared For run units on the same terminal: exclusive and longterm exclusive	For run units on other terminals: exclusive and longterm exclusive
Longterm exclusive	For run units on the same terminal: shared, exclusive, longterm shared, and longterm exclusive	For run units on other terminals: shared, exclusive, longterm shared, and longterm exclusive

Example of setting longterm exclusive locks: The first program excerpt below sets longterm exclusive locks in order to ensure that other programs cannot access any data. (The second program excerpt performs database modifications and releases the locks as soon as possible.)

The first program excerpt locks the EMPLOYEE and DEPARTMENT records in order to prevent other run units from modifying them during the pseudoconverse.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 CHNGDEPT                PIC X(8)   VALUE 'CHNGDEPT'.
01 KEEP-INFO.
   05 DEPT-LNGTRM-ID       PIC X(4)   VALUE 'DEPT'.
   05 EMPL-LNGTRM-ID       PIC X(4)   VALUE 'EMPL'.
01 MAP-WORK-REC.
   05 WORK-OLD-DEPT-ID     PIC 9(4).
   05 WORK-NEW-DEPT-ID     PIC 9(4).
   05 WORK-EMP-ID          PIC 9(4).
   05 WORK-FIRST           PIC X(10).
   05 WORK-LAST            PIC X(15).
   05 WORK-ADDRESS         PIC X(42).
PROCEDURE DIVISION.
  BIND MAP DCTEST03.
  BIND MAP DCTEST03 RECORD MAP-WORK-REC.
  MAP IN USING DCTEST03.
  MOVE WORK-EMP-ID TO EMP-ID-0415.
  OBTAIN CALC EMPLOYEE
    ON DB-REC-NOT-FOUND GO TO ERR-NO-EMP.
*** SET AN EXCLUSIVE LOCK ON THE CURRENT EMPLOYEE RECORD ***
  KEEP LONGTERM EMPL-LNGTRM-ID
  EXCLUSIVE CURRENT EMPLOYEE.
  MOVE EMP-ID-0415 TO WORK-EMP-ID.
  MOVE EMP-LAST-NAME-0415 TO WORK-LAST.
  MOVE EMP-FIRST-NAME-0415 TO WORK-FIRST.
  MOVE EMP-ADDRESS-0415 TO WORK-ADDRESS.
  IF DEPT-EMPLOYEE IS NOT EMPTY
    OBTAIN OWNER IN DEPT-EMPLOYEE
  ELSE GO TO NO-DEPT.
*** SET AN EXCLUSIVE LOCK ON THE CURRENT DEPARTMENT RECORD ***
  KEEP LONGTERM DEPT-LNGTRM-ID
  EXCLUSIVE CURRENT DEPARTMENT.
  MOVE DEPT-ID-0410 TO WORK-OLD-DEPT-ID.
*** ALLOW INPUT IN THE NEW DEPARTMENT FIELD ONLY ***
  MODIFY MAP DCTEST03 FOR ALL EXCEPT
    DFLD WORK-NEW-DEPT-ID
    ATTRIBUTES PROTECTED.
  MAP OUT USING DCTEST03.
  DC RETURN NEXT TASK CODE CHNGDEPT.

```

Example of releasing longterm exclusive locks: This program excerpt maps in the new department ID, disconnects the employee from the old department, and connects the record to the new department.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 CHNGSHOW                PIC X(8)   VALUE 'CHNGSHOW'.
01 TEMP-DEPT-DBKEY         PIC S9(8)  COMP.
01 KEEP-INFO.
   05 DEPT-LNGTRM-ID       PIC X(4)   VALUE 'DEPT'.
   05 EMPL-LNGTRM-ID       PIC X(4)   VALUE 'EMPL'.
01 MAP-WORK-REC.
   05 WORK-OLD-DEPT-ID     PIC 9(4).
   05 WORK-NEW-DEPT-ID     PIC 9(4).
   05 WORK-EMP-ID          PIC 9(4).
   05 WORK-FIRST           PIC X(10).
   05 WORK-LAST            PIC X(15).
   05 WORK-ADDRESS         PIC X(42).
PROCEDURE DIVISION.
   BIND MAP DCTEST03.
   BIND MAP DCTEST03 RECORD MAP-WORK-REC.
   MAP IN USING DCTEST03.
   IF WORK-NEW-DEPT-ID IS NOT NUMERIC
     GO TO ERR-NONNUMERIC-DEPT-ID.
*** OBTAIN NEW DEPARTMENT RECORD TO ENSURE IT EXISTS ***
   MOVE WORK-NEW-DEPT-ID TO DEPT-ID-0410.
   FIND CALC DEPARTMENT
     ON DB-REC-NOT-FOUND GO TO ERR-NO-NEW-DEPT.
   MOVE DBKEY TO TEMP-DEPT-DBKEY.
*** REOBTAIN OLD DEPARTMENT ***
   MOVE WORK-OLD-DEPT-ID TO DEPT-ID-0410.
   FIND CALC DEPARTMENT.
*** REOBTAIN EMPLOYEE RECORD ***
   MOVE WORK-EMP-ID TO EMP-ID-0415.
   FIND CALC EMPLOYEE.
   DISCONNECT EMPLOYEE FROM DEPT-EMPLOYEE.
*** REOBTAIN NEW DEPARTMENT USING SAVED DB-KEY ***
   FIND DEPARTMENT USING TEMP-DEPT-DBKEY.
   CONNECT EMPLOYEE TO DEPT-EMPLOYEE.
*** RELEASE ALL LONGTERM LOCKS ***
   KEEP LONGTERM ALL RELEASE.
   MAP OUT USING DCTEST03 OUTPUT DATA IS ATTRIBUTE
     MESSAGE IS EMP-CONNECTED-MESS LENGTH 80.
   DC RETURN NEXT TASK CODE CHNGSHOW.

```

9.4.2 Monitoring concurrent database access

You can monitor concurrent database access associated with a specific record during a pseudoconverse, instead of locking the record. In most cases, monitoring is preferable to locking because it allows other run units unrestricted access to the specified database record.

Pageable map applications: Because you cannot predict the number of occurrences that will be accessed and displayed on a pageable map, it is especially useful to monitor, rather than lock, such records.

Steps before the pseudoconverse: To monitor concurrent database access across a pseudoconverse, perform the following steps:

1. Request DC to begin monitoring database concurrent access for the specified record occurrence by issuing a **KEEP LONGTERM** statement that includes the **NOTIFY** parameter.

2. Begin the pseudoconverse by issuing a DC RETURN statement.

Steps after the pseudoconverse: In subsequent tasks, perform the following steps:

1. Determine if the record has been accessed by another run unit by issuing a KEEP LONGTERM statement with the TEST parameter. The components of the value returned as a result of the KEEP LONGTERM TEST statement are as follows:
 - **0** — The record was not accessed.
 - **1** — The record was obtained.
 - **2** — The record was modified.
 - **4** — The record's prefix was modified by a CONNECT or DISCONNECT operation.
 - **8** — The record was logically deleted.
 - **16** — The record was physically deleted.
 - **32** — The status of the record is uncertain.

For example, a value of 9 means that the record was obtained and logically deleted; the highest possible value is 31, which indicates that all the above actions were performed. You should proceed according to the effect that other run units' processing has on your application and the extent of the other run units' processing.

Typically, you should require the user to resubmit any transaction in which another run unit has modified a record's data.

Pageable map applications: You should be aware of the effect modified detail occurrences have on each other when using longterm notify locks. For example, if you are modifying a series of records that participate in the same occurrence of a sorted set, a value of 5 (obtained and modified by DISCONNECT/CONNECT) is returned beginning with the second modified detail occurrence.

2. If necessary, issue a KEEP LONGTERM statement with the UPGRADE parameter to place a longterm explicit lock on the specified record.
3. Access the database, as required.
4. Finish longterm monitoring and release longterm locks by issuing a KEEP LONGTERM statement with the RELEASE parameter.

Data sharing considerations: A data sharing environment allows programs executing on more than one CA-IDMS system to concurrently access and update data in the same areas of the data base. In order to do this, such systems must be members of a data sharing group.

KEEP LONGTERM DML statements will control or monitor data access across members of a data sharing group just as they do within a single CA-IDMS system. Programs do not need to be concerned with whether or not the data is being shared

between members, with one exception: the retrieval of data is not monitored between members. This means that if a program executing on one member issues a KEEP LONGTERM NOTIFY statement and a program on another member subsequently obtains (but does not update) the affected record, then no indication of the retrieval will be returned to the monitoring program when it checks to see what access has taken place using the KEEP LONGTERM TEST statement. If the accessing program updates the record, the notification value returned to the monitoring program will be an even number greater than 1.

Example of establishing longterm monitoring: The first program excerpt below uses the NOTIFY option of the KEEP LONGTERM statement to monitor concurrent database access across a pseudoconverse. (The second program excerpt performs processing based on the result of database monitoring.)

The first program excerpt uses the NOTIFY option of the KEEP LONGTERM statement to establish monitoring of other run units' access to the specified EMPLOYEE record. It uses the employee's CALC key as the longterm ID.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 EMPMOD                                PIC X(8)    VALUE 'EMPMOD'.
01 KEEP-INFO.
   05 KEEP-LNGTRM-ID                      PIC X(4).
01 MAP-WORK-REC.
   05 WORK-EMP-ID                          PIC 9(4).
   05 WORK-FIRST                           PIC X(10).
   05 WORK-LAST                            PIC X(15).
   05 WORK-ADDRESS                         PIC X(42).
PROCEDURE DIVISION.
  BIND MAP DCTEST03.
  BIND MAP DCTEST03 RECORD MAP-WORK-REC.
.
.
.
  OBTAIN CALC EMPLOYEE
    ON DB-REC-NOT-FOUND GO TO ERR-NO-EMP.
*** USE EMPLOYEE'S CALC KEY FOR THE LONGTERM ID ***
  MOVE EMP-ID-0415 TO KEEP-LNGTRM-ID.
*** BEGIN MONITORING ***
  KEEP LONGTERM KEEP-LNGTRM-ID
  NOTIFY CURRENT EMPLOYEE.
  MOVE EMP-ID-0415 TO WORK-EMP-ID.
  MOVE EMP-LAST-NAME-0415 TO WORK-LAST.
  MOVE EMP-FIRST-NAME-0415 TO WORK-FIRST.
  MOVE EMP-ADDRESS-0415 TO WORK-ADDRESS.
  MAP OUT USING DCTEST03.
  DC RETURN NEXT TASK CODE EMPMOD.

```

Monitoring concurrent database access: The program excerpt below checks to determine if any other run units have accessed the specified record. If any modifications have been made, the program issues a ROLLBACK and informs the user. If no modifications have been made, the program locks the record by issuing a KEEP LONGTERM UPGRADE statement before performing database access and modification.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 REDISP          PIC X(8)  VALUE 'REDISPLY'.
01 KEEP-INFO.
   05 KEEP-LNGTRM-ID PIC X(4)  VALUE 'KPID'.
   05 KL-STAT        PIC S9(8)  COMP.
01 MAP-WORK-REC.
   05 WORK-EMP-ID    PIC 9(4).
   05 WORK-FIRST     PIC X(10).
   05 WORK-LAST      PIC X(15).
   05 WORK-ADDRESS   PIC X(42).
PROCEDURE DIVISION.
  BIND MAP DCTEST03.
  BIND MAP DCTEST03 RECORD MAP-WORK-REC.
  MAP IN USING DCTEST03.
  MOVE WORK-EMP-ID TO EMP-ID-0415.
  OBTAIN CALC EMPLOYEE
    ON DB-REC-NOT-FOUND GO TO ERR-NO-EMP.
  MOVE EMP-ID-0415 TO KEEP-LNGTRM-ID.
*** TEST TO SEE IF OTHER RUN UNITS HAVE ACCESSED THE RECORD ***
  KEEP LONGTERM KEEP-LNGTRM-ID
    TEST RETURN NOTIFICATION INTO KL-STAT.
*** A RETURNED VALUE THAT IS GREATER THAN 1 MEANS ***
*** THAT THE RECORD WAS MODIFIED IN SOME WAY. ***
*** ROLLBACK AND REQUIRE THE OPERATOR TO RESUBMIT ***
*** NOTE: THE SIGNIFICANCE OF THE RETURNED ***
*** VALUE IS APPLICATION-SPECIFIC. ***
*** FOR EXAMPLE, FOR SOME APPLICATIONS ***
*** A RETURNED VALUE > 1 MAY BE ***
*** ACCEPTABLE, FOR OTHERS, IT MAY NOT. ***
  IF KL-STAT > 1
    ROLLBACK TASK CONTINUE
    MAP OUT USING DCTEST03 DATA IS ATTRIBUTE
    MESSAGE IS EMPMOD-MESS LENGTH 40
    DC RETURN DEXT TASK CODE REDISP
*** OTHERWISE UPGRADE THE LOCK TO SHARED ***
  ELSE
    KEEP LONGTERM KEEP-LNGTRM-ID
    UPGRADE SHARE.
*** DATABASE UPDATE PROCESSING ***

```

9.5 Managing tables

At runtime, your program can request DC to load a table (for example, an edit or code table) from either the DDLDCLOD area or a load (core-image) library into the program pool. This load does not imply automatic execution; your program continues to run. Typically, you use this function to place nonexecutable data in the program pool.

Making tables nonoverlayable: By default, tables and other programs loaded into the program pool can be overlaid when not in use or when in use and waiting for an event. However, unlike an executable module, a table is not reloaded during program execution if it has been overlaid. Therefore, you should define the table with the nonoverlayable attribute during system generation (or at runtime with a DCMT VARY DYNAMIC PROGRAM command) so that it cannot be overlaid before the program deletes it.

Deleting tables: When your program requests DC to delete a table, it does not physically delete that table; rather, it decrements the in-use counter maintained by DC. An in-use count of 0 signals DC that the space occupied by the table can be reused. When your task terminates, DC automatically deletes any tables that have not been explicitly deleted.

If your task requests a nonreentrant table more than once, DC loads a new copy of the table for each request and adds 1 to the in-use counter; each copy corresponds to a separate location in program variable storage. If your task loads the same reentrant or quasireentrant table more than once, it must delete that table the same number of times in order to set the in-use counter to 0.

Steps to load and delete a table: To load a table into the program pool and later delete it, perform the following steps:

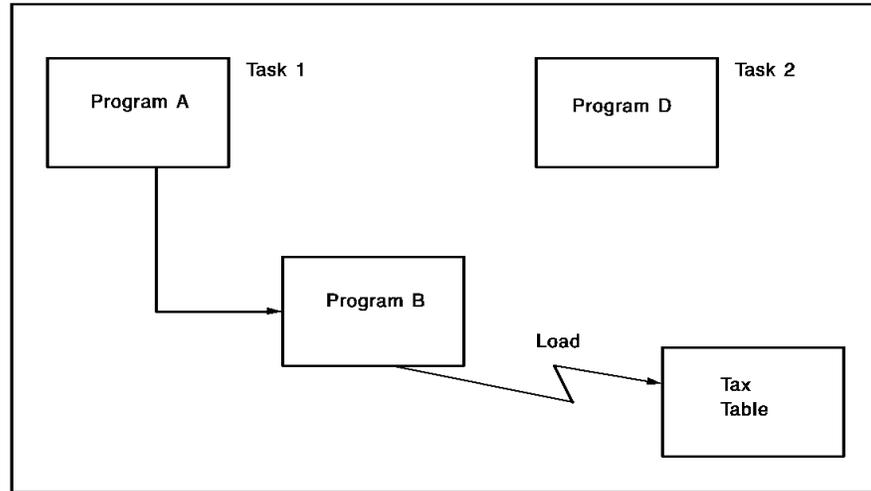
1. Request DC to load the table into the program pool by issuing a LOAD TABLE statement.
2. Perform processing, using the table as needed.
3. When processing is complete, decrement the table's in-use counter by issuing a DELETE TABLE statement.

Note: You can qualify the name of the table by providing the DICTNAME, DICTNODE, or LOADLIB parameter on the LOAD or DELETE statement.

Illustration of table management: Assume that two tasks are executing under a DC system. Task 1 consists of programs A and B; task 2 consists of program D. The following diagrams illustrate how the tasks load and delete a table:

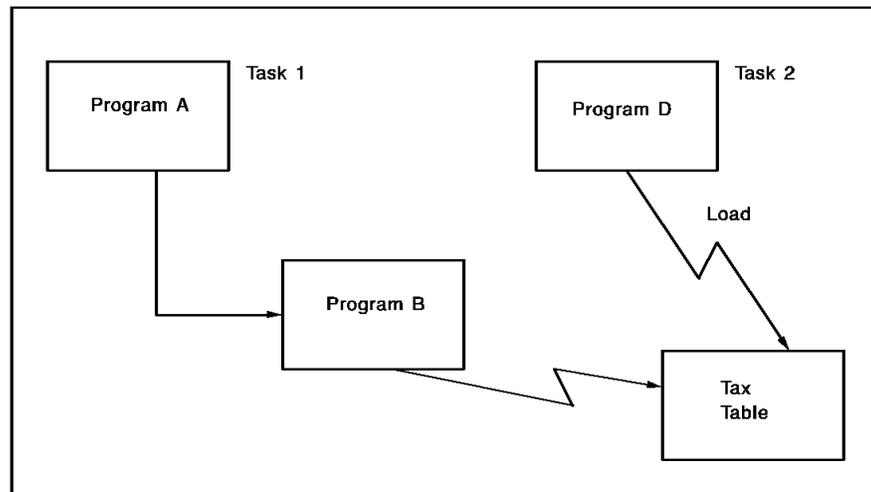
1. Program B, which is in control of task 1, loads a tax table. Program B continues to execute; DC loads the table into the program pool.

Program pool



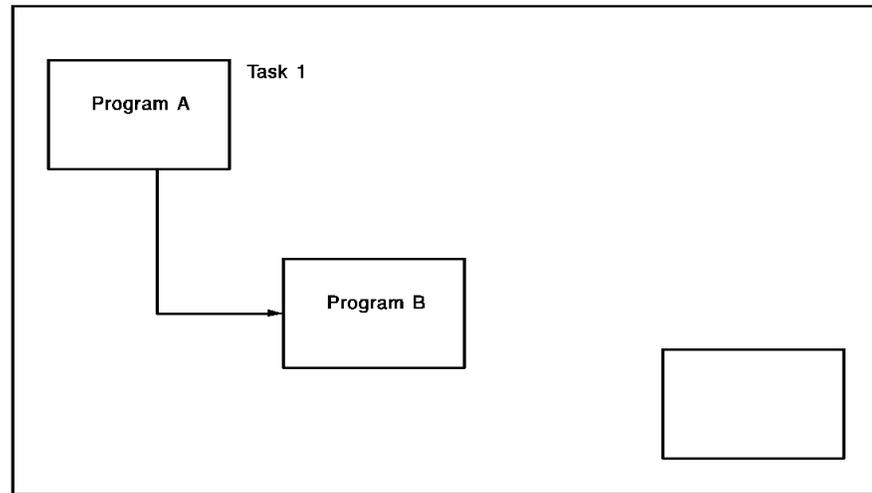
2. Program D, which is in control of task 2, loads the same tax table. Because a copy of the table exists in the program pool and is available (concurrent and not overlaid during a temporary wait), the load is completed with no physical I/O. When task 2 terminates, the table remains in the program pool, as task 1 requires its use.

Program pool



3. Task 1 deletes (signals completion of use) the table. The table remains in the program pool but its in-use counter is set to 0; its storage is now freed for use by other programs.

Program pool



Example of loading and deleting a table: The program excerpt below loads a sales tax table into the LINKAGE SECTION and computes the tax for all items in a specified order. When processing is complete, it decrements the table's in-use count by issuing a DELETE TABLE command.

```

PROGRAM-ID.                SALESTAX.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 SALES-TRANS-COUNT        PIC S9(5) COMP-3.
LINKAGE SECTION.
01 SALES-TAX-TABLE.
   02 STATE-AND-TAX         OCCURS 50 TIMES.
       05 STATE-ABB         PIC XX.
       05 STATE-SALES-TAX   PIC SV999.
   02 SALES-TAX-TABLE-END   PIC X.
PROCEDURE DIVISION.
.
.
.
*** LOAD THE SALES TAX TABLE INTO THE LINKAGE SECTION ***
    LOAD TABLE 'SALESTAX' INTO
        SALES-TAX-TABLE TO SALES-TAX-TABLE-END.
    PERFORM A100-COMPUTE-TAX UNTIL SALES-TRANS-COUNT = 0.
*** DECREMENT THE TABLE'S IN-USE COUNT ***
    DELETE TABLE FROM SALES-TAX-TABLE.
  
```

9.6 Retrieving the current time and date

DC allows you to obtain the current time and date from the operating system. You can use these values either for screen display or for journaling purposes.

►► For more information on journaling, see 9.7, “Writing to the journal file” on page 9-23 later in this chapter.

To obtain the current time and date, issue a GET TIME statement that specifies the variable-storage location into which DC is to return the current time and, optionally, the current date.

Example of obtaining the current time and date: The program excerpt below obtains the time and date for display on the terminal screen.

It obtains the current time in edit format (*hh:mm:ss:hhh*) and the current date in fixed binary format. You must change the date to display format in order to display it on the terminal screen.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 DEPTDISM                PIC X(8)  VALUE 'DEPTDISM'.
01 SOLICIT-REC.
   05 SOLICIT-DEPT-ID      PIC X(4).
   05 TASK-INFO.
      07 TC                PIC X(8).
      88 GETOUT            VALUE 'DEPTBYE'.
      07 LTERMINAL         PIC X(8).
      07 PTERMINAL         PIC X(8).
      07 CURR-USER         PIC X(32).
      07 CURR-TIME         PIC X(11).
      07 SYS-DATE          PIC 9(7) COMP-3.
      07 CURR-DATE         PIC 9(5).
PROCEDURE DIVISION.
ACCEPT TASK CODE INTO TC.
IF GETOUT DC RETURN.
BIND MAP SOLICIT.
BIND MAP SOLICIT RECORD SOLICIT-REC.
*
ACCEPT LTERM ID INTO LTERMINAL.
ACCEPT PTERM ID INTO PTERMINAL.
ACCEPT USER ID INTO CURR-USER.
*** GET THE CURRENT TIME AND DATE ***
GET TIME INTO CURR-TIME EDIT
DATE INTO SYS-DATE.
*** CHANGE THE DATE TO DISPLAY FORMAT ***
MOVE SYS-DATE TO CURR-DATE.
MOVE ZERO TO SOLICIT-DEPT-ID.
MAP OUT USING SOLICIT
NEWPAGE
MESSAGE IS INITIAL-MESSAGE LENGTH 80.
*
DC RETURN
NEXT TASK CODE DEPTDISM.

```

Example of displaying current time and date:

The mapout in the program excerpt results in this screen display:

```
LTERM:  LT12002                                PTERM:  PV12002
                                                USER:  RKN

          *** DEPARTMENT SOLICITOR SCREEN ***

TIME:  11:49:45.60                                DATE:  86.037

          DEPARTMENT ID:  0000

ENTER AN DEPT ID AND PRESS ENTER ** CLEAR TO EXIT
```

9.7 Writing to the journal file

You can write information to the DC journal file to document run-unit related information. For example, you could write to the journal for the following reasons:

- Your **site standards** may require that you record journal information at certain points in a program (for example, when signing on or off).
- You can facilitate **debugging** by writing records to the journal file. For example, as a debugging aid, you can write duplicate scratch and queue entries to the journal file because such records are deleted during ROLLBACK processing.

Steps to write to the journal file: To write to the journal file, perform the following steps:

1. Initialize the variable-storage area from which you will write to the journal file.
2. Issue a WRITE JOURNAL statement that specifies the appropriate variable-storage location.

Example of writing to the journal file: The program excerpt below writes the current task code, logical-terminal ID, physical-terminal ID, user ID, time, and date to the journal file.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 SOLICIT-REC.
   05 SOLICIT-DEPT-ID          PIC X(4).
   05 TASK-INFO.
      07 TC                    PIC X(8).
      88 GETOUT                VALUE 'DEPTBYE'.
      07 LTERMINAL             PIC X(8).
      07 PTERMINAL             PIC X(8).
      07 CURR-USER             PIC X(32).
      07 CURR-TIME             PIC X(11).
      07 SYS-DATE              PIC 9(7) COMP-3.
      07 CURR-DATE             PIC 9(5).
      07 TASK-INFO-END        PIC X.
PROCEDURE DIVISION.
*** RETRIEVE TASK CODE ***
ACCEPT TASK CODE INTO TC.
IF GETOUT DC RETURN.
BIND MAP SOLICIT.
BIND MAP SOLICIT RECORD SOLICIT-REC.
*** RETRIEVE LTERM ID, PTERM ID, AND USER ID ***
ACCEPT LTERM ID INTO LTERMINAL.
ACCEPT PTERM ID INTO PTERMINAL.
ACCEPT USER ID INTO CURR-USER.
*** RETRIEVE CURRENT TIME AND DATE ***
GET TIME INTO CURR-TIME EDIT
DATE INTO SYS-DATE.
MOVE SYS-DATE TO CURR-DATE.
MOVE ZERO TO SOLICIT-DEPT-ID.
*** WRITE DATA TO THE JOURNAL FILE ***
WRITE JOURNAL FROM TASK-INFO TO TASK-INFO-END
NOWAIT SPAN.
MAP OUT USING SOLICIT
NEWPAGE
MESSAGE IS INITIAL-MESSAGE LENGTH 80.
*
DC RETURN
NEXT TASK CODE 'DEPTDISM'.
```

9.8 Collecting DC statistics

You can collect runtime statistics related to DC transactions on a logical terminal. This information can be useful both for debugging purposes and as an aid in determining overall program efficiency.

Steps to collect statistics: To collect runtime DC statistics related to the transactions performed on a logical terminal, perform the following steps:

1. Establish a 248-byte field in program variable storage in which to copy the transaction statistics.
2. Define the beginning of the transaction by issuing a `BIND TRANSACTION STATISTICS` statement.
3. Perform pseudoconversational processing, as required.
4. Copy the contents of the transaction statistics block (TSB) into the specified location in variable storage and, optionally, to the DC log file by issuing an `ACCEPT TRANSACTION STATISTICS` statement.
5. When processing is complete, terminate statistics collection by issuing an `END TRANSACTION STATISTICS` statement, optionally writing the statistics to variable storage and the DC log file.

Example of collecting transaction statistics: Depending on the invoking task, the program excerpt below initiates statistics collection, copies the TSB to the DC log file, or terminates statistics collection and displays selected statistics on the terminal screen.

```
DATA DIVISION
WORKING-STORAGE SECTION.
01 TASKCODE                PIC X(8).
   88 FIRSTTIME            VALUE 'INIT'.
   88 SECONDTIME           VALUE 'TRANS'.
   88 FINALTIME            VALUE 'TERMSESS'.
01 STATISTICS-BLOCK.
   05 USER-ID              PIC X(32).
   05 LTERM-ID             PIC X(8).
   05 PROG-CALL            PIC S9(8) COMP.
   05 PROG-LOAD            PIC S9(8) COMP.
   05 TERM-READ            PIC S9(8) COMP.
   05 TERM-WRITE          PIC S9(8) COMP.
   05 TERM-ERROR          PIC S9(8) COMP.
   05 STORAGE-GET         PIC S9(8) COMP.
   05 SCRATCH-GET         PIC S9(8) COMP.
   05 SCRATCH-PUT         PIC S9(8) COMP.
   05 SCRATCH-DEL         PIC S9(8) COMP.
   05 QUEUE-GET           PIC S9(8) COMP.
   05 QUEUE-PUT           PIC S9(8) COMP.
   05 QUEUE-DEL           PIC S9(8) COMP.
   05 GET-TIME            PIC S9(8) COMP.
   05 SET-TIME            PIC S9(8) COMP.
   05 DB-CALLS            PIC S9(8) COMP.
   05 MAX-STACK           PIC S9(8) COMP.
   05 USER-TIME           PIC S9(8) COMP.
   05 SYS-TIME            PIC S9(8) COMP.
   05 WAIT-TIME           PIC S9(8) COMP.
   05 PAGES-READ          PIC S9(8) COMP.
   05 PAGES-WRIT         PIC S9(8) COMP.
   05 PAGES-REQ           PIC S9(8) COMP.
   05 CALC-NO             PIC S9(8) COMP.
   05 CALC-OF             PIC S9(8) COMP.
   05 VIA-NO              PIC S9(8) COMP.
   05 VIA-OF              PIC S9(8) COMP.
   05 RECS-REQ            PIC S9(8) COMP.
   05 RECS-CURR          PIC S9(8) COMP.
   05 FILLER              PIC X(4).
   05 FRAG-STORED         PIC S9(8) COMP.
   05 RECS-RELO           PIC S9(8) COMP.
   05 TOT-LOCKS           PIC S9(8) COMP.
   05 SEL-LOCKS           PIC S9(8) COMP.
   05 UPD-LOCKS           PIC S9(8) COMP.
   05 STG-HI-MARK         PIC S9(8) COMP.
   05 FREESTG-REQ        PIC S9(8) COMP.
   05 SYS-SERV            PIC S9(8) COMP.
   05 RESERVED            PIC X(40).
   05 USER-SUPP-ID        PIC X(8).
   05 BIND-DATE           PIC S9(7) COMP-3.
```

```
01 STAT-DIS.
  05 WORK-CURR-DATE          PIC 9(5).
  05 WORK-USER-ID           PIC X(32).
  05 WORK-DB-CALLS          PIC 9(4).
  05 WORK-WAIT-TIME         PIC 9(12).
  05 WORK-PAGES-READ        PIC 9(5).
  05 WORK-PAGES-WRIT        PIC 9(5).
PROCEDURE DIVISION.
  BIND MAP STATMAP.
  BIND MAP STATDIS RECORD STATISTICS-BLOCK.
*
  ACCEPT TASK CODE INTO TASKCODE.
*** FIRST TIME, INITIATE STATISTICS COLLECTION ***
  IF FIRSTTIME
    BIND TRANSACTIONS STATISTICS
    DC RETURN.
*** SUBSEQUENT TIMES, COPY STATISTICS TO VARIABLE STORAGE ***
  IF SECONDTIME
    ACCEPT TRANSACTION STATISTICS
    WRITE INTO STATISTICS-BLOCK
    DC RETURN.
*** LAST TIME, END STATISTICS COLLECTION AND ***
*** COPY STATISTICS TO VARIABLE STORAGE ***
  IF FINALTIME
    END TRANSACTION STATISTICS
    WRITE INTO STATISTICS-BLOCK
    PERFORM U100-MOVE-FIELDS-TO MAP
    MAP OUT USING STATMAP
    MESSAGE IS STAT-DISPLAY-MESS LENGTH 40
    DC RETURN.
  DC RETURN.
```

9.9 Sending messages

DC provides message management functions that allow you to send messages to the following destinations:

- The log file and the current user, optionally terminating the program
- Other users, logical terminals, or destinations

Sending messages to the current user and to other users is discussed below.

9.9.1 Sending a message to the current user

You can send a message predefined in the DDLDCMSG area of the dictionary to the current user, the log file, or both. The message definition can also specify other destinations (for example, the user's console). Additionally, the specified message indicates the action to be taken after the message is written; such action can include the following:

- **Waiting for user reply** — DC does not return control to your task until it receives a reply from the user's console.
- **Abending the program** — DC abends your program, or, optionally, the DC system.
- **Continuing program execution** — DC returns control to your program, optionally writing a snap dump of specified resources.

Retrieving predefined messages: One typical use for dictionary-defined messages is to retrieve predefined messages from the DDLDCMSG area rather than include all possible messages in program variable storage.

Messages stored in the dictionary can contain symbolic parameters. Symbolic parameters, identified by an ampersand (&) followed by a two-digit number, can appear in any order within the message. Symbolic parameters provide flexibility in message management.

Steps to send a predefined message: To send a predefined message, perform the following steps:

1. If you are using symbolic parameters, initialize the appropriate variable-storage locations.
2. Issue a WRITE LOG statement that specifies the appropriate variable-storage locations for symbolic parameters, user reply, and message text.

Note: You can specify your own message prefix (to distinguish your messages from DC/UCF system messages) by using the MESSAGE PREFIX IS parameter on the WRITE LOG statement.

Example of sending a message from the dictionary: The program excerpt below uses the following message from the DDLDCMSG area of the dictionary:

```
INPUT DATA IS IN ERROR; DATA FIELD: &01 &02
```

The symbolic parameters allow you to transmit more meaningful messages.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 SYMBOLIC-PARAMETERS.
  03 ERR-1.
    05 ERR-1-TEXT          PIC X(15).
    05 ERR-1-END          PIC X.
  03 ERR-2.
    05 ERR-2-TEXT          PIC X(15).
    05 ERR-2-END          PIC X.
    03 ERR-DEPT-ID        PIC X(6)   VALUE 'DEPT-ID'.
    03 ERR-NONNUMERIC     PIC X(10)  VALUE 'NONNUMERIC'.
01 MESSAGES.
  05 MESSAGE-AREA.        PIC X(80).
  05 MESSAGE-AREA-END    PIC X.
PROCEDURE DIVISION.
  BIND MAP SOLICIT.
  BIND MAP SOLICIT RECORD SOLICIT-REC.
*
  MAP IN USING SOLICIT.
*** IF ERROR, INITIALIZE FIELDS FOR SYMBOLIC PARMS ***
  IF SOLICIT-DEPT-ID NOT NUMERIC
    THEN MOVE ERR-DEPT-ID TO ERR-TEXT-1
    MOVE ERR-NONNUMERIC TO ERR-TEXT-2
    GO TO SOLICIT-ERROR.
.
.
SOLICIT-ERROR.
*** USE WRITE LOG STATEMENT TO COPY ***
*** DICTIONARY MESSAGE WITH PARMS INTO ***
*** PROGRAM VARIABLE STORAGE ***
  WRITE LOG MESSAGE ID 9001080
    PARMS FROM ERR-1 TO ERR-1-END
    FROM ERR-2 TO ERR-2-END
    TEXT INTO MESSAGE-AREA TO MESSAGE-AREA-END
    TEXT IS ONLY.
*** MAP OUT USING MESSAGE FROM DATA DICTIONARY ***
  MAP OUT USING SOLICIT
    MESSAGE IS MESSAGE-AREA TO MESSAGE-AREA-END.
  DC RETURN
  NEXT TASK CODE 'DEPTDIS'.
```

9.9.2 Sending a message to other users

DC provides the facilities for you to send messages to another terminal or user or to a group of terminals or users defined as a destination during system generation.

To send a message to another user, perform the following steps:

1. Initialize the variable-storage location from which the message is to be sent.
2. Issue a SEND MESSAGE statement that specifies the message's destination.

Note: To conserve resources, it is best not to specify the ALWAYS parameter in conjunction with a group of users.

Example of sending a message to another user: The program below is called by other programs in order to send a message to a specified user.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
01 MESS-INFO.
   05 MESS-USER-ID          PIC X(32).
   05 MESS-TEXT             PIC X(79).
   05 MESS-TEXT-END        PIC X.
   05 MESS-INFO-END        PIC X.
PROCEDURE DIVISION.
*** ESTABLISH ADDRESSABILITY TO USER ID AND MESSAGE TEXT ***
   GET STORAGE  FOR MESS-INFO TO MESS-INFO-END
   KEEP SHORT USER STGID 'MSG1'
   ON DC-NEW-STORAGE NEXT SENTENCE.
*** SEND MESSAGE TO SPECIFIED USER ID ***
   SEND MESSAGE ONLY TO USER ID MESS-USER-ID
   FROM MESS-TEXT TO MESS-TEXT-END.
*
   FREE STORAGE STGID 'MSG1'.
   DC RETURN.
```

9.10 Writing to a printer

You can request DC to transmit data from a task to a printer; this allows you to print reports during online processing.

Steps to transmit data to a printer: To transmit data to a printer, perform the following steps:

1. Initialize the variable-storage location from which DC is to write the specified information.
2. Initiate the printing procedure by issuing a `WRITE PRINTER` statement that indicates the appropriate variable-storage location and report ID, and specifies the print class or destination.
3. Issue subsequent `WRITE PRINTER` statements that indicate the variable-storage location of the data and the report ID.
4. Optionally, you can indicate the end of a report by issuing a `WRITE PRINTER` statement that includes the `ENDRPT` parameter.

CA-IDMS queue: DC does not transmit data directly from program variable storage to the printer. Rather, data is passed to a queue maintained by DC, and from the queue to the printer. The data stream passed to the queue by the `WRITE PRINTER` request contains only data; DC adds the necessary line and device control characters when it writes the data to the printer.

Note: The `WRITE PRINTER` command is used extensively under the `DC-BATCH` operating mode. For more information, refer to Appendix C, “Batch Access to DC Queues and Printers” on page C-1.

Example of writing to a printer: The program excerpt below writes a report to the printer associated with print class 33. The report consists of the employee ID and name, old department ID, and new department ID for each employee assigned to a new department.

```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 CHNGSHOW                PIC X(8)   VALUE 'CHNGSHOW'.
01 PRINT-CLASS             PIC 999    VALUE 33.
01 PRINT-AREA.
   05 PRI-EMP-ID           PIC X(4).
   05 PRI-EMP-LNAME        PIC X(15).
   05 PRI-EMP-FNAME        PIC X(10).
   05 PRI-OLD-DEPT-ID      PIC X(4).
   05 PRI-NEW-DEPT-ID     PIC X(4).
   05 PRINT-AREA-END      PIC X.
01 TEMP-DEPT-DBKEY        PIC S9(8) COMP.
01 MAP-WORK-REC.
   05 WORK-PRI-CTR         PIC 99.
   05 WORK-OLD-DEPT-ID    PIC 9(4).
   05 WORK-NEW-DEPT-ID    PIC 9(4).
   05 WORK-EMP-ID         PIC 9(4).
   05 WORK-FIRST          PIC X(10).
   05 WORK-LAST           PIC X(15).
   05 WORK-ADDRESS        PIC X(42).
PROCEDURE DIVISION.
.
.
.
*** DISCONNECT EMPLOYEE FROM OLD DEPARTMENT ***
*** CONNECT EMPLOYEE TO NEW DEPARTMENT      ***
.
.
.
*** PRINT PROCESSING FOR EMP TRANSFER REPORT ***
*** IF COUNTER = ZERO, SPECIFY CLASS 33      ***
   IF MAP-PRI-CTR = 0
     WRITE PRINTER FROM PRINT-AREA
       TO PRINT-AREA-END
       REPORT ID 100
       CLASS 33
   ELSE
*** IF COUNTER > 50, GO TO NEW PAGE ***
   IF-MAP-PRI-CTR > 50
     WRITE PRINTER NEWPAGE FROM PRINT-AREA
       TO PRINT-AREA-END
       REPORT ID 100
*** OTHERWISE WRITE LINE ***
   ELSE
     WRITE PRINTER FROM PRINT-AREA
       TO PRINT-AREA-END
       REPORT ID 100.
   ADD 1 TO MAP-PRI-CTR.
   MAP OUT USING DCTEST03 OUTPUT DATA IS ATTRIBUTE
   MESSAGE IS EMP-CONNECTED-MESS LENGTH 80.
   DC RETURN NEXT TASK CODE CHNGSHOW.

```

9.11 Writing JCL to a JES2 internal reader

You can write JCL to a JES2 internal reader from a DC application program by issuing a WRITE PRINTER statement that specifies the CLASS parameter.

System prerequisites: For your program to write JCL to a JES2 internal reader, the system administrator must first take these steps:

1. Define in system generation a SYSOUT line, physical terminal, and logical terminal, using these guidelines:

```
ADD LINE physical-line TYPE IS SYSOUTL DDNAME IS ddname.  
ADD PTERM physical-terminal TYPE IS SYSOUTL PRINTER CLASS IS 0  
PAGE WIDTH IS 80.  
ADD LTERM logical-terminal PRINTER CLASS = ADD (nn)
```

In the LTERM statement, *nn* is any valid DC printer class. This class should be reserved for JES2 internal readers only.

2. Include a DD card in the DC run JCL that links *dd-name* to a JES2 internal reader, using this format:

```
//ddname DD SYSOUT=(A,INTRDR),DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
```

What the program does: The DC application program can write JCL to the JES2 internal reader by using this command:

```
WRITE PRINTER FROM JCL-STATEMENT-AREA LENGTH 80 CLASS nn.
```

After the last JCL statement is written, you use the same command to write one additional line consisting of: /*EOF with 75 trailing blanks.

9.12 Modifying a task's priority

DC selects a task for processing based on its priority assignment. A task's priority is determined by the sum of the priority values assigned for the task code, the user, and the terminal. Tasks with the same priority are handled on a first-in/first-out (FIFO) basis.

To change the dispatching priority of a task:

1. Invoke the specified task.
2. Issue a `CHANGE PRIORITY` statement that specifies a new dispatching priority for the issuing task. The new priority applies only to the current execution of the task.

Note: You cannot use this statement to alter the priorities of other tasks executing under the same DC system.

9.13 Initiating nonterminal tasks

Not all tasks in a DC system are associated with a logical terminal; a task not associated with a logical terminal is called a **nonterminal** task. For example, you can initiate processing of another task while your task is still running; this is called **attaching** a task. The new task competes for processor time and runs concurrently with all other tasks, but is not associated with any terminal. You can indicate either that the nonterminal task should begin processing immediately or after a specified length of time.

Because nonterminal tasks are not associated with an LTE, they cannot perform processing related to a logical terminal. For example, nonterminal tasks cannot perform terminal I/O, receive messages, or monitor resource usage.

9.13.1 Attaching a task

Attached tasks typically perform support functions for the initiating task. For example, an attached task might perform print functions that can be requested by the user.

Steps to attach a task: To initiate a nonterminal task to be performed immediately:

1. Issue an ATTACH statement that specifies the task code of the task to be initiated.
2. If the NOWAIT parameter is specified, check for a status of 3711 (DC-MAX-TASKS), which indicates that the task was not initiated because a maximum task condition exists.
3. Perform alternative processing if 3711 is returned.
4. Perform the IDMS-STATUS routine if 3711 is not returned.

Example of attaching a task: The program excerpt below initiates the nonterminal task DEPTPRNT if the user presses PA02.

```
PROGRAM-ID.                GETMENU.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DEPTPRNT                PIC X(8)    VALUE 'DEPTPRNT'.
PROCEDURE DIVISION.
    BIND MAP SOLICIT.
    BIND MAP SOLICIT RECORD SOLICIT-REC.
    MAP IN USING SOLICIT.
    INQUIRE MAP SOLICIT MOVE AID TO DC-AID-IND-V.
*** ATTACH TASK IF OPERATOR PRESSES PA02 ***
    IF PA02-HIT
        ATTACH TASK CODE DEPTPRNT
            PRIORITY 100 NOWAIT
            ON DC-MAX-TASKS GO TO MAP-OUT-ERR-MT.
    .
*** INPUT PROCESSING ***
```

9.13.2 Time-delayed tasks

You may want to initiate a nonterminal task, but your processing needs require that it not be concurrent with the issuing task. For example, the time-delayed task may compete for resources with the issuing task. DC allows you to initiate a task at the end of a specified period of time.

Steps to initiate a time-delayed task: To initiate a time-delayed task, perform the following steps:

1. Initiate all appropriate fields.
2. Issue a SET TIMER statement that specifies the START parameter, the time interval (in seconds), the time-delayed task's task code, the timer ID, and the variable-storage location of any data to be passed to the time-delayed task.

9.13.3 External requests

DC starts an external request task in response to a request issued by a batch program running outside of the DC region of the operating system. The batch program's operating mode (PROTOCOL) must specify DC-BATCH.

►► For more information on DC-BATCH, see Appendix C, “Batch Access to DC Queues and Printers” on page C-1.

9.13.4 Queue threshold tasks

A sysgen-defined queue can cause a nonterminal task to be started automatically if a predefined threshold is reached. When the queue threshold is reached, DC initiates the nonterminal task. Such a nonterminal task reports on the queue records and then deletes them.

For example, a queue may have a threshold of 100. When the queue exceeds 100 records, DC initiates a task that prints a report and deletes the queue records.

Queue threshold tasks must completely drain the queue and delete all the queue records.

9.14 Controlling abend processing

A program can abnormally terminate in the following ways:

- **DC** terminates a program upon encountering a processing error (for example, a program check).
- The **program** terminates itself upon discovering a situation that would result in invalid results.

DC allows you to specify abend exits, which are invoked upon either a system or a user abend request. These exits specify a program to be invoked in the event of an abend; you can include an abend exit program for each level of a task. Abend exits allow you to determine the cause and severity of the abend. Based on that information, you can return control to the task, return control to the next-higher abend exit, or terminate the program.

9.14.1 Terminating a task

When your program encounters data that indicates errors have occurred, you should terminate processing. Typically, the IDMS-STATUS routine discovers processing errors and abends your program. You should also terminate processing if a situation exists that makes it impossible to ensure valid results (for example, if you are unable to reaccess a previously obtained database record).

To abnormally terminate a task, issue an ABEND statement that specifies a user-defined abend code. Optionally, you can write a formatted dump to the log file and specify whether previously established abend exits should be invoked or ignored.

►► For more information on abend exits, see 9.14.3, “Performing abend routines” on page 9-39 later in this chapter.

9.14.2 Handling db-key deadlocks

You can include logic in your program that is invoked if your run unit is terminated because of a db-key deadlock. This enables your program to maintain the terminal session and save any data that was previously entered on the screen.

At that point, your program can do one of the following:

- Ask the user to resubmit the transaction.
- Automatically restart the run unit, establish currency, and try again.

What happens when a deadlock occurs: When a run unit is terminated because its request would cause a deadlock condition, the DBMS:

1. **Rolls back the recovery unit and terminates the run unit.** The rollback operation releases all locks held by the aborted run unit.
2. **Writes the following message to the log:**

```
TASK: task-code PROG: program-name  
SUBS: subschema-name SSCSTAT: subschema-status  
RUN-UNIT run-unit-id ROLLED OUT.'
```

3. **Returns control to the issuing task** with a status code of *nn29*, which indicates that a deadlock has occurred.

What to do: You can continue a terminal session in the event of a deadlock by having your program resubmit a transaction in response to a minor status code of *nn29*. How you do this is largely a site-specific decision. Typically, you resubmit a transaction in one of two ways:

- Inform the user of the deadlock and request the user to resubmit the transaction
- Programmatically resubmit the transaction

Automatically restarting the run unit: If your program automatically restarts the run unit and retries the transaction, it must:

1. Rebind the run unit by:
 - a. Reinitializing the ERROR-STATUS field in the IDMS communications block to the value 1400
 - b. Issuing the appropriate BIND/READY sequence
2. Reestablish the appropriate currencies before retrying the transaction that originally caused the deadlock.

If you don't check for the minor code: If your program fails to check for a minor code of *nn29*, you can expect the following results:

- **If AUTOSTATUS is in effect**, your program takes the action specified in the site-specific IDMS-STATUS routine.
- **If AUTOSTATUS is not in effect**, your program responds as specified in the program code that checks status codes.

If your program does not contain any generic error-checking logic (such as the IDMS-STATUS routine) and, after receiving a minor code of *nn29*, continues to issue database requests without reestablishing a run unit, the DBMS returns a database status of *nn77* (run unit not bound).

COBOL: COBOL programs must redefine the ERROR-STATUS field of the IDMS communications block to access the minor code value.

Example of resubmitting the transaction: The program excerpt below informs the user of a database minor code of *nn29* and requests that the transaction be resubmitted:

```

WORKING-STORAGE SECTION.
01 SUBSCHEMA-CTRL.
   03 PROGRAM-NAME          PIC X(8) VALUE SPACES.
   03 ERROR-STATUS          PIC X(4) VALUE '1400'.
   .
   .
   03 SUBSCHEMA-CTRL-END    PIC X(4).
01 SSC-REDEF REDEFINES SUBSCHEMA-CTRL.
   03 FILLER                 PIC X(8) VALUE SPACES.
   03 ERRSTAT-REDEF.
     05 ERRSTAT-MAJ          PIC XX.
     05 ERRSTAT-MIN          PIC XX.
     88 DEADLOCK             VALUE '29'.
   03 FILLER                 PIC X(292).
*
01 MESSAGES.
   05 DBKEY-DEADLOCK-MESSAGE PIC X(80) VALUE
     'REQUESTED RECORD IN USE. PLEASE RESUBMIT TRANSACTION'.
   .
   .
PROCEDURE DIVISION.
   .
   .
IDMS-ABORT.
   IF DEADLOCK
   THEN
       MODIFY MAP TSKMAP01 TEMPORARY
           FOR ALL FIELDS NOMDT
       MAP OUT USING TSKMAP01
       MESSAGE IS DBKEY-DEADLOCK-MESSAGE LENGTH 80
       DC RETURN NEXT TASK CODE 'UPDATASK'.
IDMS-ABORT-EXIT.
   EXIT.
   COPY IDMS IDMS-STATUS.
*****
IDMS-STATUS SECTION.
***** IDMS-STATUS FOR IDMS-DC *****
   IF DB-STATUS-OK GO TO ISABEX.
   PERFORM IDMS-ABORT.
   MOVE ERROR-STATUS TO SSC-ERRSTAT-SAVE
   MOVE DML-SEQUENCE TO SSC-DMLSEQ-SAVE
   SNAP FROM SUBSCHEMA-CTRL TO SUBSCHEMA-CTRL-END
       ON ANY-STATUS NEXT SENTENCE.
   ABEND CODE SSC-ERRSTAT-SAVE
       ON ANY-STATUS NEXT SENTENCE.
ISABEX. EXIT.
DMCL-DC-GEN-GOBACK SECTION.
GOBACK.

```

9.14.3 Performing abend routines

You can establish linkage to an abend routine to which DC passes control if the issuing task terminates. Optionally, you can cancel linkage to a previously established abend routine. Each level in a task can have one abend exit in effect at any given time; if more than one abend exit has been established for a level, DC recognizes the last abend exit requested.

Executing abend exits: When a task terminates abnormally (following either a processing error or an ABEND request), abend exits for the program that was executing at the time of the abend and for all higher-level programs will be executed before the task is terminated. You can prevent DC from executing abend exits automatically either by coding the EXITS IGNORED clause in an ABEND request (explained above) or by specifying the abort or continue options in the abend routine's DC RETURN statement. DC RETURN requests are typically handled as follows:

- **Normal** termination passes control either to an abend exit at a higher level or to DC:

DC RETURN.

- **Abort** termination passes control directly to DC, bypassing any other exit programs:

DC RETURN ABORT.

SET ABEND EXIT statements: To establish linkage to an abend exit, which will be invoked if the issuing task terminates, issue a SET ABEND EXIT statement that specifies the program to be called in the event of an abend.

To cancel any previously requested abend exits for the issuing task level, issue a SET ABEND EXIT OFF command.

9.15 Establishing and posting events

At certain times, you may need to suspend execution of your task (that is, enter a wait state) until some specific *event* is completed. The most frequent event is I/O. Typically, the wait is automatically handled by DC, which puts the task in a wait state and, upon completion of the I/O, places the task in a ready state.

You can define an event simply by naming the event in a wait request. DC, upon receiving the wait request, places your task in a wait state. The task is returned to a ready state when another task (the task performing the event), upon completion, posts the event by name. One typical use of user-defined events is to synchronize the concurrent execution of different tasks; for example, a terminal task and a concurrent nonterminal task.

Steps to establish and post an event: To place a task in a wait state, waiting for the completion of an event, perform the following steps:

1. Establish a binary fullword field (PIC S9(8) COMP) that identifies the event control block (ECB) to be posted.
2. Begin execution of the task that will post the event by issuing either an ATTACH or a SET TIMER statement.
3. Place the issuing task in a wait state by issuing a WAIT statement that names the event to be posted.
4. Post the event, redispaching the waiting task, by issuing either a POST or a SET TIMER POST statement in the secondary program.

Chapter 10. Advanced CA-IDMS Programming Topics

- 10.1 About this chapter 10-3
- 10.2 Calling a DC program from a CA-ADS dialog 10-4
- 10.3 Basic mode 10-6
 - 10.3.1 Reading data from the terminal 10-7
 - 10.3.2 Writing data to the terminal 10-7
- 10.4 Determining if asynchronous I/O is complete 10-8
- 10.5 Communicating with database procedures 10-9
 - 10.5.1 BIND PROCEDURE 10-9
 - 10.5.2 ACCEPT PROCEDURE CONTROL LOCATION 10-10
- 10.6 Managing queued resources 10-12

10.1 About this chapter

This chapter presents information about how to:

- Call a DC program from a CA-ADS dialog
- Transfer data between the terminal and program in basic mode
- Determine whether a request for asynchronous I/O has completed
- Add fields and statements to a program for communication with database procedures
- How to acquire and release a queued resource

10.2 Calling a DC program from a CA-ADS dialog

CA-ADS dialogs can call COBOL, PL/I, or Assembler programs by using the LINK function. For example, a commonly used date conversion routine could be coded in COBOL for use by all CA-ADS dialogs running under a DC system.

Because CA-ADS calls your program using the LINK command, linkage conventions are the same as if the call were from another DC program.

The calling dialog can pass the following records to the linked program:

- Subschema control block
- Map request block
- Any records to be used in the linked program

Within the linked program, you can issue DC RETURN statements with the NEXT TASK CODE parameter to perform pseudoconversational processing as required.

Extended run unit: The linked program may not need to issue any BIND statements or reestablish currencies if the CA-ADS dialog establishes an extended run unit.

►► For more information on CA-ADS and extended run units, refer to *CA-ADS Reference Guide*.

Steps to call a program from CA-ADS: To code a program to be called by CA-ADS dialogs, perform the following steps:

1. Define any passed records in the LINKAGE SECTION and code a PROCEDURE DIVISION USING statement.
2. If an extended run unit has been established, do not issue a BIND RUN-UNIT statement. You can issue BIND RECORD statements for any records which have not already been bound for the run unit, and you can issue other appropriate BIND statements.
3. Perform processing, as required.

If an extended run unit has been established, do not issue FINISH or ROLLBACK statements within the called program. To issue either of these statements, return to the calling dialog with an indicator in a passed status field and let the dialog end the run unit. If you do not follow this procedure, the CA-ADS program may receive an error (DC174019) when it tries to save currencies for a run unit that no longer exists.

4. Return control to the CA-ADS dialog by issuing one of the following DC RETURN statements:
 - If the program or one of its subroutines has issued a DC RETURN statement, issue a DC RETURN statement that specifies a next task code of 'ADSR'

- If the program issues no DC RETURN statements, issue a DC RETURN statement that specifies no next task code

Example of a subroutine called by CA-ADS: The program excerpt below is a subroutine called by a CA-ADS dialog to perform data conversion functions.

Depending on the conversion code, it converts a Julian date to Gregorian or a Gregorian date to Julian.

```

WORKING-STORAGE SECTION.
01 CONVERT-CODES.
   05 JULGREG          PIC X    VALUE 'J'.
   05 GREGJUL         PIC X    VALUE 'G'.
01 GREGORIAN.
   10 MM              PIC 99   VALUE ZEROS.
   10 DD              PIC 99   VALUE ZEROS.
   10 YY              PIC 99   VALUE ZEROS.
01 JULIAN.
   10 JULIAN-YY       PIC 99   VALUE ZEROS.
   10 JULIAN-DDD      PIC 999  VALUE ZEROS.
LINKAGE SECTION.
*** DEFINE RECORDS THAT ARE PASSED FROM CA-ADS ***
01 COPY IDMS SUBSCHEMA-CTRL.
01 COPY IDMS RECORD DATE-RECORD.
01 COPY IDMS RECORD DIALOG-REFERENCE-RECORD.
PROCEDURE DIVISION USING SUBSCHEMA-CTRL
DATE-RECORD
DIALOG-REFERENCE-RECORD.
   IF CONV-DIRECTION = JULGREG
     PERFORM A100-JULGREG
   ELSE
     IF CONV-DIRECTION = GREGJUL
       PERFORM A100-GREGJUL
     ELSE
       PERFORM A100-ERROR.
*** RETURN CONTROL TO CA-ADS PROGRAM ***
DC RETURN.
*** DATE CONVERSION AND ERROR PROCESSING ***
.
.
.

```

10.3 Basic mode

In basic mode, DC performs device-dependent data transfers between your program and the terminal. Your program must format the data and supply device-control characters based on the type of terminal in use; DC inserts the necessary line control information. For example, with 3270-type devices, you must send and receive data with device-control information that includes write control characters, orders, and buffer addresses.

The figure below shows a basic mode data transfer. DC appends framing characters to the input data stream and performs the required I/O.

Data stream as built by the user

Data and device-control information

Data stream as passed by basic mode request

LINE CONTROL	Data and device-control information	LINE CONTROL
--------------	-------------------------------------	--------------

►► For information on using basic mode to support System Network Architecture (SNA) protocols, refer to *CA-IDMS DML Reference - Assembler*.

I/O requests under basic mode: Basic mode supports synchronous and asynchronous read and write requests. The terms synchronous and asynchronous do not refer to line protocol for data transmission but rather to task processing during I/O operations. Synchronous and asynchronous I/O requests function in the following manner:

- Following a **synchronous** I/O request, control returns to DC, which places the issuing task in an inactive state. When the requested I/O operation is complete, DC places the task in a ready state and the task resumes processing according to its established dispatching priority.
- Following an **asynchronous** I/O request, the issuing task continues executing.

DC assumes that all I/O requests are synchronous unless a program explicitly requests asynchronous processing.

What you can do in basic mode: Using basic mode terminal management, you can perform the following functions:

- **Read data** — You can transfer data from the terminal to program variable storage.
- **Write data** — You can transfer data from program variable storage to the terminal.
- **Determine if I/O is complete** — You can check to determine if a previously issued asynchronous request is complete.

10.3.1 Reading data from the terminal

To transfer data from the terminal screen to program variable storage, issue either a `READ TERMINAL` or a `WRITE THEN READ TERMINAL` statement. This transfer begins when the user signals completion of the data entry by pressing an AID key. With 3270-type devices, data can optionally be transferred to the program without user intervention.

Note: `WRITE THEN READ TERMINAL` is not recommended because it is inherently conversational and holds resources.

Acquiring the input buffer: You must dynamically acquire the input buffer for record-element descriptions from the storage pool when the read operation is complete:

- If you specify `WAIT`, your program must acquire the input buffer by including a `GET STORAGE` parameter in the `READ TERMINAL` or `WRITE THEN READ TERMINAL` request. Your program is also responsible for releasing the acquired storage explicitly with a `FREE STORAGE` statement. If storage is not explicitly freed, DC releases all acquired buffers when the task terminates.
- If you specify `NOWAIT`, your program must acquire the input buffer by including a `GET STORAGE` parameter in the `CHECK TERMINAL` request.

Where to define data: Because storage is acquired by an explicit program request, you must define the associated data-item descriptions in the program's `LINKAGE SECTION`.

10.3.2 Writing data to the terminal

To transfer data from program variable storage to the terminal screen, issue a `WRITE TERMINAL` statement.

If the output buffer has been dynamically acquired, you can optionally release that area by including a `FREE STORAGE` parameter in the `WRITE TERMINAL` request. The associated storage is released when the write operation is complete.

Output buffers that are explicitly acquired and released must be defined in the program's `LINKAGE SECTION`.

10.4 Determining if asynchronous I/O is complete

When your program issues an asynchronous I/O request, DC establishes an ECB that is posted only after the requested I/O is complete. Before performing further I/O operations, you must issue a `CHECK TERMINAL` statement to determine if the ECB has been posted. If the ECB is unposted, indicating that the I/O is not complete, DC places the task in an inactive state. When the operation is complete, DC reactivates the task according to its established dispatching priority.

The `CHECK TERMINAL` statement must be used following all asynchronous I/O requests, regardless of mode. That is, mapping mode and line mode output requests that specify `NOWAIT` must issue a `CHECK TERMINAL` statement before issuing any subsequent I/O requests.

10.5 Communicating with database procedures

Database procedures, which can be invoked before or after various DML functions, are defined in the schema by the DBA. For example, a data compression routine might be invoked before STORE and MODIFY; a decompression routine might be invoked after FIND.

Use of database procedures: Database procedures typically have more authority than application programs. For example, they can access all record elements of a schema-defined record and not just the fields defined in the subschema view. Therefore, if your program must provide more information than is provided by the DBMS itself, you can establish communications with a database procedure. Such instances are unusual; in most cases, you are not aware of the procedures called before or after various DML commands.

Steps to establish communication: To establish communications with a database procedure, add the following fields to program variable storage:

- **An 8-byte character literal** aligned on a fullword boundary. This field contains the name of the procedure to be called.
- **A 256-byte area** to which the procedure will be bound. This field defines the information to be passed.

Statements to communicate with database procedures: The following statements enable your program to communicate with database procedures:

- BIND PROCEDURE establishes communication and passes data to the procedure
- ACCEPT PROCEDURE CONTROL LOCATION returns data from the procedure to program variable storage

These statements are explained below.

10.5.1 BIND PROCEDURE

The BIND PROCEDURE statement establishes communication between your program and a DBA-written database procedure. Additionally, the specified data in variable-storage is copied to the application program information block in the central version.

When to use it: Consult with your DBA to determine when to issue the BIND PROCEDURE. After issuing a BIND PROCEDURE statement, you can modify fields in the 256-byte block without affecting communications with the procedure (for example, by using the ACCEPT PROCEDURE CONTROL LOCATION statement). The data passed is the information contained in the block at the time of the BIND PROCEDURE statement.

Example of the definition in variable storage: The program excerpt below shows a sample 256-byte DBA-defined application program information block as listed in program variable storage.

```
DATA DIVISION
WORKING-STORAGE SECTION.
01 CHECKID                PIC X(8)      VALUE 'CHECKID'.
01 CHECKID-CTRL.
   05 CHECKID-DATE        PIC X(8).
   05 CHECKID-USER        PIC X(32).
   05 CHECKID-INFO        PIC X(216).
```

10.5.2 ACCEPT PROCEDURE CONTROL LOCATION

You can use the ACCEPT PROCEDURE CONTROL LOCATION statement to return a copy of the data bound to a database procedure to a specified location in program variable storage. A BIND PROCEDURE statement previously placed information into this block; this information may have been subsequently updated by the procedure.

ACCEPT PROCEDURE CONTROL LOCATION should be used by programs running under, but in a different region/partition from, the central version.

Example of communicating with a database procedure: The program excerpt below shows the use of the BIND PROCEDURE and the ACCEPT PROCEDURE CONTROL LOCATION statements.

The BIND PROCEDURE statement is issued only once; the ACCEPT PROCEDURE CONTROL LOCATION statement is issued after STORE processing to return information from the user-written procedure. The database procedure itself is transparent to your application.

```

DATA DIVISION
WORKING-STORAGE SECTION.
01 CHECKID                PIC X(8)      VALUE 'CHECKID'.
01 CHECKID-CTRL.
   05 CHECKID-DATE        PIC X(8).
   05 CHECKID-USER        PIC X(32).
   05 CHECKID-INFO        PIC X(216).
PROCEDURE DIVISION.
.
.
  READ NEW-EMP-FILE-IN.
  AT END MOVE 'Y' TO EOF-SW.
*** ESTABLISH COMMUNICATION AND TRANSFER INFO TO ***
*** THE APPLICATION PROGRAM INFORMATION BLOCK ***
  BIND PROCEDURE FOR CHECKID TO CHECKID-CTRL.
  PERFORM A300-STORE-EMP THRU 0300-EXIT
  UNTIL END-OF-FILE.
*** MOVE DATA FROM THE PROCEDURE TO ***
*** PROGRAM VARIABLE STORAGE ***
  ACCEPT CHECKID-CTRL FROM CHECKID PROCEDURE.
  PERFORM U100-WRITE-PROC-INFO.
  FINISH.
  GOBACK.
A300-STORE-EMP.
.
*** ESTABLISHING CURRENCY AND INITIALIZATION FOR STORE ***
.
  STORE EMPLOYEE.
  PERFORM IDMS-STATUS.
  PERFORM U500-WRITE-NEW-EMP-REPORT.
A300-GET-NEXT.
  READ NEW-EMP-FILE-IN
  AT END MOVE 'Y' TO EOF-SW.
A300-EXIT.
  EXIT.
U100-WRITE-PROC-INFO.
  DISPLAY '**** STORE PROCEDURE INFORMATION ****'
  'DATE ' CHECKID-DATE
  'USER' CHECKID-USER
  'INFO FOLLOWS: ' CHECKID-INFO.

```

10.6 Managing queued resources

Resources are objects that your program must explicitly ask for before it can do any work. Multiple resources may be required to perform a logical unit of work. For example, a database area is a resource that you ask for by issuing a **READY** statement; a database record occurrence is a resource that you ask for by issuing a **FIND/OBTAIN** statement.

Holding resources: The number of resources that you hold and the way that you hold them affects other run units. For example, resources can be shared or exclusive.

You should adhere to the following guidelines when holding resources:

- **Free resources as soon as you are finished** in order that other run units can access them.
- **Hold resources for as short a time as possible.**
- **Acquire the lowest-level lock that you need.** For example, use shared locks instead of exclusive locks whenever possible.

Examples of resources: Typical resources include:

- Database areas
- Database records
- Storage areas
- Common routines
- Queues
- Site-specific functions (for example, database update)

Meaning of queued resource: Your site may utilize queued resources. A queued resource is any resource that requires serial access. That is, only one program can access it at a time.

DC allows you to perform the following resource management functions:

- You can **test** to see if a resource is currently available.
- You can **acquire** a resource for exclusive use.
- You can **release** a previously acquired resource.

These functions are explained below, followed by a list of suggestions that you can use to avoid deadlocks.

Testing for resource availability: To determine if a resource or list of resources is currently available, perform the following steps:

1. Issue an **ENQUEUE** request that includes the **TEST** parameter.

2. Check for the following statuses:

- **0000** indicates that all the tested resources were available and have now been enqueued for your task.
- **3908** indicates that at least one of the tested resources is already owned by another task.
- **3909** indicates that at least one of the tested resources is not yet owned by another task and is available to your task.

Acquiring resources: To acquire and lock a resource or list of resources, perform the following steps:

1. Issue an ENQUEUE request that includes either the WAIT or the NOWAIT parameter.
2. Check for the following statuses:
 - **0000** indicates that all requested resources have been acquired and locked.
 - **3901** indicates that at least one of the tested resources cannot be enqueued immediately; to wait would cause a deadlock. No new resources have been acquired.
 - **3908** indicates that at least one of the tested resources is currently owned by another task. No new resources have been acquired.

Releasing resources: After all processing is complete, release resources by issuing a DEQUEUE statement. You can release resources by name or all at once (by including the ALL parameter).

Avoiding deadlock: One of the conditions of deadlock is that a program is holding resources while waiting for other resources. The following list explains techniques that your site can use to minimize this condition:

- **Request all required resources at the same time.** Whenever possible, you should try to ensure that your program isn't holding resources while waiting for other resources.
- **If you are denied access to a resource, you should release all previously acquired resources and start over.** After you release previously acquired resources, you can acquire all resources at the same time, as specified above.
- **Your site can follow a protocol of sequential order.** All programs follow a protocol that prescribes the order in which database records will be retrieved and updated.

Note: This protocol will work only if every program in the system follows it.

For example, all update applications that use an area sweep can agree to enter the database starting with the DEPARTMENT record rather than the OFFICE record.

This protocol can also specify the order in which locks will be acquired and released.

Sharing queued resources between CA-IDMS systems: In a data sharing environment, queued resources can be shared between CA-IDMS systems that are members of a data sharing group. The benefit of sharing these resources is that access to them can be controlled between programs executing on any member of the group. Whether or not a specific queued resource is shared, is determined by specifications made by the CA-IDMS system administrator. Programs accessing queued resources are not sensitive to whether or not a resource is shared, since the DML syntax is the same in either case.

Chapter 11. Testing

- 11.1 About this chapter 11-3
- 11.2 Preparing programs for execution 11-4
- 11.3 Selecting local mode or central version 11-5
- 11.4 Using SYSIDMS parameters and DCUF SET statements 11-6
- 11.5 Overriding subschemas (Release 10.2) 11-7
 - 11.5.1 Overriding a batch program's subschema 11-7
 - 11.5.2 Overriding an online program's subschema 11-9
- 11.6 Setting up an online test application 11-10

11.1 About this chapter

This chapter discusses the following topics related to the testing phase of program development:

- Preparing programs for execution — A discussion on precompiling, compiling, and link editing your program
- Selecting local mode or central version — A discussion on using local mode and central version in the test environment
- Overriding subschemas (Release 10.2) — A discussion on overriding the subschema at runtime in both the batch and the online environments
- Setting up an online test application — A discussion on creating an online test environment

11.2 Preparing programs for execution

To prepare a CA-IDMS load module, perform the following steps:

1. Execute the appropriate **precompiler** to obtain a source-language program. The precompiler is the preprocessor that creates expanded source code and copies any specified dictionary record descriptions or modules.
2. Execute the host-language **compiler** or **assembler** to obtain an object program.
3. Execute the **linkage editor** to obtain a load module (or phase). You should store CA-IDMS load modules in a load (core-image) library defined to the test system.

►► For more information on this phase of testing, refer to the language-specific CA-IDMS DML reference manual.

11.3 Selecting local mode or central version

Follow the guidelines listed below to determine whether to use local mode or central version in the test environment:

- Use **local mode** for testing batch programs. Be sure to back up the database before running any update applications.
- Use the **central version** mainly for online programs. Run batch jobs under the central version only to test aspects of central version processing (for example, update locks).

11.4 Using SYSIDMS parameters and DCUF SET statements

Using SYSIDMS parameters you can change the specification of these components of the physical environment in which your program executes without changing the program source:

- Database to be accessed
- Dictionary whose load area contains the subschema
- System to which the program should bind

To take advantage of this feature, the BIND RUN_UNIT statements in the program should not specify the DBNAME, DICTNAME, and NODENAME parameters. Any such hard-coded specification cannot be overridden at execution time.

Batch execution: For a program executing in batch mode, you can make DBNAME, DICTNAME, and NODENAME specifications in the JCL using SYSIDMS parameters.

►► For documentation of SYSIDMS parameters, see *CA-IDMS Database Administration*.

For sample JCL, refer to the language-specific DML reference manual.

Online execution: For an online program, you can issue a DCUF SET statement to specify database, dictionary, and system. A DCUF SET statement can be submitted to the system by the user or by the program itself.

►► For more information about DCUF SET statements, refer to *CA-IDMS System Tasks and Operator Commands*.

User session attributes: When a program executes under the central version, the executing user is signed on the system automatically if the user is authorized and an explicit signon has not occurred. Signon processing establishes a set of attributes for the user session, including, for example, DBNAME and DICTNAME and the values assigned to them.

The program can access attribute information with a call to the IDMSIN01 entry point to the IDMS module. The program can use this feature to determine whether the user has the appropriate values assigned to the different components of the execution environment.

►► For more information about using calls to IDMSIN01, see Appendix F, “Calls to IDMSIN01” on page F-1.

11.5 Overriding subschemas (Release 10.2)

You can override the program-specified subschema to access a test database that exists in an multiple-database environment. This allows you to perform testing without disrupting the production environment.

The procedures for overriding subschemas differ in the batch and the online environment.

Database name table: A database to be accessed by an application program with navigational DML must have one or more subschemas defined for it. The central version maintains this association in the database name table created prior to Release 12.0 where an entry exists for each database that can be accessed under that central version. A database name table entry includes the following information:

- The name of the database
- The names of the subschemas that map to the database
- For each subschema that maps to the database, the name of an equivalent subschema that provides the same database perspective (that is, the same record definitions) but maps to different page ranges (that is, different data)

11.5.1 Overriding a batch program's subschema

To override the subschema named in a batch application program in the OS/390 environment, perform the following steps:

1. Include an 01-level LINKAGE SECTION entry that names two subordinate data items:

```
01  RUNTIME-TEST-PARMS.
    05  PARM-LENGTH          PIC S9(4) COMP.
    05  RUNTIME-TEST-SUBSCHEMA PIC X(8).
```

2. Include a USING statement in the PROCEDURE DIVISION heading:


```
PROCEDURE DIVISION USING RUNTIME-TEST-PARMS.
```

3. Before issuing any BIND statements, perform processing to determine if a subschema override has been included in the execution JCL:

```
IF PARM-LENGTH NOT EQ ZERO
  MOVE RUNTIME-TEST-SUBSCHEMA TO SUBSCHEMA-SSNAME.
```

4. Issue database BINDS and perform other processing as needed
5. At runtime, include a PARM option in the JCL EXEC statement that specifies the name of the alternative subschema

Local mode considerations: The database and the appropriate subschemas must be defined in the database name table in the load (core-image) library.

Note: For more information about the database name table, see *CA-IDMS Database Administration*.

VSE/ESA users: You can pass the alternative subschema name by using a SYSPARM:

```
// OPTION SYSPARM=ssname.
```

BS2000/OSD considerations: You can pass the alternative subschema name by using a job variable. Perform the following steps:

1. Include a mnemonic name in the SPECIAL-NAMES paragraph of the CONFIGURATION SECTION:

```
CONFIGURATION SECTION.
SPECIAL-NAMES.
  JV-LNKNAME IS MNEMOJV.
```

LNKNAME is the linkname specified in the JCL; MNEMOJV is the name used by the program to reference the job variable.

2. Include a data-item that will contain the job variable value:

```
WORKING-STORAGE SECTION.
01 JVVALUE PIC X(8).
```

3. Before issuing any BIND statements, access the job variable value and determine if a subschema override has been requested:

```
ACCEPT JVVALUE FROM MNEMOJV.
IF JVVALUE NOT EQ '/'*
  THEN MOVE JVVALUE TO SUBSCHEMA-SSNAME.
```

4. Issue database BINDS and perform other processing as needed.

5. At runtime, include the following JCL commands:

```
/SET-JV-LINK L-NAME=*LNKNAME,JV-NAME=user-JV-NAME
/MOD-JV JV-CONTENTS=*LINK(LINK-NAME=*LNKNAME),SET-VALUE=C'subschema-name'
```

Batch Assembler programmers: Batch Assembler programs can use the GETJV macro to move the job variable into program variable storage.

Example of overriding a batch subschema: The program excerpt and JCL below illustrate the batch subschema override technique in the OS/390 COBOL environment.

The PARM option on the EXEC statement specifies the name of a test subschema used to override the production subschema. If the parameter is passed, the application program moves the parameter to the SUBSCHEMA-SSNAME field in program variable storage before issuing the BIND RUN-UNIT command.

Source code:

```

SCHEMA SECTION.
DB EMPSS01 WITHIN EMPSCHM.
  |
01 SUBSCHEMA-SSNAME PIC X(8) VALUE 'EMPSS01'.
  |
LINKAGE SECTION.
01 RUNTIME-TEST-PARMS.
  05 PARM-LENGTH PIC S9(4) COMP.
  05 RUNTIME-TEST-SUBSCHEMA PIC X(8).
PROCEDURE DIVISION USING RUNTIME-TEST-PARMS.
  MOVE 'TESTPROG' TO PROGRAM-NAME.
  IF PARM-LENGTH NOT EQUAL TO 0 THEN
    MOVE RUNTIME-TEST-SUBSCHEMA TO SUBSCHEMA-SSNAME.
  BIND RUN-UNIT.
.
.
.

```

Runtime JCL:

```
//RUNJOB EXEC PGM=TESTPROG,PARM='EMPSS01T'
```

11.5.2 Overriding an online program's subschema

A program executing under the central version can be directed to access a specific database by using a DCUF SET DBNAME command. DCUF SET DBNAME establishes a default database for the current logical terminal and overrides the subschema named in the program's BIND RUN-UNIT statement. For example, to establish EMPTSTDB as the default database, either the user or the program can issue this CA-IDMS command:

```
DCUF SET DBNAME EMPTSTDB
```

►► For more information on specifying a default database, see *CA-IDMS System Operations*.

OS/390 and BS2000/OSD systems: The specified database can be overridden by a specification in an IDMSOPTI module or SYSCTL file.

VSE/ESA systems: The specified database can be overridden by a specification in an IDMSOPTI module.

11.6 Setting up an online test application

There are two typical online test configurations, although your site standards for online testing may be different. The two configurations are as follows:

- Online test programs are link edited into a **test load (core-image) library** that is defined to the DC system. This library is designated for test programs belonging to a specified user or group of users.
- Online test programs are link edited into a load (core-image) library that is defined to a DC system that contains a system dictionary and at least one **application dictionary**. This application dictionary should have been defined for testing in the DC system.

►► For information on using extended architecture (OS/390) to test programs above the 31-bit line, see Appendix D, “XA Considerations” on page D-1.

Dynamically defining programs and tasks: Before your application can execute under CA-IDMS, you must ensure that all of its programs and tasks are defined to CA-IDMS. You can define programs and tasks either at system generation or dynamically by issuing DCMT VARY DYNAMIC PROGRAM and DCMT VARY DYNAMIC TASK commands. For example, to dynamically define the DCADDEMP program and its associated task, issue the following DCMT statements:

```
DCMT VARY DYNAMIC PROGRAM DCADDEMP QUASIREENTRANT .
```

```
DCMT VARY DYNAMIC TASK ADDEMP INVOKES DCADDEMP INPUT .
```

Clist: Because an application can consist of many programs and task codes, it is a good idea to define an application's dynamic program and task definition statements as a module in the dictionary. This module can then be invoked as a command list (clist) from the online DC system.

For more information on command lists, refer to *CA-IDMS System Operations*.

NEW COPY: You may need to redefine a recompiled program or map if NEW COPY is defined as MANUAL at system generation. To mark a previously defined program to new copy, issue the following online CA-IDMS command:

```
DCMT VARY PROGRAM DCADDEMP NEW COPY
```

Using a test load library (OS/390 and BS2000/OSD only): To execute your test application in a DC system that uses a test load library for such applications, perform the following steps:

1. When coding is finished, compile the programs and link edit them into the load library that has been assigned the specified version number.
2. Define the programs to the DC system either at sysgen or by issuing DCMT VARY DYNAMIC PROGRAM commands.
3. Define the tasks to the DC system either at sysgen or by issuing DCMT VARY DYNAMIC TASK commands.

4. Establish the runtime test version number by issuing a DCUF TEST command.
5. Perform online application testing, as necessary.

Using an application dictionary: To execute your test application in a DC system that uses an application dictionary, perform the following steps:

1. Link edit all programs into a load (core-image) library that has been defined to the DC system.
2. Establish the application dictionary as the session default dictionary by issuing a DCUF SET DICTNAME command.
3. Define the programs to the DC system either at sysgen or by issuing DCMT VARY DYNAMIC PROGRAM commands.
4. Define the tasks to the DC system either at sysgen or by issuing DCMT VARY DYNAMIC TASK commands.
5. Perform online application testing, as necessary.

Chapter 12. Debugging

12.1 About this chapter	12-3
12.2 Debugging batch programs with the CA-IDMS trace facility	12-4
12.3 Using the CA-OLQ menu facility	12-6
12.4 Reading task dumps	12-7
12.4.1 Contents of a snap dump	12-7
12.4.2 How to use the dump	12-9
12.5 Error checking	12-12

12.1 About this chapter

This chapter discusses the following topics related to the debugging phase of program development:

- Using the CA-IDMS trace facility — To trace program execution
- Using the CA-OLQ menu facility — To confirm database access
- Reading task dumps — To determine the contents of DC control blocks listed in a task dump
- Error checking — To inventory typical programming errors and possible solutions

Note: The online debugger is an additional facility for debugging online CA-IDMS/DC and CA-IDMS/UCF programs written in Assembler, COBOL, or PL/I. That facility is described in *CA-IDMS Online Debugger*.

12.2 Debugging batch programs with the CA-IDMS trace facility

You can use the CA-IDMS trace facility to trace database calls in the following types of programs:

- Batch application programs that run either under the central version or in local mode
- CA-IDMS utilities, compilers, and reports

The trace facility writes one line to the SYSLST file for each call to the IDMS module. This line contains the following:

- DML sequence number, if the DEBUG option was specified at compile time and mode is not DC-BATCH
- Database key
- Error status
- DML verb number
- DML verb name
- Record, set, or area name (if applicable)

What you can do: You can use the CA-IDMS trace facility to:

- Help debug application programs
- Analyze and tune database navigation
- Analyze unfamiliar programs that have been assigned to you for maintenance

Activating the trace facility: To activate the trace facility, specify the SYSIDMS parameter DMLTRACE=ON. For example:

```
//SYSIDMS *  
DBNAME=TSTDICT  
DMLTRACE=ON  
.  
.  
.
```

This activates a trace of all DML calls made by the program.

Trace facility output: The following example shows CA-IDMS trace facility output for a sample COBOL program:

```

Verb=59 BIND SUBSCHEMA-->EMPSS01      DBNAME=EMPDB      PROGRAM=CBDMLO4
Verb=59 BIND SUBSCHEMA-->IDMSNWKL     DBNAME=SYSTEM     PROGRAM=RHDCRUAL
Verb=37 READY Area Retrieval          AREA->DDLDCLOD
Verb=54 ACCEPT Current of Run-Unit
Verb=48 BIND Record                   REC->LOADHDR-156   ADDR=8502AF0C
Verb=32 OBTAIN CALC                   REC->LOADHDR-156
I D M S SSCSTAT=0326 ERRREC=LOADHDR-156 ERRAREA=DDLDCLOD DBKEY=20113:0
Verb=02 FINISH
Verb=59 BIND SUBSCHEMA-->IDMSNWKL     DBNAME=SYSTEM     PROGRAM=RHDCRUAL
Verb=37 READY Area Retrieval          AREA->DDLDCLOD
Verb=54 ACCEPT Current of Run-Unit
Verb=48 BIND Record                   REC->LOADHDR-156   ADDR=8505C520
Verb=32 OBTAIN CALC                   REC->LOADHDR-156
I D M S SSCSTAT=0326 ERRREC=LOADHDR-156 ERRAREA=DDLDCLOD DBKEY=20113:0
Verb=48 BIND Record                   REC->LOADHDR-156   ADDR=8505C520
Verb=32 OBTAIN CALC                   REC->LOADHDR-156
I D M S SSCSTAT=0326 ERRREC=LOADHDR-156 ERRAREA=DDLDCLOD DBKEY=20113:0
Verb=02 FINISH
Verb=59 BIND SUBSCHEMA-->IDMSSECU     DBNAME=SYSUSER    PROGRAM=RHDCRUAL
Verb=37 READY Area Retrieval          AREA->DDLSEC
Verb=54 ACCEPT Current of Run-Unit
Verb=48 BIND Record                   REC->USER          ADDR=8009390C
Verb=32 OBTAIN CALC                   REC->USER
I D M S SSCSTAT=0370 ERRAREA=DDLSEC   DBKEY=8000006:0
Verb=02 FINISH
Verb=59 BIND SUBSCHEMA-->IDMSSECU     DBNAME=SYSUSER    PROGRAM=RHDCRUAL
Verb=37 READY Area Retrieval          AREA->DDLSEC
Verb=54 ACCEPT Current of Run-Unit
Verb=48 BIND Record                   REC->PROFILE       ADDR=85024810
Verb=48 BIND Record                   REC->ATTRIBUTE     ADDR=85024810
Verb=32 OBTAIN CALC                   REC->PROFILE
I D M S SSCSTAT=0370 ERRAREA=DDLSEC   DBKEY=8000006:0
Verb=02 FINISH

```

Turning trace on and off: You can use the DML trace facility selectively by adding logic to the program itself. You can switch the trace facility on and off within the program by issuing a call to the IDMSIN01 entry point of the IDMS module.

►► For information about how to call IDMSIN01 to manage the DML trace facility, see Appendix F, “Calls to IDMSIN01” on page F-1.

12.3 Using the CA-OLQ menu facility

During the debugging phase of program development, you may need to determine if your application accessed the proper record occurrences or that database modifications were actually applied. You can use the CA-OLQ menu facility to help you accomplish these tasks.

The CA-OLQ menu facility can perform the following functions:

- Check the sequence of records retrieved by your program
- Test database navigation logic
- Confirm database access and modification

Retrieving database records: To retrieve database records, perform the following steps after signing on to the CA-OLQ menu facility:

1. On the MENU screen, choose the RECORD option.
2. On the SIGNON screen, indicate the appropriate subschema.
3. On the RECORD SELECT screen, indicate which database records or logical record you want to retrieve.
4. On the FIELD SELECT screen, indicate which fields in the previously specified database records or logical record you want to display; optionally, specify selection criteria.
5. Press [Enter]. CA-OLQ automatically generates retrieval paths and performs the database access.
6. Press [Enter] again. CA-OLQ displays the retrieved data.

Note: The sequence listed above is the default sequence for the RECORD option. You need only press [Enter] after each step to continue to the next screen.

Storing an CA-OLQ qfile: If you will be using the CA-OLQ menu facility to perform the same database access repeatedly, you might want to store the logic in a qfile. To create a qfile, perform the following steps:

1. Perform steps 1 through 6 listed above.
2. On the MENU screen, select the EXPRESS ROUTINE option.
3. On the EXPRESS ROUTINE screen, specify the create option and a name.

Executing an CA-OLQ qfile: To execute a qfile, perform the following steps:

1. On the MENU screen, select the EXPRESS ROUTINE option.
2. On the EXPRESS ROUTINE screen, specify the execute option and a qfile name.

►► For more information, refer to *CA-OLQ User Guide*.

12.4 Reading task dumps

You can use a task dump to obtain task-related information that may not be available under the online debugger. For example, you can find out about the control blocks related to task or program definition by reading a task dump.

You should be familiar with dump reading and hexadecimal notation before trying to read a task dump.

12.4.1 Contents of a snap dump

The table below lists the order and contents of a DC formatted snap dump. All of the following information is listed if the ALL parameter of the SNAP command is specified.

Structure	Title starts with	Notes
Summary of all resources for all active tasks	SYSTEM PHOTO	Always listed with task and system snaps unless PHOTO disabled
System registers User registers	SYSTEM REGISTERS	Task and system snaps
System trace entries	TRACE ENTRIES, ORDERED OLDEST TO NEWEST	Task snaps that result from program checks and system snaps
Abend control element (ACE) including PSW, data at PSW, and registers	ABEND C.E.	Task snaps that result from program checks and system snaps
Maps of region and nucleus	MAP OF REGION	System snaps
Task's TCE	TASKS TCE ADDRESS	Task and system snaps
Task's DCE	TASKS DCE ADDRESS	Task and system snaps
Task's LTE	TASKS LTE ADDRESS	Task and system snaps
Task's PTE	TASKS PTE ADDRESS	Task and system snaps
Task's PLE	TASKS PLE ADDRESS	Task and system snaps
Task's resources	TASKS RESOURCE CHAIN	Task and system snaps
Options	OPTIONS ADDRESS	Task snaps that result from program checks and system snaps

Structure	Title starts with	Notes
CCE	CCE ADDRESS	Task snaps that result from program checks and system snaps
SVC parms	SVC PARMS ADDRESS	Task snaps that result from program checks and system snaps
ESE	ESE ADDRESS	System snaps
ERE area	ERE AREA ADDRESS	System snaps
CSA	CSA ADDRESS	Task snaps that result from program checks and system snaps
TCA header	TCA HEADER ADDRESS	System snaps
DCE area	DCE AREA ADDRESS	System snaps
TCE area	TCE AREA ADDRESS	System snaps
RCA header	RCA HEADER ADDRESS	System snaps
RLE area	RLE AREA ADDRESS	System snaps
RCE area	RCE AREA ADDRESS	System snaps
DPE area	DPE AREA ADDRESS	System snaps
Loader DCBs	LOADER DCBS	System snaps
LTERM table	LTERM TABLE ADDRESS	System snaps
PLE, OS/390 storage, PTEs, sets	PHYSICAL LINE ENTRY	System snaps
Task table	TASK TABLE ADDRESS	System snaps
Queue table	QUEUE TABLE ADDRESS	System snaps
Destination table	DEST TABLE ADDRESS	System snaps
STG headers (SCTs and SCEs)	STG TBL HDR ADDRESS	System snaps
All storage pools (0 through nnn)	STORAGE POOL NNN	System snaps
Program tables (PDTs and PDEs)	PGM TABLES ADDRESS	System snaps
All program pools present in the system	<small>24 BIT PROGRAM POOL 24 BIT REENTRANT POOL 31 BIT PROGRAM POOL 31 BIT REENTRANT POOL</small>	System snaps
Run unit table	SYS-RU-TAB ADDRESS	System snaps
Extent table	SYS-EXT-TAB ADDRESS	System snaps

Structure	Title starts with	Notes
DMCL table	DMCL TABLE ADDRESS	System snaps
Operating-system-dependent module	OSXX MODULE ADDRESS	System snaps
Nucleus modules	NUCLEUS ADDRESS	System snaps (reentrant systems only)
SVC module	SVC MODULE ADDRESS	System snaps (reentrant systems only)
Drivers	DRIVERS ADDRESS	
DBIO module	DBIO MODULE ADDRESS	System snaps (reentrant systems only)
DBMS module	DBMS MODULE ADDRESS	System snaps (reentrant systems only)

12.4.2 How to use the dump

This section tells you the items to look at first to use a DC formatted task dump efficiently.

Abend message: The abend message that precedes the dump tells you the name of the abending program and the offset of the abend:

```
IDMS DC027001 V12 T3891 D003 PROGRAM CHECK IN SOC7TST AT OFFSET C8E
      PSW WAS 079D1E00 002C3C8E DUMP OF TASK FOLLOWS
IDMS DC027009 V12 T3891 15:24:07 95.111 CURRENT TASK CODE IS TSK01
IDMS DC027010 V12 T3891 CURRENT LTE ID IS LT12008
IDMS DC027011 V12 T3891 CURRENT USER ID IS RKN
```

System photo: The system photo (if provided) tells you the abending program's resource control element (RCE) address:

```
*** SYSTEM PHOTO WHEN *TASK* SNAP REQUESTED ***
RELEASE: CA-IDMS 15.0 TAPE: C09506 OP SVS: OS/390
TASK CODE: *SYSTEM* TASK ID: 00000000 DISP PRI: 00000255 PROGRAM: *MASTER* LTERM: *N/A*
RESOURCES: RCE ADDR RCE TYP RESOURCE RES ADDR RESOURCE INFO
            000EA724 STORAGE STORAGE 001BFC80 LENGTH 00002380
            :
            :
            :
TASK CODE: TSK01 TASK ID: 00003891 DISP PRI: 00000050 PROGRAM: SOC7TST LTERM: LT12008
RESOURCES: RCE ADDR RCE TYPE RESOURCE RES ADDR RESOURCE INFO
            000E2F4 STORAGE STORAGE 001C3540 LENGTH 00000040
            000EC6EC STORAGE STORAGE 00125940 LENGTH 000000C0
            000EAF04 STORAGE STORAGE 00125000 LENGTH 00000940
            000EB0A4 STORAGE STORAGE 001B8E80 LENGTH 00000100
            000EBBDC STORAGE STORAGE 001B8F80 LENGTH 00000080
            000EC11C STORAGE STORAGE 001BECC0 LENGTH 00000080
            000ED97C QUEUE QCE ADDR 001C3548 LENGTH 00000000
            >>> 000EB0E4 PROGRAM PROG ADD 002C3000 PDE ID SOC7TST
```

Abend control element: The abend control element (ACE). Note the program status word (PSW), the data at the PSW, and the user mode registers (0-15):

►► For more information, including a control block cross reference, refer to *CA-IDMS DSECT Reference Guide*.

12.5 Error checking

The table below presents a brief list of typical programming errors and possible solutions.

Problem	Language	Reason and/or action
Unconnected member records are showing up as set members.	All	Be sure to issue an IF MEMBER statement before OBTAIN OWNER in a set with the optional or manual set membership options.
Skipping records in an area sweep.	All	Your processing may have taken you to another database page; be sure to issue a FIND CURRENT record-name before issuing an OBTAIN NEXT WITHIN area-name.
A program performs extra processing in addition to IDMS-STATUS.	COBOL	<p>IDMS-STATUS is a COBOL SECTION; be sure to do one of the following:</p> <ul style="list-style-type: none"> ■ Place IDMS-STATUS at the end of the program. ■ Ensure that the code following IDMS-STATUS is also a SECTION. ■ Always perform IDMS-STATUS THRU ISABEX.
Queue records written to the database are not being kept.	DC-BATCH	Be sure to issue either FINISH TASK or COMMIT TASK, or the queue records will be deleted at the end of the run unit.
Storage violation when initializing acquired storage fields.	DC	When you define acquired storage length by using the THROUGH option, CA-IDMS does not acquire storage for the dummy byte. If you initialize fields on the group level, it may include the dummy byte, thus causing a storage violation.
Map displayed with no variables.	DC	Be sure to issue a BIND MAP statement for the map and BIND MAP RECORD statements for all map records.

Problem	Language	Reason and/or action
The same errors occur despite repeated modification and recompilation.	DC	Be sure to issue a DCMT VARY PROGRAM NEW COPY statement following recompilation. Be sure to VARY the correct version of the program.

Appendix A. PL/I Considerations

- A.1 About this appendix A-3
- A.2 Transferring control A-4
- A.3 Using the Online Debugger with PL/I A-5
 - A.3.1 Computation Phase A-5
 - A.3.2 Sample Online Debugger Session A-6

A.1 About this appendix

This appendix explains:

- Passing parameters to PL/I programs in the TRANSFER CONTROL command, which differs from the procedure in the COBOL environment.
- Debugging a PL/I program using the CA-IDMS Online Debugger

A.2 Transferring control

In order to pass parameters in a LINK or XCTL statement, you must include the following PL/I declarative in your program:

```
DECLARE IDMSP ENTRY;
```

If you pass parameters to a PL/I program from a non-PL/I program (CA-ADS dialog, or a COBOL or Assembler program), you must include special parameters in order to establish addressability to the passed data.

The program excerpt below shows the extra code necessary to transfer from a non-PL/I program to a PL/I program.

The parameters F1, F2, and F3 provide the addresses on which to base the structures that are passed.

```
TESTPROC: PROCEDURE (F1,F2,F3) OPTIONS (MAIN,REENTRANT);
DCL (EMPSS01T SUBSCHEMA, EMPSCHEM SCHEMA) MODE (IDMS_DC) DEBUG;
DCL IDMS ENTRY OPTIONS (INTER,ASM);
DCL IDMSP ENTRY;
DCL ADDR BUILTIN;
DCL (F1,F2,F3) FIXED;
DCL PASSED_FIELD_1 FIXED BIN(31) BASED (ADDR(F1));
INCLUDE IDMS (SUBSCHEMA_CTRL BASED (ADDR(F2)));
INCLUDE IDMS (RECORD_AA_BASED (ADDR(F3)));
.
.
.
```

►► For more information on passing parameters to a PL/I program from an Assembler program, refer to *CA-IDMS DML Reference - PL/I*.

A.3 Using the Online Debugger with PL/I

You can use the online debugger to detect, trace, and resolve programming errors in DC PL/I programs. To use the online debugger with PL/I, you should be familiar with both hexadecimal notation and hexadecimal arithmetic.

The phases of the debugging process are discussed below, followed by a sample PL/I online debugger session.

A.3.1 Computation Phase

Before beginning the debugging process, it is a good idea to determine the breakpoints you want to set and the storage locations you want to examine:

- To determine the hexadecimal offset of an **executable program instruction** at which you wish to set a breakpoint, perform the following steps:
 1. Examine the cross-reference table portion of your link-edit listing for an entry in the form *program-name1*. Record the hexadecimal offset listed under ORIGIN:

CROSS REFERENCE TABLE					
CONTROL SECTION	NAME	ORIGIN	LENGTH	ENTRY NAME	LOCATION
PLISTART		00	50	PLICALLA	6
PLIMAIN		50	8		
*PLIPROG2		58	394		
*PLIPROG1		3F0	EB4	PLI3PROG	3F8
IDMSPLI		12A8	284		

2. Examine the PL/I compiler portion of your listing and record the line number of the statement at which you wish to set the breakpoint:

```

133          WORK_LAST = EMP_LAST_NAME_0415;
134          WORK_FIRST = EMP_FIRST_NAME_0415;
                                     /*
MAP OUT (DCTEST01) OUTPUT DATA YES
MESSAGE (INITIAL_INSTRUCTIONS_MSG_1)
LENGTH (25)
DETAIL NEW KEY (DBKEY).
                                     */
135          /* IDMS PL/I DML EXPANSION */ DO;
136          DML_SEQUENCE=0013;
137          DCCFLG1=0;
138          DCCFLG1=13;
139          DCCFLG2=16;
140          DCCFLG3=0;
141          DCCFLG4=4;
142          DCCFLG5=72;
143          DCCFLG6=0;

```

3. Examine the Assembler listing generated by the LIST option, locate the previously recorded PL/I line number, and record its corresponding hexadecimal displacement value:

```

* STATEMENT NUMBER 136
0006AA 41 80 7 21C          LA  8,SUBSCHEMA_CTRL.D
                                CCALIGN_AREA.FILLE
                                R0001
0006AE 58 40 3 124          L   4,292(0,3)
0006B2 50 40 8 008          ST  4,SSC_ERRSAVE_AREA
                                .DML_SEQUENCE

```

4. Add the origin offset and the breakpoint instruction's hexadecimal displacement to obtain the breakpoint address:

$X'3F0' + X'6AA' = X'A9A'$

- To determine the offset of **AUTOMATIC variables**, locate the variable storage map and record the displacement value for each variable you wish to examine during the debugging process:

MAP_WORK_REC	1	796	31C	AUTO
WORK_DEPT_ID	1	796	31C	AUTO
WORK_EMP_ID	1	800	320	AUTO
WORK_FIRST	1	804	324	AUTO
WORK_LAST	1	814	32E	AUTO
WORK_ADDRESS	1	829	33D	AUTO
WORK_STREET	1	829	33D	AUTO
WORK_CITY	1	849	351	AUTO
WORK_STATE	1	864	360	AUTO
WORK_ZIP	1	866	362	AUTO
WORK_DEPT_NAME	1	871	367	AUTO

You locate AUTOMATIC variables at run time through Register 13.

- To determine the location of **STATIC INTERNAL variables**, examine the static internal storage map to find the hexadecimal offset for each variable you wish to examine during the debugging process.

You locate STATIC INTERNAL variables at run time through Register 3.

A.3.2 Sample Online Debugger Session

To use the online debugger with an DC PL/I program, perform the following steps:

1. Compile the program with the LIST, OFFSET, XREF STORAGE, and MAP compiler options before defining it to the DC system.
2. Record breakpoint and storage displacements as explained in A.3.1, "Computation Phase" on page A-5 above.
3. Initiate the debugger session by entering the DEBUG task code from DC; the DEBUG> prompt is displayed, indicating that the debugger is in control:

```

ENTER NEXT TASK CODE:
debug
DEBUG>

```

4. Indicate the program to be debugged by entering DEBUG followed by the program name; the debugger verifies the program name:

```

DEBUG>
debug pliprogram
DEBUG PLIPROG
DEBUG> DEBUGGING INITIATED FOR PLIPROG VERSION 1
DEBUG>

```

5. Establish breakpoints by issuing the AT command, followed by \$, which signifies the base register, followed by the previously computed breakpoint address; the debugger verifies the establishment of the breakpoint:

```
DEBUG>
at $ + @a9a

AT @A9A
AT> @A9A ADDED
DEBUG>
```

6. After all breakpoints have been set, leave the setup phase of the debugger session by issuing the EXIT command:

```
DEBUG>
exit
```

7. Initiate the run-time phase by issuing the task code that invokes the task that the program participates in:

```
ENTER NEXT TASK CODE:
deptmod
```

8. When a breakpoint is encountered at run time, the debugger assumes control and identifies the address, program, and the debugger expression that was used to establish the breakpoint:

```
AT OFFSET @A9A IN PLIPROG EXPRESSION @BDE
DEBUG>
```

9. You can now examine program variable storage by issuing LIST commands; indirect addressing is used based on the previously noted register and offset:

```
list %:r13 + @31c 32

LIST %:R13 + @31C 32
001DB7F4 F3F2F0F0 F0F0F0F4 C8C5D9C2 C5D9E340 *32000004HERBERT *
001DB804 4040C3D9 C1D5C540 40404040 40404040 * CRANE *
```

If your program contains any nested procedures or begin blocks, you will need to navigate the chain of dynamic storage areas (DSAs) to obtain the correct variable-storage base address. To navigate the DSA chain for nested procedures or begin blocks, list the contents of register 13 to determine the DSA for the current level of nesting:

```
list %:r13

LIST %:R13
001C7A30 84200000 001C7948 00000000 5E422A20 *D.....;...*
```

For subsequent levels of nesting, perform the following step:

- a. List the absolute address contained 4 bytes off of the previously displayed line:

```
list @1c7948

LIST @1C7948
001C7948 84200000 001C74D8 00000000 4E4227EC *D.....Q.....*
```

- b. When you have reached the final level of nesting, use the address 4 bytes off of the display as the base address to list AUTOMATIC variable-storage values:

```
DEBUG>
list 1c74d8 + @31c 32

LIST 1C74D8 + @31C 32
001C77F4 F3F2F0F0 F0F0F0F4 C8C5D9C2 C5D9E340 *32000004HERBERT *
001C7804 4040C3D9 C1D5C540 40404040 40404040 * CRANE *
```

To examine variables defined as BASED storage, perform the following steps:

- a. Using indirect addressing, list the contents of the associated pointer variable:

```
DEBUG>
list %:r13 + @d4

LIST %:R13 + @D4
001499E0 00149AC8 00000000 00000000 00000000 *...H.....*
```

- b. List the absolute address to display the BASED variable's values:

```
DEBUG>
LIST @149ac8 16
00149AC8 F1F1F1F1 C4C5D7E3 00000000 00000000 *1111DEPT.....*
```

10. To continue program execution, enter the RESUME command:

```
DEBUG>
resume
```

11. To end a debugger session, enter the QUIT command from the DEBUG> prompt:

```
DEBUG>
quit

QUIT
QUIT DEBUGGER
ENTER NEXT TASK CODE:
```

Appendix B. Assembler Considerations

B.1 About this appendix B-3

B.1 About this appendix

This appendix explains the following Assembler topics, which differ from the COBOL environment:

- Batch error checking — A discussion on coding error-checking routines for batch programs
- DC error checking — A discussion on coding error-checking routines for online programs

►► For more information related to using Assembler with DC, refer to *CA-IDMS DML Reference - Assembler*.

Checking the status of calls to DB: Assembler programs do not use the IDMS-STATUS block; you must explicitly code your own error-checking routines. You should check the ERRSTAT field after every DB DML call; if the DBMS returns an unexpected nonzero value, you should:

1. Display the following IDMS communications block fields:
 - PGMNAME
 - ERRSTAT
 - ERRORREC
 - ERRORSET
 - ERRAREA
 - RECNAME
 - AREANAME
 - DMLSEQ (if the DEBUG option is specified)

You should also display any other relevant variable-storage fields.

2. Issue the @ROLLBAK command.
3. Terminate the program.

Checking the status of calls to DC: Assembler DC programs do not need to use the IDMS-DC communications block. You explicitly check the value returned to register 15 to determine the result of a DC call. If the call to DC included database access, you must check:

- Register 15 for return codes issued by DC
- The ERRSTAT field for status codes issued by DB

If DC returns an unexpected nonzero status, you should:

1. Save the register 15 value

2. Write a memory dump of the IDMS communications block and any other relevant variable-storage fields by using the #SNAP command
3. Terminate the program by using the #ABEND command

Testing for the return code in register 15 is not usually necessary because most Assembler DML commands have options that take action based on the return code value.

Appendix C. Batch Access to DC Queues and Printers

C.1 About this appendix C-3

C.1 About this appendix

This appendix explains how a batch application program can use services of the CA-IDMS central version.

DC-BATCH mode: DC-BATCH allows your batch program to access DC queues and printers. Using DC-BATCH, your program can access the database, issue CA-IDMS queue management commands, and transmit data to DC printers.

Note: DC-BATCH uses the IDMS communications block.

Batch access to CA-IDMS queues: You can use DC-BATCH to establish a task within a batch application program. This allows your program to read data from queue records while performing normal database activities. Additionally, you can take advantage of DC facilities for locking queue records and performing recovery.

Perform the following steps to access queue records from a DC-BATCH program:

1. Link edit the program with the batch interface module, IDMS.
2. Specify a mode of DC-BATCH.
3. Initiate the DC task by issuing a BIND TASK statement before any other BIND statements. BIND TASK establishes communication with the DC system and allocates a packet-data-movement buffer to contain the queue data.
4. Issue retrieval and modification statements beginning with BIND RUN-UNIT and ending with FINISH. Within a task, you can code as many BIND/READY/FINISH sequences as required.
5. Issue GET QUEUE, PUT QUEUE, and DELETE QUEUE statements to access queue records. Queue access requests must fall between the BIND TASK and the FINISH TASK statements; they need not fall between BIND RUN-UNIT and FINISH.
6. Terminate the DC task by issuing a FINISH TASK statement. FINISH TASK relinquishes control over all database areas associated with the task and establishes an end-of-task checkpoint in the journal file for the queue areas that have been accessed by the task.

Within the task, you can issue COMMIT TASK and ROLLBACK TASK statements to write checkpoints and effect recovery coordinated with the CA-IDMS run unit.

Note: Be sure to issue a BIND TASK statement and a FINISH TASK statement and to include the TASK parameter of the COMMIT and ROLLBACK statements.

Batch access to DC printers: To access DC printers from a batch application program, perform the following steps:

1. Link edit the program with the batch interface module, IDMS.
2. Specify a mode of DC-BATCH.
3. Issue a BIND TASK statement.

4. Issue WRITE PRINTER requests as needed to build a report and direct it to a printer.

Note: Batch programs cannot issue WRITE PRINTER SCREEN requests.

5. Terminate the DC task with a FINISH TASK statement.

Appendix D. XA Considerations

D.1 About this appendix D-3

D.1 About this appendix

This appendix explains how your program can use XA features.

XA support: CA-IDMS supports XA for Assembler and VS COBOL II programs. To run your application in 31-bit mode, the following conditions must be met:

- Your program must be able to run above the 16-megabyte line. For more information, refer to the appropriate IBM documentation.
- Your DC system must contain at least one XA program pool, XA reentrant pool, and XA storage pool.
- You must link edit your program with the following options:
RMODE=ANY,AMODE=31
- You must define the task, either at sysgen or by using a DCMT VARY DYNAMIC TASK command, with the LOCATION=ANY parameter.

Appendix E. Running a Program Under TCF

- E.1 About this appendix E-3
- E.2 Overview of TCF E-4
- E.3 Defining a TCF task to the DC system E-6
- E.4 Using the UCE for communication under TCF E-7
- E.5 Determining if TCF is active E-9
- E.6 Starting a new session E-10
- E.7 Resuming a suspended session E-11
- E.8 Processing a pseudoconverse E-12
 - E.8.1 Suspend processing E-12
 - E.8.2 End processing E-12
 - E.8.3 Switch processing E-12
- E.9 Displaying error messages E-14
- E.10 Sample application under TCF E-15

E.1 About this appendix

This appendix explains the processing that your program must perform in order to run under the Transfer Control Facility (TCF). TCF allows you to transfer from one online application to another without having to return first to DC.

E.2 Overview of TCF

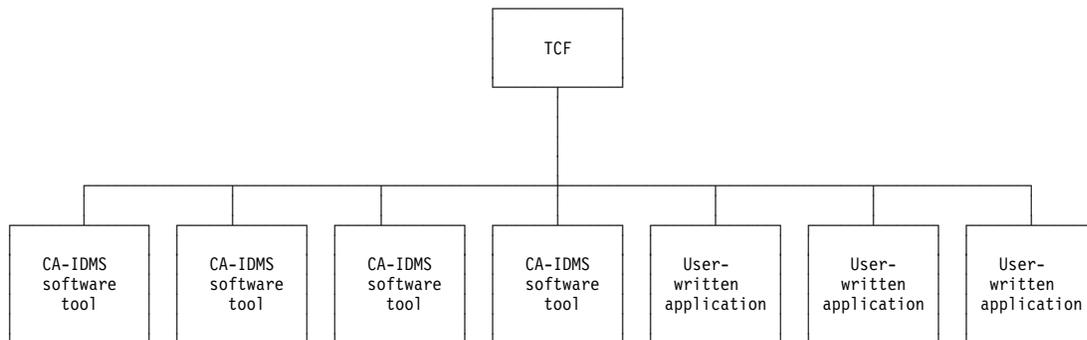
Using TCF, you can suspend a session of an online application, transfer directly to another online application, then transfer back and resume the suspended session.

Before writing an application to run under TCF, you should be thoroughly familiar with TCF and the CA-IDMS software tools that it invokes.

► For more information on TCF, refer to *CA-IDMS Transfer Control Facility*.

TCF internal processing: You should be aware of TCF internal processing:

- TCF invokes an application through a TRANSFER CONTROL LINK function. This allows TCF to regain control after every pseudoconverse and every DC RETURN statement. The figure below shows a typical TCF program structure.



- All communication between an application and TCF occurs through the universal communications element (UCE). The UCE is the link between an application and TCF. Each application and TCF set fields in the UCE that indicate specific actions to be taken.

The record layout of the UCE is presented below.

Note: Be sure to copy **version 2** of the UNIVERSAL-COMMUNICATIONS-ELEMENT. Depending on the language, you may need to define a synonym with a shorter name.

```

01 UNIVERSAL-COMM-ELEMENT.
03 UCE-IDENT-02          PIC XXXX.
03 UCE-DBNAME-02        PIC X(8).
03 UCE-NODE-NAME-02     PIC X(8).
03 UCE-DICT-NAME-02     PIC X(8).
03 UCE-DICT-NODE-02     PIC X(8).
03 UCE-SCHEMA-NAME-02   PIC X(8).
03 UCE-SCHEMA-VER-02    PIC 9999  USAGE COMP.
03 UCE-SUBSCHEMA-NAME-02 PIC X(8).
03 UCE-SUBSCHEMA-VER-02 PIC 9999  USAGE COMP.
03 UCE-INPUT-POINTER-02 PIC S9(8)  USAGE COMP.
03 UCE-INPUT-LENGTH-02 PIC S9(8)  USAGE COMP.
03 UCE-OUTPUT-POINTER-02 PIC S9(8)  USAGE COMP.
03 UCE-OUTPUT-LENGTH-02 PIC S9(8)  USAGE COMP.
03 UCE-ENTITY-OCCURRENCE-02 PIC X(32).
03 UCE-ENTITY-OCCUR-VER-02 PIC 9999  USAGE COMP.
03 FILLER                PIC XX.
03 UCE-ACTION-CODE-02    PIC XXXX.
03 UCE-RETURN-CODE-02   PIC S9(8)  USAGE COMP.
03 UCE-MSG-CODE-02      PIC 9(7)   USAGE COMP-3.
03 UCE-MSG-TEXT-POINTER-02 PIC S9(8)  USAGE COMP.
03 FILLER                PIC X(32).
03 UCE-SYS-INIT-TIME-02  PIC S9(8)  USAGE COMP.  DO NOT MODIFY
03 UCE-FROM-TASK-02     PIC X(8).          DO NOT MODIFY
03 UCE-ACTIVE-TASK-02   PIC X(8).          DO NOT MODIFY
03 UCE-NEXT-TASK-02    PIC X(8).          DO NOT MODIFY
03 UCE-ENTRY-TASK-02   PIC X(8).          DO NOT MODIFY
03 UCE-PT-LIST-POINTER-02 PIC S9(8)  USAGE COMP.  DO NOT MODIFY
03 UCE-NBR-TASKS-02    PIC S9999  USAGE COMP.  DO NOT MODIFY
03 UCE-NBR-SESSIONS-02 PIC S9999  USAGE COMP.  DO NOT MODIFY
03 UCE-QUEUE-ID-02     PIC S9(8)  USAGE COMP.  DO NOT MODIFY
03 UCE-SESSION-DESCR-02 PIC X(16).
03 UCE-CURR-TASK-FLAG-02 PIC X.
                        88 UCE-SUSPEND-02 VALUE 'S'.
                        88 UCE-END-02   VALUE '0'.
                        88 UCE-CONVERSE-02 VALUE 'P'.
03 UCE-NEXT-TASK-FLAG-02 PIC X.
                        88 UCE-NEW-02   VALUE 'N'.
                        88 UCE-RESUME-02 VALUE '0'.

```

- Your program is responsible for saving its own variable storage in the form of a queue or a scratch record when performing **suspend** processing. Suspend processing can be either implicit or explicit:
 - A TCF user **implicitly** suspends an application by switching to another TCF application.
 - A TCF user **explicitly** suspends an application by issuing a SUSPEND command.

Additionally, a TCF user can suspend an entire TCF session by issuing a SWITCH SUSPEND command.

E.3 Defining a TCF task to the DC system

To make your task eligible to run under TCF, you need to use the system generation TASK statement to define the task code that invokes your program under TCF. Include the following parameters:

- TCF TASK IS TCF — Enables your task to run under TCF
- PRODUCT CODE IS — Identifies a generic TCF task code for your task

For example:

```
TASK EMPTSKT INVOKES EMPPRG INPUT SAVE
      TCF TASK IS TCF
      PRODUCT CODE IS EMPTSK.
```

TCF tasks must be defined at system generation; they cannot be defined dynamically.

►► For more information about system generation, refer to *CA-IDMS System Generation*.

E.4 Using the UCE for communication under TCF

You use fields in the UCE:

- To communicate with TCF
- To communicate with other applications running under TCF

Communicating with TCF: When your program begins, it checks certain UCE fields to determine invocation conditions. You perform processing based on how your program is invoked.

When your program ends, it sets UCE fields to tell TCF what to do next; for example, whether to switch to another application, perform a pseudoconverse, suspend the TCF session, or end the TCF session.

Use the following fields in the UCE to communicate with TCF:

- UCE-IDENT-02 indicates whether your program is currently running under TCF. If it is, this field is equal to the value UMBR.
- UCE-SESSION-DESCR-02 is used as a queue or scratch record ID. Use this ID to retrieve the variable storage from your application's previously suspended TCF session.
- UCE-NEXT-TASK-02 specifies the task to which TCF should switch.
- UCE-CURR-TASK-FLAG-02 is the flag you set to indicate to TCF whether to suspend a session, end a session, or begin a pseudoconverse.
- UCE-NEXT-TASK-FLAG-02 is the flag you set to indicate to TCF and to other applications whether to begin a new session or restart an old session.
- UCE-MSG-CODE-02, UCE-RETURN-CODE-02, and UCE-MSG-TEXT-POINTER-02 are used to process errors under TCF.

Communicating with other applications: You can pass data to and receive data from other applications that run under TCF. For example, you could pass a schema name, a subschema name, syntax, or input parameters.

The UCE provides different fields for different kinds of data:

- To pass a schema name, use the UCE-SCHEMA-NAME-02 field. Optionally, include a version number by using the UCE-SCHEMA-VER-02 field.
- To pass a subschema name, use the UCE-SUBSCHEMA-NAME-02 field. Optionally, include a version number by using the UCE-SUBSCHEMA-VER-02 field.
- To pass large amounts of data (33 bytes or more), use the UCE-INPUT-POINTER-02 field. UCE-INPUT-LENGTH-02 specifies the input data length.

You can also use the UCE-OUTPUT-POINTER-02 field.
UCE-OUTPUT-LENGTH-02 specifies the output data length.

- To pass small amounts of data (32 bytes or less), use the UCE-ENTITY-OCCURRENCE-02 field. In some situations, you may need to include a version number by using the UCE-ENTITY-OCCUR-VER-02 field.

E.5 Determining if TCF is active

If TCF has invoked the task, the UCE-IDENT-02 field contains the literal UMBR. If UMBR is not present, the session was not invoked by TCF.

COBOL example: For example, in COBOL:

```
PROCEDURE DIVISION USING UNIVERSAL-COMM-ELEMENT.  
  IF UCE-IDENT-02 NOT = 'UMBR'  
    THEN GO TO A100-NON-TCF-SESSION.
```

Assembler example: Assembler programs check register 1 to determine if they are being invoked under TCF. Register 1 points to a one-entry parameter list that points to the UCE. If TCF has invoked the task, the first four bytes of the UCE contain the literal UMBR. For example:

```
      LTR  R1,R1                UNDER TCF?  
      BZ  NONTCF              NO, GO ON  
      L   R2,0(R1)            MAYBE  
      CLC 0(4,R2),=CL4'UMBR'  CHECK FOR 'UMBR'  
      BNE NONTCF              NO, GO ON  
+ *** TCF PROCESSING ***
```

E.6 Starting a new session

To start a new session under TCF, perform the following steps:

1. Check the following fields for data:
 - UCE-INPUT-POINTER-02 typically points to syntax to be used to start a new session of your application. To determine the data length, refer to UCE-INPUT-LENGTH-02.
 - UCE-ENTITY-OCCURRENCE-02 contains an entity name to be used to start a new session of your application. To determine the version number, refer to UCE-ENTITY-OCCUR-VER-02.

If either of these fields contains data, you should start a new session as specified by the passed data.

2. Check the UCE-NEXT-TASK-FLAG-02 field and perform processing as follows:
 - If **N** (new) is specified, you should start a new session using the DBNAME and NODENAME fields from the UCE.
 - If **O** (old) is specified, you should resume the previously suspended session as explained in E.7, “Resuming a suspended session” on page E-11 later in this section.
3. Perform processing, as required.
4. When processing is complete, move 'P' (pseudoconverse) to UCE-CURR-TASK-FLAG-02 to indicate to TCF that a pseudoconverse is to take place.
5. Issue a DC RETURN statement.

E.7 Resuming a suspended session

To resume a previously suspended session, perform the following steps:

1. Use the session descriptor (UCE-SESSION-DESCR-02) as the queue or scratch ID to retrieve the variable storage needed to resume the session. If the storage area cannot be found, perform the following steps:
 - a. Move +4 to UCE-RETURN-CODE-02.
 - b. Issue a DC RETURN statement.

This returns control to TCF and displays an error message on the TCF Error Message screen.
2. Perform processing, as required.
3. When processing is complete, move 'P' (pseudoconverse) to UCE-CURR-TASK-FLAG-02 to indicate to TCF that a pseudoconverse is to take place.
4. Issue a DC RETURN statement.

E.8 Processing a pseudoconverse

At the beginning of a pseudoconverse, you should check a user-defined map field for the following:

1. Does the TCF user want to **suspend** the session?
2. Does the TCF user want to **quit** the session?
3. Does the TCF user want to **switch** to another application that runs under TCF?

If none of the above is specified, you should perform processing, as required.

E.8.1 Suspend processing

If the TCF user wants to suspend a session, perform the following steps:

1. Move the name of the session descriptor to UCE-SESSION-DESCR-02.
2. Save program variable storage as either a scratch or a queue record using the session descriptor as the scratch or queue ID.
3. Move 'S' (suspend) to UCE-CURR-TASK-FLAG-02.
4. Issue a DC RETURN statement.

E.8.2 End processing

If the TCF user wants to end a session, perform the following steps:

1. Move 'O' (off) to UCE-CURR-TASK-FLAG-02.
2. Issue a DC RETURN statement.

E.8.3 Switch processing

If the TCF user issues any form of the SWITCH command, perform the following steps:

1. Move the name of the session descriptor to UCE-SESSION-DESCR-02.
2. Save program variable storage as either a scratch or a queue record using the site-standard session descriptor as the scratch or queue ID.
3. Perform the following steps:
 - **If no task code or product code is specified**, move spaces to UCE-NEXT-TASK-02.
 - **If a task code or product code is specified**, move that code to UCE-NEXT-TASK-02.

If the TCF user specifies the NEW option of the SWITCH command, move 'N' (new) to UCE-NEXT-TASK-FLAG-02; otherwise move 'O' (old) to UCE-NEXT-TASK-FLAG-02.

Note: If the TCF user specifies new, you may need to pass data to the switched-to task using either the UCE-INPUT-POINTER-02 or UCE-ENTITY-OCCURRENCE-02 fields in the UCE or some other site-standard convention.

4. Issue a DC RETURN statement.

E.9 Displaying error messages

At times, you may want to intentionally abend your task by issuing a WRITE LOG statement with a given severity code. To do this under a TCF, perform the following steps:

1. Move the message number to UCE-MSG-CODE-02.
2. Move -1 to UCE-RETURN-CODE-02.
3. Issue a GET STORAGE statement to acquire the storage that is to contain the message text.
4. Issue a WRITE LOG statement. Include the RETURN TEXT option and specify the previously acquired storage.
5. Move the address of the message text into UCE-MSG-TEXT-POINTER-02.

Note: COBOL programs can do this by calling an Assembler subroutine.

6. Issue a DC RETURN statement.

This returns control to TCF and displays the error message on the TCF Error Message screen.

E.10 Sample application under TCF

The program below performs processing that enables it to run under TCF.

This program checks TCF-related fields in the UCE and performs TCF processing before performing any application-specific processing.

```
WORKING-STORAGE SECTION.  
01 WS-START PIC X(10) VALUE '*WS START*'.  
01 USER-IDENT.  
   05 USER-ID-FIRST-EIGHT PIC X(8).  
   05 USER-ID-REST PIC X(24).  
01 TASK-ID PIC X(8).  
01 SESSION-DESC-WORK.  
   05 SDW-1 PIC X(8).  
   05 SDW-2 PIC X(8).
```

```

01 TCF-REC.
  02 TCF-REC-COMMLINE      PIC X(7).
                           88 SUS-COMMAND VALUE 'SUS'
                           'SUSP' 'SUSPE' 'SUSPEN'
                           'SUSPEND'.
                           88 BYE-COMMAND VALUE 'BYE'
                           'QUIT' 'QUI' 'END'.
                           88 SWITCH-COMMAND VALUE 'SWI'
                           'SWIT' 'SWITC' 'SWITCH'.
  02 TCF-REC-QUIT          PIC X VALUE ' '.
  02 TCF-REC-SUSPEND       PIC X VALUE ' '.
  02 TCF-REC-SWITCH        PIC X VALUE ' '.
  02 TCF-REC-HELP          PIC X VALUE ' '.
  02 TCF-REC-SWI-TASK      PIC X(8).
  02 TCF-REC-OLDNEW        PIC X.
                           88 SWI-OLD VALUE 'O'.
                           88 SWI-NEW VALUE 'N'.

01 DATA-REC.
  02 DATA-REC-FIELD1      PIC X VALUE ' '.
  02 DATA-REC-FIELD2      PIC X VALUE ' '.
  02 DATA-REC-FIELD3      PIC X VALUE ' '.
  02 DATA-REC-FIELD4      PIC X VALUE ' '.
01 WS-END                  PIC X(8) VALUE '*WS END*'.

LINKAGE SECTION.
01 COPY IDMS UNIVERSAL-COMM-ELEMENT VERSION 2.
01 UNIVERSAL-COMM-ELEMENT.
  03 UCE-IDENT-02          PIC XXXX.
  03 UCE-DBNAME-02         PIC X(8).
  03 UCE-NODE-NAME-02      PIC X(8).
  03 UCE-DICT-NAME-02      PIC X(8).
  03 UCE-DICT-NODE-02      PIC X(8).
  03 UCE-SCHEMA-NAME-02    PIC X(8).
  03 UCE-SCHEMA-VER-02     PIC 9999  USAGE COMP.
  03 UCE-SUBSCHEMA-NAME-02 PIC X(8).
  03 UCE-SUBSCHEMA-VER-02  PIC 9999  USAGE COMP.
  03 UCE-INPUT-POINTER-02  PIC S9(8)  USAGE COMP.
  03 UCE-INPUT-LENGTH-02   PIC S9(8)  USAGE COMP.
  03 UCE-OUTPUT-POINTER-02 PIC S9(8)  USAGE COMP.
  03 UCE-OUTPUT-LENGTH-02  PIC S9(8)  USAGE COMP.
  03 UCE-ENTITY-OCCURRENCE-02
                           PIC X(32).
  03 UCE-ENTITY-OCCUR-VER-02
                           PIC 9999  USAGE COMP.
  03 FILLER                 PIC XX.
  03 UCE-ACTION-CODE-02     PIC XXXX.
  03 UCE-RETURN-CODE-02    PIC S9(8)  USAGE COMP.
  03 UCE-MSG-CODE-02        PIC 9(7)  USAGE COMP-3.
  03 UCE-MSG-TEXT-POINTER-02
                           PIC S9(8)  USAGE COMP.

```

```

03 FILLER PIC X(32).
03 UCE-SYS-INIT-TIME-02 PIC S9(8) USAGE COMP.
03 UCE-FROM-TASK-02 PIC X(8).
03 UCE-ACTIVE-TASK-02 PIC X(8).
03 UCE-NEXT-TASK-02 PIC X(8).
03 UCE-ENTRY-TASK-02 PIC X(8).
03 UCE-PT-LIST-POINTER-02 PIC S9(8) USAGE COMP.
03 UCE-NBR-TASKS-02 PIC S9999 USAGE COMP.
03 UCE-NBR-SESSIONS-02 PIC S9999 USAGE COMP.
03 UCE-QUEUE-ID-02 PIC S9(8)
    USAGE COMP.
03 UCE-SESSION-DESCR-02 PIC X(16).
03 UCE-CURR-TASK-FLAG-02 PIC X.
    88 UCE-SUSPEND-02 VALUE 'S'.
    88 UCE-END-02 VALUE '0'.
    88 UCE-CONVERSE-02 VALUE 'P'.
03 UCE-NEXT-TASK-FLAG-02 PIC X.
    88 UCE-NEW-02 VALUE 'N'.
    88 UCE-RESUME-02 VALUE '0'.

PROCEDURE DIVISION USING UNIVERSAL-COMM-ELEMENT.
MAIN-LINE.
*** CHECK FOR TCF SESSION
    IF UCE-IDENT-02 NOT = 'UMBR'
        THEN GO TO C100-SESSION.
*** MOST LIKELY PSEUDO-CONV
    IF UCE-CONVERSE-02
        THEN GO TO A100-PSEUDOCONVERSE.
*** NOT PCONV, DATA SENT?
    IF UCE-INPUT-POINTER-02 NOT = 0 OR
        UCE-ENTITY-OCCURRENCE-02 NOT = SPACES
        THEN
            GO TO A100-START-WITH-DATA.
*** NEW SESSION SPECIFIED?
    IF UCE-NEW-02
        THEN GO TO A100-START-NEW-SESSION
*** ELSE DEFAULT TO OLD
    ELSE
        GO TO A100-START-OLD-SESSION.

A100-PSEUDOCONVERSE.
    BIND MAP TCFMAP01.
    BIND MAP TCFMAP01 RECORD TCF-REC.
    BIND MAP TCFMAP01 RECORD DATA-REC.
    ACCEPT USER ID INTO USER-IDENT.
    ACCEPT TASK ID INTO TASK-ID.
*** MENU OR COMMAND-LINE SUSPEND
    IF (TCF-REC-SUSPEND NOT = '_' )
        OR SUS-COMMAND
        THEN
            MOVE USER-ID-FIRST-EIGHT TO SDW-1.
            MOVE TASK-ID TO SDW-2.
            MOVE SESSION-DESC-WORK TO UCE-SESSION-DESCR-02.
*** USE SESS-DESCRIPTOR FOR QID
            PERFORM U100-SAVE-STORAGE
            MOVE 'S' TO UCE-CURR-TASK-FLAG-02
            DC RETURN.

```

```
***                               MENU OR COMMAND-LINE QUIT
  IF (TCF-REC-QUIT NOT = '_' )
    OR BYE-COMMAND
  THEN
    MOVE '0' TO UCE-CURR-TASK-FLAG-02
    DC RETURN.
***                               MENU OR COMMAND-LINE SWITCH
  IF (TCF-REC-SWI-TASK NOT = SPACES)
    OR SWITCH-COMMAND
  THEN
    PERFORM B100-SWITCH
  ELSE
    MOVE 'P' TO UCE-CURR-TASK-FLAG-02
    GO TO C100-SESSION.
*
  A100-START-WITH-DATA.
*** START SESSION USING THE DATA PASSED IN      ***
*** UCE-INPUT-POINTER-02 OR UCE-ENTITY-OCCURRENCE-02 ***
*
  A100-START-NEW-SESSION.
*** START A NEW SESSION, MOVE 'P' ***
*** TO UCE-CURR-TASK-FLAG-02      ***
*
  A100-START-OLD-SESSION.
*** RESTART OLD SESSION, GET PREVIOUS VARIABLE ***
*** STORAGE FROM SCRATCH OR QUEUE AND MOVE 'P' ***
*** TO UCE-CURR-TASK-FLAG-02.      ***
*** IF UCE-SESSION-DESCR-02 IS EMPTY, OR IF ***
*** GET QUEUE/SCRATCH FAILS, MOVE +4 TO ***
*** UCE-RETURN-CODE-02 AND ISSUE A DC RETURN ***
*
  IF UCE-SESSION-DESCR-02 = SPACES
    MOVE +4 TO UCE-RETURN-CODE-02
    DC RETURN.
  GET QUEUE ID UCE-SESSION-DESCR-02
    FROM WS-START
    TO WS-END
    RETENTION 7
  ON ANY-ERROR-STATUS
    MOVE +4 TO UCE-RETURN-CODE-02
    DC RETURN.
  MOVE 'P' TO UCE-CURR-TASK-FLAG-02.
  GO TO C100-SESSION.
```

```
*
B100-SWITCH.
  MOVE USER-ID-FIRST-EIGHT TO SDW-1.
  MOVE TASK-ID             TO SDW-2.
  MOVE SESSION-DESC-WORK   TO UCE-SESSION-DESCR-02.
*                           USE SESS-DESCRIPTOR FOR QID
  PERFORM U100-SAVE-STORAGE.
  IF TCF-REC-SWI-TASK = SPACES
    THEN MOVE SPACES TO UCE-NEXT-TASK-02
    DC RETURN.
  MOVE TCF-REC-SWI-TASK TO UCE-NEXT-TASK-02.
  IF SWI-NEW THEN
    MOVE 'N' TO UCE-NEXT-TASK-FLAG-02
  ELSE
    MOVE '0' TO UCE-NEXT-TASK-FLAG-02.
  DC RETURN.
*
C100-SESSION.
*** PROGRAM PROCESSING ***
*
U100-SAVE-STORAGE.
*** SAVE WORKING STORAGE FROM WS-START TO WS-END ***
*** IN THIS EXAMPLE, ITS A QUEUE RECORD          ***
  PUT QUEUE ID UCE-SESSION-DESCR-02
    FROM WS-START
    TO WS-END
    RETENTION 7.
IDMS-ABORT.
IDMS-ABORT-EXIT.
EXIT.
COPY IDMS IDMS-STATUS.
```

Appendix F. Calls to IDMSIN01

F.1 About IDMSIN01 F-3

F.1 About IDMSIN01

IDMSIN01 is an entry point to the IDMS module that provides various IDMS functions to user programs, including:

- Deactivate the DML trace
- Reactivate the DML trace
- Retrieve 'GETPROF' user profile information
- Establish 'SETPROF' user profile information
- Translate an internal 8 byte DATETIME stamp to displayable form
- Return current DATE and TIME in a displayable form

Note: IDMSIN01 cannot be called from a system mode program such as user exits.

How to call IDMSIN01: You use standard calling conventions to call IDMSIN01. The first two parameters passed are the address of an RPB block and the address of the function REQUEST-CODE and RETURN-CODE fields.

Calls to IDMSIN01 for the functions listed above are shown in this example:

```

ENVIRONMENT DIVISION.

DATA DIVISION.
WORKING-STORAGE SECTION.

01 RPB.
   02 FILLER          PIC X(36).

01 REQ-WK.
   02 REQUEST-CODE   PIC S9(8) COMP.
   02 REQUEST-RETURN PIC S9(8) COMP.

01 WORK-FIELDS.
   02 WK-DTS-FORMAT  PIC S9(8) COMP VALUE 0.
   02 WK-CDTS        PIC X(26).
   02 WK-KEYWD       PIC X(8).
   02 WK-VALUE       PIC X(32).
   02 WK-DBNAME      PIC X(8).

01 USER-WORK-DATA.
   02 WK-SCHEMA      PIC X(18).
   02 WK-CUSER       PIC X(18).
   02 WK-DTS         PIC X(8).

```

```

*****
PROCEDURE DIVISION.
*****

*****
* Call IDMSIN01 to deactivate the DML trace
* which was originally activated by the corresponding
* SYSIDMS parm (DMLTRACE=ON).
*
*   Parm 1 is the address of the RPB.
*   Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*****

      MOVE 1 TO REQUEST-CODE
      CALL 'IDMSIN01' USING RPB REQ-WK.

*****
* Call IDMSIN01 to reactivate the DML trace
* which was originally activated by the corresponding
* SYSIDMS parm (DMLTRACE=ON), which had been previously
* deactivated earlier on in this job.
*
*   Parm 1 is the address of the RPB.
*   Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*****

      MOVE 0 TO REQUEST-CODE
      CALL 'IDMSIN01' USING RPB REQ-WK.

*****
* Call IDMSIN01 to request a 'GETPROF' to get the user
* profile default DBNAME, which was established by the
* SYSIDMS parm DBNAME=xxxxxxx when running 'Mini CV', or
* by the DCUF SET DBNAME xxxxxxxx when running under CV.
*
*   Parm 1 is the address of the RPB.
*   Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*   Parm 3 is the address of the 8 byte GETPROF keyword.
*   Parm 4 is the address of the 32 byte GETPROF returned value.
*****

      MOVE 2 TO REQUEST-CODE
      MOVE 'DBNAME' TO WK-KEYWD
      CALL 'IDMSIN01' USING RPB REQ-WK WK-KEYWD
         WK-VALUE.
      MOVE WK-VALUE TO WK-DBNAME.
      IF WK-DBNAME = SPACES
         DISPLAY 'DBNAME is set to BLANKS'
      ELSE
         DISPLAY 'DBNAME is set to ' WK-DBNAME.

```

```

*****
* Call IDMSIN01 to request a 'SETPROF' to set the user
* profile default SCHEMA to the value 'SYSTEM'.
*
*   Parm 1 is the address of the RPB.
*   Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*   Parm 3 is the address of the 8 byte SETPROF keyword.
*   Parm 4 is the address of the 32 byte SETPROF value.
*****

      MOVE 3 TO REQUEST-CODE
      MOVE 'SCHEMA' TO WK-KEYWD
      MOVE 'SYSTEM' TO WK-VALUE
      CALL 'IDMSIN01' USING RPB REQ-WK WK-KEYWD
      WK-VALUE.
      IF REQUEST-RETURN NOT = 0
      DISPLAY 'SETPROF returned error ' REQUEST-RETURN.
*****
* Call IDMSIN01 to have an 8 byte internal DATETIME stamp
* returned as a displayable 26 character DATE/TIME display.
*
*   Parm 1 is the address of the RPB.
*   Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*   Parm 3 is the address of the 4 byte format indicator (0's).
*   Parm 4 is the address of the 8 byte internal DATETIME stamp.
*   Parm 5 is the address of the 26 byte DATE/TIME returned.
*****

      MOVE 5 TO REQUEST-CODE
      MOVE 'UNKNOWN' TO WK-CDTS
      CALL 'IDMSIN01' USING RPB REQ-WK
      WK-DTS-FORMAT WK-DTS WK-CDTS
      DISPLAY WK-SCHEMA ' ' WK-CUSER ' ' WK-CDTS.

*****
* Call IDMSIN01 to have the current DATE and TIME
* returned as a displayable 26 character DATE/TIME display.
*
*   Parm 1 is the address of the RPB.
*   Parm 2 is the address of the REQUEST-CODE and RETURN-CODE.
*   Parm 3 is the address of the 4 byte format indicator (1).
*   Parm 4 is the address of the 26 byte DATE/TIME returned.
*****

      MOVE 5 TO REQUEST-CODE
      MOVE 1 TO WK-DTS-FORMAT
      CALL 'IDMSIN01' USING SQLRPB REQ-WK
      WK-DTS-FORMAT WK-CDTS
      DISPLAY 'THE DATE AND TIME IS -> ' WK-CDTS.

```

Appendix G. 10.2 Services Batch Interface

G.1 About the 10.2 services batch interface G-3

G.1 About the 10.2 services batch interface

Batch programs that require CA-IDMS 10.2 services only can use the optional 10.2 services batch interface to access a later release.

Since only 10.2 features are available through this interface, later release features such as SYSIDMS parameters and SQL access are not supported through this interface.

This appendix describes the requirements for using the 10.2 services batch interface.

CA-IDMS installation: The 10.2 services batch interface requires two load modules supplied on the CA-IDMS installation tape:

- IDML
- IDMSB102

Usage: To use the 10.2 services batch interface, the following conditions must be met:

- The batch job JCL includes a steplib that contains IDMSB102
- The batch program is linked with either IDMS (Release 10.2) or IDML
- If the batch program has been relinked with a later version of the IDMS module, it must be relinked with either the 10.2 IDMS module or the IDML module

Note: A link with IDMSINTB is supported for upward compatibility but is neither required nor recommended for using the 10.2 batch services interface.

- DBNAME must be specified. Since SYSIDMS parameters are not supported through this interface, you can do one of the following:
 - Modify the BIND RUN-UNIT statements in the program to specify the DBNAME parameter; for example:

```
BIND RUN-UNIT DBNAME EMPDEMO.
```

- Update the DBTABLE within the central version to utilize subschema mapping and default dbname parameters; for example:

```
DBNAME *DEFAULT  
  SUBSCHEMA EMPSS?? MAPS TO EMPSS?? USING DBNAME EMPDEMO;
```

►► For more information about updating the DBTABLE, refer to *CA-IDMS Database Administration*.

Batch execution considerations: Be aware of these considerations when preparing a program to use one of the batch interfaces:

- If the batch program is linked with either IDMS (Release 10.2) or IDML, and IDMSB102 is in a batch program JCL steplib, the 10.2 services batch interface will *always* be used.

- If the batch program is linked with any later version of the IDMS module, the 10.2 services batch interface will *never* be used, even if IDMSB102 is in a batch program JCL steplib.
- To use a later versions of the batch interface with an existing 10.2 program, be sure that IDMSB102 is not in a batch program JCL steplib.
- If the COBOL program has been compiled with the DYNAM option, you must rename the IDML module to IDMS and place it in a separate library. This library must be the first library after the STEPLIB concatenation.
- If signon security is in effect, a valid user ID must be provided for the batch user exit BTCIDXIT, which allows specification of the user ID to be checked by security. A sample BTCIDXIT may be found in the distribution source library.

Index

A

ABEND 9-37
abend control element 12-9
ACCEPT (DC) 9-9—9-10
ACCEPT statements
 BIND ADDRESS 4-30
 DATABASE-STATISTICS 2-30—2-31
 db-key 4-24—4-28
 general discussion 4-24
 page information 4-28
 PROCEDURE CONTROL
 LOCATION 10-10—10-11
access modes 2-17
access restrictions 2-8—2-9
ACE
 See abend control element
AID 6-6, 7-8, 7-9, 9-5, 10-7
 definition 6-6
area
 database pages 2-5—2-6
 general discussion 2-5
area locks 2-13, 2-17
area sweep 4-10—4-12, 4-15
Assembler B-3
 batch error checking B-3
 DC error checking B-3—B-4
asynchronous 10-6
attention identifier key
 See AID
attribute byte 7-6
 definition 7-5
attributes 11-6
automatic editing
 See online mapping, automatic editing
automatic set membership 3-11, 4-35
AUTOSTATUS 6-12

B

BAL
 See Assembler
basic mode 10-6—10-8
batch execution 11-6, G-3
batch mapping compiler 7-4
batch, 10.2 services batch interface G-4
batch, 10.22 services batch interface G-3

bill of materials 5-6
 retrieving 5-8—5-10
 storing 5-6—5-7
BIND MAP 2-28
BIND MAP RECORD 2-28
BIND PROCEDURE 10-9—10-10
BIND RECORD 2-26
BIND RUN-UNIT 2-25—2-26, G-3
BIND RUN-UNIT statement 2-29
BIND RUN_UNIT 11-6
BIND statements
 BIND MAP 2-28
 BIND MAP RECORD 2-28
 BIND PROCEDURE 10-9—10-10
 BIND RECORD 2-26
 BIND RUN-UNIT 2-25—2-26
 BIND TASK C-3
 COPY IDMS SUBSCHEMA-BINDS 2-27
BL
 See base locator
BLL
 See base locator for linkage
BTCIDXIT user exit G-4
buffers 2-4

C

CA-ADS 10-4—10-5
 See also CA-ADS
CA-OLQ 12-6
CALC 3-5
 See also FIND/OBTAIN statements, CALC
central version 11-5, 11-6
 access modes 2-17
 area locks 2-17
 concurrent area use 2-19
 database name table 11-7
 definition 1-4
 implicit record locks 2-19
chained sets
 See sets, chained
CHANGE PRIORITY 9-34
CHECK TERMINAL 10-8
checkpoints 2-20
clist
 See command list
clustering 3-6

COBOL
 IDMS-STATUS for BATCH 2-33
 IDMS-STATUS for DC 6-12
 VS COBOL II D-3
 command list 11-10
 COMMIT
 frequency of use 2-22
 TASK 8-24, C-3
 variable-length records 3-5
 common errors
 See error checking
 common system area 12-10
 common work area 8-11
 communications blocks
 DC 6-11—6-12
 general discussion 2-29
 IDMS 2-32—3-30
 compiler directives
 COPY IDMS 2-25
 operating modes 2-24
 subschema 2-24
 compiling 11-4
 CONNECT 4-44—4-45
 connect options
 See sets, membership options
 control statements
 BIND/READY/FINISH 2-25—2-28
 COMMIT 2-20
 ROLLBACK 2-29
 conversational programming 6-6
 COPY IDMS SUBSCHEMA-BINDS 2-27
 CSA
 See common system area
 currency 3-22—3-26
 general discussion 2-30
 queue records, in 8-23
 scratch records, in 8-17
 CWA
 See common work area

D
 data description language (DDL) 2-4
 data integrity
 See locks
 data structure diagram 3-20—3-21
 sample 3-20
 database administrator
 database description 2-4
 database key 3-22, 3-29, 8-27
 retrieving records 4-15—4-18
 database key (*continued*)
 saving 4-24—4-28
 update example 7-27
 database keys 2-11
 database name table 11-7
 database pages 2-5—2-6
 database procedures 10-9—10-11
 ACCEPT PROCEDURE CONTROL
 LOCATION 10-10—10-11
 BIND PROCEDURE 10-9—10-10
 database retrieval
 See retrieval statements
 database statistics 2-30—2-31
 date F-3—F-5
 See also DC, current time and date
 DB programming
 See navigational DML programming
 db-key
 See database key
 db-key deadlocks
 DBNAME 11-6
 DC
 abend processing 9-37—9-40
 as an operating system 6-4
 Assembler considerations B-3—B-4
 communications block 6-11—6-12
 current time and date 9-21—9-22
 data integrity 9-11—9-17
 DC-BATCH C-3—C-4
 error handling 6-10
 event control blocks 9-41
 journal file 9-23—9-24
 messages 9-28—9-30
 modifying task priority 9-34
 nonterminal tasks 9-35—9-36
 programming techniques 9-3—9-41
 pseudoconversational programming 6-6
 response time 6-9
 statistics 9-25—9-27
 table management 9-18—9-20
 task priority 9-34
 task-related information 9-9—9-10
 tasks 6-5
 terminal management 7-3—7-33, 10-6—10-8
 transactions 6-5
 writing to a printer 9-31—9-32
 DC printer
 See WRITE PRINTER
 DC RETURN 6-6, 9-5
 DC-BATCH 2-29, 8-22, 9-36, C-3—C-4

DCE
 See dispatch control element

DCMT
 VARY DYNAMIC PROGRAM 9-18, 11-10—11-11
 VARY DYNAMIC TASK 11-10—11-11
 VARY PROGRAM NEW COPY 11-10
 VARY QUEUE 8-23

DCUF
 SET DBNAME 11-9
 SET DICTNAME 11-11
 TEST 11-11

DCUF statements 11-6

DDL
 See data description language (DDL)

DDLDCMSG 9-28

DDLDCRUN 8-22

DDLDCSCR 8-15

deadlock prevention 10-13

debugging 12-3—12-13
 CA-OLQ menu facility 12-6
 dump reading 12-7—12-11
 error checking 12-12—12-13
 PL/I A-5—A-8
 using the trace facility 12-4

device-media control language (DMCL) 2-4

dialog
 See CA-ADS

dictionary name 2-29

dictionary node 2-29

DICTNAME 11-6

DIRECT location mode
 storing 4-35
 with FIND/OBTAIN DB-KEY 4-15—4-18

DISCONNECT 4-45—4-47

disconnect options
 See sets, membership options

dispatch control element 12-10

DMCL
 See device-media control language (DMCL)

DML trace facility
 See trace facility

DMLTRACE F-3—F-5
 See also trace facility

dump reading 12-7—12-11

duplicates option
 See records, duplicates option

dynamic program definition 11-10

dynamic task definition 11-10

E

ECB
 See event control block

ERASE 4-15, 4-39—4-44

error checking 12-12—12-13

error handling 6-10
 See also INQUIRE MAP

error messages
 for maps 7-13
 suppressing display of 7-13

establishing communications 2-24—2-28

event control block 9-41, 10-8

extended architecture
 See XA considerations

extended run unit
 See CA-ADS

F

FIND/OBTAIN statements
 CALC 4-4—4-5
 CURRENT 4-14—4-15
 general discussion 4-4
 OWNER 4-12—4-14
 USING DB-KEY 4-15—4-18
 WITHIN AREA 4-10—4-12
 WITHIN SET 4-5—4-6
 WITHIN SET USING SORT KEY 4-6—4-10

FINISH 2-26
 TASK C-4

fragments 4-21

G

generic key searches 4-9

generic-key search 4-20

GET 4-21—4-23

GET TIME 9-21—9-22

I

I
 See sets, linkage

identical data
 testing for 7-14

IDML load module G-3

IDMS communications block 2-32—2-33, 6-11—6-12
 when used 2-29

IDMS statistics block 2-23

IDMS-STATUS
 explanation 2-32

IDMS-STATUS (*continued*)

figure (batch) 2-33

figure (DC) 6-12

under DC 6-11

IDMSB102 load module G-3

IDMSIN01 F-3—F-5

IDMSIN01 entry point 11-6, 12-5

IDMSRPTS utility

general discussion 2-9—2-10

sample JCL 2-10

IF 4-31—4-34

IF EMPTY 4-31—4-32

IF MEMBER 4-32—4-34

indexed sets 3-8, 3-9

See also sets, indexed

retrieval commands 4-18—4-19

INQUIRE MAP 7-8—7-11

INQUIRE MAP statement

DIFFERENT parameter 7-14

IDENTICAL parameter 7-14

IO

See sets, linkage

J

journal file 9-23—9-24

K

KEEP 4-48—4-49

KEEP LONGTERM 9-11—9-17

explicit locks 9-11—9-14

monitoring records 9-14—9-17

L

line mode 7-30—7-33

LINK

See TRANSFER CONTROL, LINK

link editing 11-4

linkage options

See sets, linkage

load library

local mode 11-5

area locks 2-17

location mode

See records, location mode

locks

area 2-13

DC 9-11—9-17

effect on run units 2-15—2-16

exclusive 2-13

locks (*continued*)

explicit 2-14

explicit (online) 9-11—9-14

implicit 2-14, 2-22, 2-23

longterm locks 9-14

record 2-13—2-16

shared 2-13

logical terminal element 8-6, 8-10, 8-12, 8-15, 12-10

LTE

See logical terminal element

M

mandatory set membership 3-10

manual set membership 3-11, 4-32—4-34, 4-35,
4-44—4-45

MAP IN 7-8

MAP OUT 7-5—7-7

MAP OUTIN 7-12—7-13

map request block 7-5

mapping mode

asynchronous output requests 10-8

detail area 7-15

footer area 7-15

general discussion 7-4—7-5

header area 7-15

housekeeping statements 7-5

INQUIRE MAP 7-8—7-11

MAP IN 7-8

MAP OUT 7-5—7-7

MAP OUTIN 7-12—7-13

MODIFY MAP 7-11—7-12

maps

error messages for 7-13

MDT

See modified data tag

member 3-8

membership options

See sets, membership options

messages 9-28—9-30

with maps 7-6

mixed page groups 4-17

modified data tag 7-19

definition 7-5

retrieval example 7-20—7-22

update example 7-23

MODIFY 4-37—4-39

MODIFY MAP 7-11—7-12

MRB

See map request block

N

N

See sets, linkage

navigational DML programming 3-3—3-30

currency 3-22—3-26

data structure diagram 3-20—3-21

execution sequence 3-27—3-30

introduction 3-3

records 3-4—3-7

sets 3-8—3-19

navigational DML statements 1-5

NO

See sets, linkage

NODENAME 11-6

nonterminal tasks

See also DC, nonterminal tasks

ATTACH 9-35

external request 9-36

queue threshold 9-36

SET TIMER 9-36

NP

See sets, linkage

NPO

See sets, linkage

O

O

See sets, linkage

OBTAIN

See FIND/OBTAIN statements

ON clause (AUTOSTATUS) 6-12

online debugger

PL/I considerations A-5—A-8

sample PL/I session A-6—A-8

online mapping 7-4

automatic editing 7-9

online programming

introduction 6-3

operating modes 2-24

DC-BATCH 9-36

optional set membership 3-10, 4-32—4-34,

4-44—4-45—4-47

order options

See sets, order options

owner 3-8, 3-10

P

P

See sets, linkage

PAGE-INFO parameter 4-16

PDE

See program definition element

PL/I

online debugger A-5—A-8

TRANSFER CONTROL A-4

precompiler 1-4, 11-4

preprocessor

See precompiler

print queues

See queues, print

priority

See DC, task priority

procedures

See database procedures

profile attributes 11-6

profiles F-3—F-5

program definition element 12-10

program management 9-4—9-8

program pools

tables 9-18—9-20

program registration 2-9, 2-25

pseudoconversational programming 6-6

PXE

See program expansion element

Q

queue management 8-22—8-26

COMMIT TASK 8-24

deleting 8-23

header record 8-22

locking 8-24

retention period 8-23

retrieving 8-23

storing 8-22

queue records

See queue management

queued resources 10-12—10-13

queues

print 9-31

R

READY 2-26

READY considerations 2-26

ready options

See usage modes, ready options

record
 locks 2-13—2-16, 2-19
 occurrence 2-7
 type 2-7

record locks 9-11—9-14

records
 area name 3-7
 duplicates option 3-6—3-7
 identification 3-5
 length 3-5
 location mode 3-5—3-6
 name 3-4
 storage mode 3-5
 type 3-4

records, location mode
 DIRECT 4-35

recovery units 2-20—2-23

redefining records
 See synonyms

Release 10.2 G-3

resources
 See queued resources

response time 6-9

restricting access
 area usage mode 2-18—2-19
 KEEP 4-48—4-49
 KEEP LONGTERM 9-11—9-14

retention period
 See queue management, retention period

retrieval statements
 FIND/OBTAIN CALC 4-4—4-5
 FIND/OBTAIN CURRENT 4-14—4-15
 FIND/OBTAIN OWNER 4-12—4-14
 FIND/OBTAIN USING DB-KEY 4-15—4-18
 FIND/OBTAIN WITHIN AREA 4-10—4-12
 FIND/OBTAIN WITHIN SET 4-5—4-6
 FIND/OBTAIN WITHIN SET USING SORT
 KEY 4-6—4-10
 general discussion 4-4
 GET 4-21—4-23
 indexed records 4-18—4-20
 indexed sets 4-18—4-19
 RETURN 4-19—4-20

RETURN 4-19—4-20

ROLLBACK B-3
 TASK C-3

run unit
 definition 2-13
 establishing 2-25—2-28
 general discussion 2-13
 specifying a dictionary name for 2-29

run unit (*continued*)
 specifying a dictionary node for 2-29
 terminated from db-key deadlocks 9-37

S

save statements
 See ACCEPT statements

saving I/O
 FIND/GET 4-21
 FIND/OBTAIN DB-KEY 4-15—4-18
 IF 4-31—4-34
 RETURN 4-19—4-20

saving page information 4-28

schema
 general discussion 2-4

scratch area 8-16

scratch management 8-15
 logical terminal element 8-15
 record ID 8-16
 task control element 8-15

scratch record ID 8-16

scratch records
 See scratch management

secondary dictionary 11-11

SEND MESSAGE 9-29—9-30

session attributes 11-6

SET ABEND EXIT 9-39—9-40

sets 3-8—3-19
 chained 3-8
 indexed 3-8, 3-9
 linkage 3-9—3-10
 membership options 3-10—3-11
 order options 3-11—3-12
 set name 3-9
 sorted 4-6—4-10

signon processing 11-6

signon security G-4

SNA
 See System Network Architecture

sort keys
 contiguous 4-6
 noncontiguous 4-7
 retrieving sorted records 4-6—4-10
 RETURN 4-19—4-20

SRID
 See scratch record ID

STAE
 See SET ABEND EXIT

statistics
 database 2-30—2-31

statistics (*continued*)
DC 9-25—9-27

storage management 8-4—8-14
shared 8-9—8-10
shared kept 8-10—8-11
user 8-5—8-6
user kept 8-6—8-9

storage mode
See records, storage mode

storage pools
See storage management

STORE 4-35—4-37

subroutines
ACCEPT BIND ADDRESS 4-30

subschema
access restrictions 2-8—2-9
default usage modes 2-19
general discussion 2-8
mapping G-3
overriding 11-7—11-9
program registration 2-9, 2-25
subschema 2-8

sweep
See area sweep

symbolic key 2-7
CALC 3-5
contiguous 2-7, 4-6
noncontiguous 2-7, 4-7

synchronous 10-6

synonyms 5-4—5-5

SYSTEMS parameters 11-6, 12-4

System Network Architecture 10-6

system-owned index 3-8

T

tables
See DC, tables

task code 9-9—9-10

task control element 8-4, 8-5, 8-6, 8-12, 8-15

task dump
See dump reading

task priority
See DC, task priority

tasks 6-5

TCE
See task control element

TCF
See transfer control facility

terminal management 7-3—7-33
basic mode 10-6—10-8

terminal management (*continued*)
line mode 7-30—7-33
mapping mode 7-4—7-14

test 11-10

test load library 11-10

time
See DC, current time and date

trace facility 12-4

transaction statistics block 9-25

transactions 6-5

TRANSFER CONTROL
LINK 9-7—9-8
LINK from CA-ADS 10-4—10-5
LINK under TCF E-4
PL/I considerations A-4
XCTL 9-6—9-7

transfer control facility E-3—E-19
sample program E-15

TSB
See transaction statistics block

U

UCE
See universal communications element

universal communications element E-4

updating the database
connecting records 4-44—4-45
disconnecting records 4-45—4-47
erasing records 4-39—4-44
general discussion 4-35
modifying records 4-37—4-39
storing records 4-35—4-37

usage modes
area 2-18—2-19
ready options 2-18

V

VIA 3-6

VSAM
CONNECT restriction 4-45
DISCONNECT restriction 4-45
MODIFY (ESDS and KSDS) 4-38
restrictions with FIND/OBTAIN DB-KEY 4-16
restrictions with IF 4-31
set order 3-12
STORE (RRDS) 4-35

W

walking a set 3-8, 4-5—4-6

WCC

See write control character

write control character 7-11

definition 7-5

WRITE JOURNAL 9-23—9-24

WRITE LOG 9-28—9-29

WRITE PRINTER 9-31—9-32

DC-BATCH C-4

X

XA considerations

assembler D-3

VS COBOL II D-3

XCTL

See TRANSFER CONTROL, XCTL

